Event Detection, version 2

Deliverable D4.2.2

Version DRAFT

Authors: Rodrigo Agerri[1], Itziar Aldabe[1], Zuhaitz Beloki[1], Egoitz Laparra[1], Maddalen Lopez de Lacalle[1], German Rigau[1], Aitor Soroa[1], Antske Fokkens[2], Ruben Izquierdo[2], Marieke van Erp[2], Piek Vossen[2], Christian Girardi[3], Anne-Lyse Minard[3]

Affiliation: (1) EHU, (2) VUA, (3) FBK

Building structured event indexes of large volumes of financial and economic data for decision making

ICT 316404

Grant Agreement No.	316404
Project Acronym	NEWSREADER
Project Full Title	Building structured event indexes of large volumes of financial and economic data for decision making.
Funding Scheme	FP7-ICT-2011-8
Project Website	http://www.newsreader-project.eu/
Project Coordinator	Prof. dr. Piek T.J.M. Vossen VU University Amsterdam Tel. + 31 (0) 20 5986466 Fax. + 31 (0) 20 5986500 Email: piek.vossen@vu.nl
Document Number	Deliverable D4.2.2
Status & Version	DRAFT
Contractual Date of Delivery	September 2014
Actual Date of Delivery	October 3, 2014
Type	Report
Security (distribution level)	Public
Number of Pages	57
WP Contributing to the Deliverable	WP4
WP Responsible	EHU
EC Project Officer	Susan Fraser

Authors: Rodrigo Agerri[1], Itziar Aldabe[1], Zuhaitz Beloki[1], Egoitz Laparra[1], Maddalen Lopez de Lacalle[1], German Rigau[1], Aitor Soroa[1], Antske Fokkens[2], Ruben Izquierdo[2], Marieke van Erp[2], Piek Vossen[2], Christian Girardi[3], Anne-Lyse Minard[3]

Keywords: Event detection, EN pipelines, NL pipeline, ES pipeline, IT pipeline, Scaling of text processing

Abstract: This deliverable describes the second prototype for event detection. It focuses on English, Dutch, Italian and Spanish. It uses an open architecture which works with generic NLP modules that perform different tasks for event detection. Each task is executed by one module, which allows custom pipelines to be used for text processing.

Table of Revisions

Version	Date	Description and reason	By	Affected sections
0.1	01 September 2014	Structure of the deliverable set	EHU	All
0.1	25 September 2014	Dutch, Italian, Spanish pipelines added	EHU, FBK, VUA	3, 4, 5
0.1	30 September 2014	Event detection section added	EHU	2
0.1	30 September 2014	Draft of the deliverable set	EHU, FBK, VUA	All

Executive Summary

This deliverable describes the second cycle of event detection, developed within the European FP7-ICT-316404 "Building structured event indexes of large volumes of financial and economic data for decision making (NewsReader)" project. The prototype and results presented are part of the activities performed in tasks T4.2 Event Detection, T4.3 Authority and factuality computation and T4.5 Scaling of text processing of Work Package WP4 (Event Detection).

The second prototype on event detection includes improved and new modules in the English pipeline. We have improved the modules that perform tokenization, POS-tagging, parsing, time recognition, named entity recognition, word sense disambiguation, named entity disambiguation, coreference resolution, semantic role labeling, event classification, and opinion mining. We have integrated new modules to perform temporal and causal relation extraction. We have also implemented a new module for text classification. Each task is executed by one module, which allows us to custom different pipeline topologies for text processing.

In the second cycle of event detection, we have worked on the adaptation of the English pipeline to the financial domain. We have also developed Dutch, Italian and Spanish pipelines. So far, we have focused on generic NLP modules that perform the necessary tasks for event detection. The multilingual interoperable semantic interpretation of the information has also been studied.

Finally, we have continued collecting and processing different datasets. Five datasets covering different topics have been processed by the NLP pipeline. Together these sets consists of approximately 326K articles in English and 7K articles in Spanish.

Contents

1 Introduction

This deliverable describes the second version of the **Event Detection** framework developed in NewsReader to process large and continuous streams of English, Dutch, Spanish and Italian news articles. In this period, we have worked on the improvement of the systems for event detection. Event detection addresses the development of text processing modules that detect mentions of events, participants, their roles and the time and place expressions in the four project languages.

NewsReader uses an open and modular architecture for Natural Language Processing (NLP) as a starting point. The system uses the NLP Annotation Framework[1] [?] (NAF) as a layered annotation format for text that can be shared across languages, and separate modules have been developed to add new interpretation layers using the output of previous layers. Text-processing requires basic and generic NLP steps such as tokenization, lemmatization, part-of-speech tagging, parsing, word sense disambiguation, named-entity and semantic role recognition, etc. for all the languages within the project.

Semantic interpretation involves the detection of event mentions and those named entities that play a role in these events, including time and location relations. This implies covering all expressions and meanings that can refer to events, their participating named entities, place and time relations. It also means to resolve coreference relations for these named entities and relations between different event mentions. As a result of this process, the text is enriched with semantic concepts and identifiers that can be used to access lexical resources and ontologies. For each unique event, we will also derive its factuality score based on the textual properties and its provenance.

Moreover, in order to achieve cross-lingual semantic interoperability, entity and event mentions should be projected to language independent knowledge representations. Thus, named entities are linked as much as possible to external sources such as DBpedia entity identifiers while event mentions are aligned to abstract event representations thanks to the Predicate Matrix [?]. We are also developing new techniques and resources to achieve interoperable semantic interpretation for English, Dutch, Spanish and Italian thanks to DBpedia cross-lingual links and multilingual semantic projections through local wordnets of the Predicate Matrix. In the second year, we have continued improving existing modules and creating new ones.

NewsReader provides an abstraction layer for large-scale distributed computations, separating the what from the how of computation and isolating NLP developers from the details of concurrent programming.

This deliverable presents the main NLP processing modules for English, Dutch, Italian and Spanish addressed by the NewsReader project in order to process event across documents. The evaluation results of the pipelines will be included in deliverable D3.3.2 Annotated data.

The remainder of the document consists of the following sections. Section 2 presents the main NLP processing modules for English. Sections 3, 4 and 5 describe the Dutch,

[1] `http://wordpress.let.vupr.nl/naf`

Italian and Spanish processing pipelines, respectively. Section 6 presents a process applied to all the pipelines in which the topic of each document is detected.Section 7 describes the processed data in this second year. Finally, Section 8 presents the main conclusions of this deliverable.

2 Event Detection

This section introduces the main NLP tasks addressed by the NewsReader project in order to process events across documents in four different languages: English, Dutch, Spanish and Italian. NewsReader Deliverable D4.1[2] provides a detailed survey about the current availability of resources and tools to perform event detection for the four languages involved in the project.

Event Detection (WP04) addresses the development of text processing modules that detect mentions of events, participants, their roles and the time and place expressions. Thus, text-processing requires basic and generic NLP steps, such as tokenization, lemmatization, part-of-speech tagging, parsing, word sense disambiguation, named entity and semantic role recognition for all the languages in NewsReader. Named entities are as much as possible linked to external sources (Wikipedia, DBpedia, JRC-Names, BabelNet, Freebase, etc.) and entity identifiers. Furthermore, event detection involves the identification of event mentions, event participants, the temporal constraints and, if relevant, the location. It also implies the detection of expressions of factuality of event mentions and the authority of the source of each event mention.

Moreover, NewsReader is developping:

- new techniques for achieving interoperable Semantic Interpretation of English, Dutch, Spanish and Italian
- wide-coverage linguistic processors adapted to the financial domain
- new scaling infrastructures for advanced NLP processing of large and continuous streams of English, Dutch, Spanish and Italian news articles.

During the second cycle of the NewsReader project (Event Detection, version 2) we focused on improving the existing modules of the English pipeline presented in NewsReader Deliverable 4.2.2[3] as the "IXA-pipeline". We have also worked on the adaptation of the pipeline to the financial domain. By the end of the year, we will evaluate and improve the event detection system based on the benchmark evaluations that are planned in WP03.

In the second year, we have worked on generic pipelines with new functionalities for Dutch, Italian and Spanish. These pipelines could also be evaluated by the gold-standards generated in WP03.

[2]`http://www.newsreader-project.eu/files/2012/12/NewsReader-316404-D4.1.pdf`

[3]`http://www.newsreader-project.eu/files/2012/12/NewsReader-316404-D4.2.pdf`

Although we already use NAF to harmonize the different outcomes, during the first cycle of the project, we detected the need to stablish a common semantic framework for representing the event mentions. For instance, SEMAFOR[4] uses FrameNet [?] for semantic role labelling (SRL), while Mate-tools[5] uses PropBank [?] for the same task. Additionaly, as a backup solution, we are also processing the text with Word Sense Disambiguation modules to match WordNet [?] identifiers across predicate models and languages.

To allow interoperable semantic interpretation of texts in multiple languages and predicate models, we started the development of the *Predicate Matrix*, a new lexical resource resulting from the integration of multiple sources of predicate information including FrameNet [?], VerbNet [?], PropBank [?] and WordNet [?]. By using the Predicate Matrix, we provide a more robust interoperable lexicon by discovering and solving inherent inconsistencies among the resources. We plan to extend the coverage of current predicate resources (by including from WordNet morphologically related nominal and verbal concepts), to enrich WordNet with predicate information, and possibly to extend predicate information to languages other than English (by exploiting the local wordnets aligned to the English WordNet). The first version was integrated in the first prototype of the event detection system, and during this second cycle, we have integrated a new *Predicate Matrix*. In addition, we have worked with the dbpedia as a resource to allow a better interoperable semantic interpretation.

3 English NLP Processing

The descriptions of the modules are provided along with some technical information. This technical information includes: a) the description of the input and output that the modules require and obtain; b) the dependencies with other modules and third-party modules and libraries; c) level of operation of the module; d) if the module is language dependent or not; e) the required resources for a correct functioning of the module; f) the possible formats the module works with and g) the github address of the module. A complete specification of the output attributes and elements is provided in Appendix A.

3.1 Tokenizer

- **Module**: ixa-pipe-tok
- **Description of the module**: This module provides Sentence Segmentation and Tokenization for English and Spanish and other languages such as Dutch, German, English, French, Italian and Spanish. It implements a rule-based segmenter and tokenizer originally inspired by the Stanford English Penn Treebank tokenizer[6] but with several modifications and improvements. These include tokenization for other

[4]`http://code.google.com/p/semafor-semantic-parser/wiki/FrameNet`
[5]`http://code.google.com/p/mate-tools/`
[6]`http://www-nlp.stanford.edu/software/tokenizer.shtml`

languages such as Italian, normalization according the Spanish Ancora Corpus [?], paragraph treatment, and more comprehensive gazeteers of non breaking prefixes. The tokenizer depends on a JFlex[7] specification file which compiles in seconds and performs at a very reasonable speed (around 250K word/second, and much quicker with Java multithreading). JFlex is a lexical analyser generator. The module is part of the IXA pipes [?],[8] a modular set of Natural Language Processing tools (or pipes) which provide easy access to NLP technology for English and Spanish.

- **Input**: Raw text
- **Input representation**: NAF raw layer
- **Output**: Tokens and sentences.
- **Output representation**: NAF text layer
- **Required modules**: None
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: None
- **Dependencies**: Java, Maven, NAF Java library, JFlex
- **Flexible in- and output**: It takes plain text or raw text in NAF. It produces tokenized and segmented text in NAF, running text and CoNLL formats.
- **github address**: `https://github.com/newsreader/ixa-pipes`

3.2 POS tagging

- **Module**: ixa-pipe-pos
- **Description of the module**: This module provides POS tagging and lemmatization for English and Spanish. The module is part of the IXA pipes. We have obtained the best results so far with *Perceptron* models and the same featureset as in [?]. The models have been trained and evaluated on the WSJ treebank using the usual partitions (e.g., as explained in [?]. We currently obtain a performance of 96.88% vs 97.24% in word accuracy obtained by [?].

  Lemmatization is currently performed via 3 different dictionary lookup methods: a) *Simple Lemmatizer*: It is based on HashMap lookups on a plain text dictionary.

[7]`http://jflex.de/`
[8]`http://ixa2.si.ehu.es/ixa-pipes/`

Currently we use dictionaries from the LanguageTool project[9] under their distribution licenses; b) Morfologik-stemming:[10] The Morfologik library provides routines to produce binary dictionaries, from dictionaries such as the one used by the Simple Lemmatizer above, as finite state automata. This method is convenient whenever lookups on very large dictionaries are required because it reduces the memory footprint to 10% of the memory required for the equivalent plain text dictionary; and c) We also provide lemmatization by lookup in WordNet-3.0 [?] via the JWNL API.[11] By default, the module accepts tokenized text in NAF format as standard input and outputs NAF.

- **Input**: Tokens
- **Input representation**: NAF text layer
- **Output**: Lemmas and POS-tags
- **Output representation**: NAF terms layer
- **Required modules**: Tokenizer module
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: POS model; Lemmatizer dictionaries: plain text dictionary and morfologik-stemming dictionary.
- **Dependencies**: Java, Maven, NAF Java library, JWNL API, Apache OpenNLP.
- **Flexible in- and output**: It accepts tokenized text in NAF. It outputs NAF or CoNLL formats.
- **github address**: `https://github.com/newsreader/ixa-pipes`

3.3 Constituency Parser

- **Module**: ixa-pipe-parse
- **Description of the module**: This module provides statistical constituent parsing for English and Spanish. The module is part of the IXA pipes. Maximum Entropy models are trained to build shift reduce bottom up parsers [?] as provided by the Apache OpenNLP Machine Learning API. Parsing models for English have been trained using the Penn treebank. Furthermore, ixa-pipe-parse provides two methods

[9]`http://languagetool.org/`
[10]`https://github.com/morfologik/morfologik-stemming`
[11]`http://jwordnet.sourceforge.net/`

of headword finders: one based on Collins' head rules as defined in his PhD thesis [?], and another one based on Stanford's parser Semantic Head Rules.[12] The latter are a modification of Collins' head rules according to lexical and semantic criteria. We obtain a F1 87.42%. The module accepts lemmatized and POS tagged text in NAF format as standard input and outputs NAF.

- **Input**: Lemmatized and POS tagged text
- **Input representation**: NAF terms layer
- **Output**: Constituents; Syntactic tree of sentences.
- **Output representation**: NAF constituency layer
- **Required modules**: Tokenizer and POS tagger modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: Parsing model
- **Dependencies**: Java, Maven, NAF Java library, Apache OpenNLP
- **Flexible in- and output**: It accepts lemmatized and POS tagged text in NAF format. In addition to NAF output, ixa-pipe-parse also allows to output the parse trees into Penn Treebank bracketing style.
- **github address**: `https://github.com/newsreader/ixa-pipes`

3.4 Dependency Parser

- **Module**: ixa-pipe-srl
- **Description of the module**: This module is based on the MATE-tools [?], a pipeline of linguistic processors that performs lemmatization, part-of-speech tagging, dependency parsing, and semantic role labeling of a sentence. As the input of the module is a NAF file that includes lemmatization and pos-tagging, the module only implements the dependency parser [?]. The module is ready to work with Spanish and English. For the latter one, the dependency parser had the top score in the CoNLL shared task 2009, obtaining 90.24% Labeled attachment score (LAS).
- **Input**: Lemmatized and POS tagged text
- **Input representation**: NAF terms layer

[12]`http://www-nlp.stanford.edu/software/lex-parser.shtml`

- **Output**: Dependencies
- **Output representation**: NAF deps layer
- **Required modules**: Tokenizer and POS tagger modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: mate-tools package, dependency parsing model[13]
- **Dependencies**: Java, Maven, NAF Java library, mate-tools
- **Flexible in- and output**: It accepts lemmatized and POS tagged text in NAF format. The modules allows to output dependencies trees in NAF and CoNLL formats.
- **github address**: `https://github.com/newsreader/ixa-pipe-srl`

3.5 Time expression detection and normalization

- **Module**: fbk-timepro
- **Description of the module**: TimePro identifies the tokens corresponding to temporal expressions in English, assigns them to one of the 4 TIMEX classes defined in ISO-TimeML and normalizes them following TIDES specification ([?]). The temporal expressions recognizer is based on machine learning and it is trained on TempEval3 data. The average result for English is: 83.81% precision, 75.94% recall and 79.61% F1-measure values. The temporal expressions normalizer uses the library timenorm ([?]), enhanced by some pre-processing and post-processing for the selection of the best normalization value. Timenorm is shown to be the best performing system for most evaluation corpora (it obtained 81.6% F1-measure on TempEval3 test corpus) compared with other systems such as HeidelTime ([?]). This module has been integrated into TextPro pipeline but it also accepts a NAF input file with tokenization, POS-tagging and chunking information and provides a NAF file as output.
- **Input**: token layer, term layer with lemma, POS, entity and constituents
- **Input representation**: NAF terms layer
- **Output**: timex3
- **Output representation**: NAF timex layer
- **Required modules**: tokenizer, POS-tagger, NERC, Parser

[13]The module use additional resources to perform the semantic role labeling.

- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: language model, language dependent rules, English grammar for timenorm
- **Dependencies**: timenorm ([?]), YamCha[14]
- **Flexible in- and output**: use also TextPro format (i.e. column format)
- **github address**:

3.6 Named Entity Recognition and Classification

- **Module**: ixa-pipe-nerc
- **Description of the module**: This module is multilingual Named Entity Recognition and Classification tagger. ixa-pipe-nerc is part of IXA pipes. The named entity types are based on: a) the CONLL 2002[15] and 2003[16] tasks which were focused on language-independent supervised named entity recognition for four types of named entities: persons, locations, organizations and names of miscellaneous entities that do not belong to the previous three groups. We provide very fast models trained on local features only (84.53 F1), similar to those of [?] with several differences: We do not use POS tags, chunking or gazetteers in our baseline models but we do use bigrams, trigrams and character ngrams. We also provide some models with external knowledge (87.11 F1); b) the Ontonotes 4.0 dataset. We have trained our system on the full corpus with the 18 NE types, suitable for production use. We have also used 5K sentences at random for testset from the corpus and leaving the rest (90K aprox) for training. The Ontonotes CoNLL 4 NE types with local features model obtains F1 86.21. The Ontonotes 3 NE types with local features configuration obtains F1 89.41. The module allows to format its output in CoNLL style tabulated BIO format as specified in the CoNLL 2003 shared evaluation task.
- **Input**: Lemmatized and POS tagged text
- **Input representation**: NAF terms layer
- **Output**: Named entities
- **Output representation**: NAF entities layer
- **Required modules**: Tokenizer and POS tagger modules
- **Level of operation**: sentence level

[14]`http://chasen.org/~taku/software/yamcha/`
[15]`http://www.clips.ua.ac.be/conll2002/ner/`
[16]`http://www.clips.ua.ac.be/conll2003/ner/`

- **Language dependent**: yes
- **Resources**: CoNLL 2003 models, Ontonotes 4.0 models, properties file
- **Dependencies**: Java, Maven, NAF Java library, Apache OpenNLP
- **Flexible in- and output**: It accepts lemmatized and POS tagged text in NAF format. The modules allows to output dependencies trees in NAF and CoNLL formats.
- **github address**: `https://github.com/newsreader/ixa-pipes`

3.7 Word Sense Disambiguation

- **Module**: wsd-ukb
- **Description of the module**: UKB is a collection of programs for performing graph-based Word Sense Disambiguation. UKB applies the so-called Personalized PageRank on a Lexical Knowledge Base (LKB) to rank the vertices of the LKB and thus perform disambiguation. UKB has been developed by the IXA group. The module accepts lemmatized and POS tagged text in NAF format as standard input and outputs NAF.
- **Input**: Lemmatized and POS tagged text
- **Input representation**: NAF terms layer
- **Output**: Synsets
- **Output representation**: NAF terms layer
- **Required modules**: Tokenizer and POS tagger modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: English WordNet
- **Dependencies**: C++, boost libraries
- **Flexible in- and output**: no
- **github address**: `https://github.com/ixa-ehu/ukb`

3.8 Named Entity Disambiguation

- **Module**: ixa-pipe-ned
- **Description of the module**: This module performs the Named Entity Disambiguation task based on DBpedia Spotlight. Assuming that a DBpedia Spotlight Rest server for a given language is locally running, the module will take NAF as input (containing elements) and perform Named Entity Disambiguation. The module accepts text with named entities in NAF format as standard input, it disambiguates them and outputs them in NAF. The module offers the "disambiguate" and "candidates" service endpoints. The former takes the spotted text input and it returns the identifier for each entity. The later is similar to disambiguate, but returns a ranked list of candidates.
- **Input**: Named entities and sentences
- **Input representation**: NAF entities layer[17]
- **Output**: Disambiguated named entities
- **Output representation**: NAF entities layer
- **Required modules**: NERC module
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: DBpedia spotlight server
- **Dependencies**: Java, Maven, NAF Java library, DBpedia spotlight
- **Flexible in- and output**: no
- **github address**: `https://github.com/newsreader/ned-spotlight`

3.9 Coreference Resolution

- **Module**: corefgraph
- **Description of the module**: The module of coreference resolution included in the IXA pipeline is loosely based on the Stanford Multi Sieve Pass system [**?**]. The system consists of a number of rule-based sieves. Each sieve pass is applied in a deterministic manner, reusing the information generated by the previous sieve and the mention processing. The order in which the sieves are applied favours a highest

[17]Note: Linguistic annotations of a particular level always span elements of previous levels. In this particular case, the module also uses the terms layer to obtain the sentences of the given entities.

precision approach and aims at improving the recall with the subsequent application of each of the sieve passes. This is illustrated by the evaluation results of the CoNLL 2011 Coreference Evaluation task [?; ?], in which the Stanford's system obtained the best results. The results show a pattern which has also been shown in other results reported with other evaluation sets [?], namely, the fact that a large part of the performance of the multi pass sieve system is based on a set of significant sieves. Thus, this module focuses for the time being, on a subset of sieves only, namely, Speaker Match, Exact Match, Precise Constructs, Strict Head Match and Pronoun Match [?]. So far we have evaluated our module on the dev-auto part of the Ontonotes 4.0 corpus. We score 56.4 CoNLL F1, around 3 points worse than Stanford's system.

- **Input**: lemma, morphosyntactic information (morphofeat in NAF), named-entities, constituents
- **Input representation**: NAF entities, term, and constituency layers
- **Output**: coreferences
- **Output representation**: NAF coreferences layer
- **Required modules**: Tokenizer, POS-tagger and NERC modules
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: none
- **Dependencies**: pyKAF, pycorpus, networkx, pyYALM
- **Flexible in- and output**: It accepts lemmatized and POS tagged text, entities and constituents in NAF format. The modules allows to output coreference clusters in NAF and CoNLL formats.
- **github address**: `https://bitbucket.org/Josu/corefgraph`

3.10 Semantic Role Labeling

- **Module**: ixa-pipe-srl
- **Description of the module**: This module is based on the MATE-tools [?], a pipeline of linguistic processors that performs lemmatization, part-of-speech tagging, dependency parsing, and semantic role labeling of a sentence. They report on the CoNLL 2009 Shared Task a labelled semantic F1 of 85.63 for English and 79.91 for Spanish. As the input of the module is a NAF file that includes lemmatization, pos-tagging and dependency parsing, the module only implements the semantic role

labeler [?]. The module is ready to work with Spanish and English. By default, the module accepts parsed text in NAF format as standard input and outputs the enriched text in NAF.

- **Input**: Lemmatized and POS tagged text and syntactic dependencies
- **Input representation**: NAF terms, deps layers
- **Output**: Semantic roles
- **Output representation**: NAF srl layer
- **Required modules**: Tokenizer, POS tagger and Dependency parsing modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: mate-tools package, PredicateMatrix
- **Dependencies**: Java, Maven, NAF Java library, mate-tools
- **Flexible in- and output**: It accepts lemmatized and POS tagged text and syntactic dependencies in NAF format. The modules allows to output semantic roles in NAF and CoNLL formats.
- **github address**: `https://github.com/newsreader/ixa-pipe-srl`

3.11 Event coreference

- **Module**: vua-eventcoreference
- **Description of the module**: This module takes the predicates of the SRL layer as input and matches the predicates semantically. If the predicates are sufficiently similar, a coreference set is created with references to the predicates as coreferring expressions. If there is no match, predicates form a singleton set in the coreference layer.
- **Input**: SRL
- **Input representation**: SRL predicates
- **Output**: Coreference sets for events
- **Output representation**: NAF coref layer
- **Required modules**: ixa-pipe-srl
- **Level of operation**: sentence level

- **Language dependent**: yes
- **Resources**: wordnet-lmf
- **Dependencies**: Java, Maven, KAF/NAF Saxparser, WordnetTools
- **Flexible in- and output**: no
- **github address**: `https://github.com/cltl/EventCoreference`

3.12 Temporal relation extraction

- **Module**: fbk-temprel
- **Description of the module**: TempRelPro extracts and classifies temporal relations between two events, two time expressions, or an event and a time expression ([**?**]). The module is based on machine learning and is trained using yamcha tool on the TempEval3 data. It detects relations between: the document creation time and the main event of each sentence; the main events of two consecutive sentences; the time expressions and the events inside a sentence; all the events inside a sentence. All the events annotated by the Coreference module are considered and all the time expressions identified by the TimePro module. The result for relation classification (identification of the relation type given the relations) on TempEval3 test corpus is: 58.8% precision, 58,2% recall and 58,5% F1-measure. This module is part of TextPro pipeline, a multilingual NLP pipeline developed at FBK.
- **Input**: token layer, term layer with lemma, POS, entity, constituent, SRL, event coreference and time expression
- **Input representation**: NAF terms layer
- **Output**: tlink
- **Output representation**: NAF temporal relation layer
- **Required modules**: tokenizer, POS-tagger, NERC, Parser, ixa-pipe-srl, vua-eventcoreference, fbk-time
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: language model, language dependent rules
- **Dependencies**: Explicit Discourse Connectives Tagger ([**?**]), MorphoPro, YamCha [18]

[18] `http://chasen.org/~taku/software/yamcha/`

- **Flexible in- and output**: use also TextPro format (i.e. column format)
- **github address**:

3.13 Causal relation extraction

- **Module**: fbk-causalrel
- **Description of the module**: CausalRelPro extracts explicit causal relations between two events in the same sentence ([**?**]). All the events annotated by the Coreference module are considered. The module is based on machine learning and is trained using yamcha tool on the TimeBank corpus manually enriched with causal information (causal signals and causal relations). One model is trained for the annotation of causal signals (e.g. *as a result of*, *due to*) and another for the extraction of the causal links between events. The evaluation done on the test part of the corpus for the task of causal relation extraction gave a precision of 67.3%, a recall of 22.6% and a F1 measure of 33.9%. This module is part of TextPro pipeline, a multilingual NLP pipeline developed at FBK.
- **Input**: token layer, term layer with lemma, POS, entity, constituent, SRL, event coreference, time expression and temporal relation
- **Input representation**: NAF terms layer
- **Output**: tlink
- **Output representation**: NAF causal relation layer
- **Required modules**: tokenizer, POS-tagger, NERC, Parser, ixa-pipe-srl, vua-eventcoreference, fbk-time, fbk-temprel
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: language model, language dependent rules
- **Dependencies**: Explicit Discourse Connectives Tagger ([**?**]), MorphoPro, YamCha [19]
- **Flexible in- and output**: use also TextPro format (i.e. column format)
- **github address**:

[19] `http://chasen.org/~taku/software/yamcha/`

3.14 Factuality

- **Module**: VUA-factuality
- **Description of the module**: The factuality module classifies for each event whether it is factual or not. The module uses the MAchine Learning for LanguagE Toolkit Mallet [?] (version 2.0.7) trained on FactBank v1.0[20] to determine the factuality of an event in text. An alternative version of the module that uses rules instead of FactBank is currently under development.
- **Input**: Tokenized and POS-tagged text
- **Input representation**: NAF terms layer
- **Output**: Factuality layer with token spans
- **Output representation**: NAF factuality layer
- **Required modules**: tokenizer, pos-tagger modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: FactBank, Mallet
- **Dependencies**: Perl
- **Flexible in- and output**: no
- **github address**: `https://github.com/newsreader/Factuality-Classifier`

3.15 Opinions

- **Module**: opinion-miner
- **Description of the module**: This is a module that detects and extracts fine-grained opinions, where one single opinion contains three elements: the opinion expression (the subjective statement itself), the opinion target (what the opinion is about) and the opinion holder (who is stating the opinion). This module has been developed on the OpeNER project,[21] where it has been trained on different domains (hotel reviews, political news...), and for different languages (Dutch, English, Spanish, Italian, French and German) using corpora manually annotated also during the project. The extraction and tagging of opinions is divided into two steps. First, the detection of opinion entities (holder, target and expression) using Conditional Random Fields

[20] `http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2009T23`
[21] `http://www.opener-project.eu/`

using as features the tokens lemmas and POS on the context, as well as syntactic features and features extracted from lexicons. The second step is the opinion entity linking (expression¡-target and expression-¡holder) using binary Support Vector Machines. In this step all the single opinion entities detected are grouped into triples ¡expression,target,holder¿ according to the output of the SVM classifiers. In this case, besides the local context features, dependency features and features capturing the relative location of the opinion elements are included. The models have been trained with a rich set of features, but the opinion tagger can be used with a reduced subset of this features considering that the performance will be affected.

- **Input**: NAF text processed trough the pipeline, this module will use the information provided by all the rest
- **Input representation**: token, term, entity, dependency and constituency NAF layers
- **Output**: fine-grained opinion triples ¡expression, target, holder¿
- **Output representation**: NAF opinion layer
- **Required modules**: tokeniser, lemmatiser, POS tagger, polarity tagger, named entity tagger, constituency parser and dependency parser
- **Level of operation**: document
- **Language dependent**: yes
- **Resources**: models trained on the OpeNER project
- **Dependencies**: CRF library,[22] SVM-Light library,[23] KafNafParser[24] and VUA_pylib library[25]
- **Flexible in- and output**: it takes NAF or KAF input text, and generates NAF or KAF output text enriched with the extracted opinions
- **github address**: `https://github.com/cltl/opinion_miner_deluxe`

4 Dutch NLP Processing

4.1 Tokenizer

- **Module**: ixa-pipe-tok

[22] http://www.chokkan.org/software/crfsuite/
[23] http://www.chokkan.org/software/crfsuite/
[24] https://github.com/cltl/KafNafParserPy
[25] https://github.com/cltl/VUA_pylib

- **Description of the module**: This module provides Sentence Segmentation and Tokenization for English and Spanish and other languages such as Dutch, German, English, French, Italian and Spanish. It implements a rule-based segmenter and tokenizer originally inspired by the Stanford English Penn Treebank tokenizer[26] but with several modifications and improvements. These include tokenization for other languages such as Italian, normalization according the Spanish Ancora Corpus [?], paragraph treatment, and more comprehensive gazeteers of non breaking prefixes. The tokenizer depends on a JFlex[27] specification file which compiles in seconds and performs at a very reasonable speed (around 250K word/second, and much quicker with Java multithreading). JFlex is a lexical analyser generator. The module is part of the IXA pipes [?],[28] a modular set of Natural Language Processing tools (or pipes) which provide easy access to NLP technology for English and Spanish.
- **Input**: Raw text
- **Input representation**: NAF raw layer
- **Output**: Tokens and sentences.
- **Output representation**: NAF text layer
- **Required modules**: None
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: None
- **Dependencies**: Java, Maven, NAF Java library, JFlex
- **Flexible in- and output**: It takes plain text or raw text in NAF. It produces tokenized and segmented text in NAF, running text and CoNLL formats.
- **github address**: `https://github.com/newsreader/ixa-pipes`

4.2 POS tagging, lemmatization and parsing

- **Module**: vua-alpino
- **Description of the module**: This module performs morphosyntactic and dependency anaylisis of Dutch text. It is based on the Alpino parser,[29] which is a dependency and constituency parser for Dutch. Therefore this module is a wrapper around the

[26]`http://www-nlp.stanford.edu/software/tokenizer.shtml`
[27]`http://jflex.de/`
[28]`http://ixa2.si.ehu.es/ixa-pipes/`
[29]`http://www.let.rug.nl/vannoord/alp/Alpino/`

Alpino parser to deal with NAF files as input and output formats. Some special features of Alpino have been used in order to reduce the time of processing. It basically means that the parser is run just once to generate the XML files provided as output, and then these files are processed by different extractors to obtain all the information.

- **Input**: Tokens
- **Input representation**: NAF text layer
- **Output**: Lemmas and POS-tags
- **Output representation**: NAF term, constituency and dependency layer
- **Required modules**: Tokenizer module
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: None
- **Dependencies**: Python, NAF python parser and Alpino parser version of 17 June 2014 or later. Note that this version only runs on Linux.
- **Flexible in- and output**: Alpino is software developed by a third party. For more information on possible in- and output, checkout the Alpino website at `http://www.let.rug.nl/vannoord/alp/Alpino/`. Our wrapper accepts tokenized text in NAF or KAF. It outputs NAF or KAF formats.
- **github address**: Wrapper: `https://github.com/cltl/morphosyntactic_parser_nl`, Alpino (git address): `git://urd.let.rug.nl/Alpino.git`

4.3 Time Expressions

- **Module**: VUA-HeidelTime
- **Description of the module**: This module identifies time expressions in text and normalizes them. The core component of the module is HeidelTime [?], a temporal tagger supporting English, German, Dutch, Vietnamese, Arabic, Spanish, Italian, French, Chinese and Russian.[30] HeidelTime identifies temporal expressions based on language specific patterns. Identified temporal expressions are normalized and represented according to TIMEX annotations [?]. Dutch patterns have been written by Matje van de Camp for a corpus of short biographies [?]. HeidelTime is open source and can be used as a standalone module using TreeTagger or it can be integrated

[30]HeidelTime source code is available here: `https://code.google.com/p/heideltime/`

in UIMA architectures. The current version of the module provides a NAF wrapper around HeidelTime's standalone version. It can take the raw text layer or NAF token layer as input, calls HeidelTime standalone and creates a time expression layer as output. The disadvantage of this approach is that the standalone version requires running TreeTagger to obtain POS-tags and lemmas, which is freely available but cannot be redistributed. Moreover, our vua-alpino module already provides POS-tags and lemmas. An alternative version of the module that uses the NAF term layer as input is currently under development.

- **Input**: Raw text or tokens
- **Input representation**: NAF raw-text layer or NAF token layer
- **Output**: Normalized time expressions in timex
- **Output representation**: NAF time expression layer
- **Required modules**: Tokenizer producing NAF or KAF
- **Level of operation**: Document level
- **Language dependent**: yes (language specific rules and patterns)
- **Resources**: TreeTagger with resources for Dutch, language specific rules
- **Dependencies**: Python 2.7, TreeTagger for the correct operating system
- **Flexible in- and output**: HeidelTime is independently developed software that can either take raw text as input and produce marked-up text as output or be integrated in UIMA pipelines. Our wrapper can use NAF or KAF as in- and output.
- **github address**: Wrapper: `https://github.com/cltl/NAF-HeidelTime`

4.4 Named Entity Recognition and Classification

- **Module**: ixa-pipe-nerc
- **Description of the module**: This module is multilingual Named Entity Recognition and Classification tagger which is part of IXA pipes. The named entity types are based on the CONLL 2002[31] and 2003[32] tasks which were focused on language-independent supervised named entity recognition for four types of named entities: persons, locations, organizations and names of miscellaneous entities that do not belong to the previous three groups. We provide very fast models trained on local features only (84.53 F1), similar to those of Zhang and Johnson (2003) with several

[31]`http://www.clips.ua.ac.be/conll2002/ner/`
[32]`http://www.clips.ua.ac.be/conll2003/ner/`

differences: POS tags, chunking or gazetteers are not used in our baseline models but we do use bigrams, trigrams and character ngrams. We also provide some models with external knowledge (87.11 F1); Furthermore, the Ontonotes 4.0 dataset has been also considered. We have trained our system on the full corpus with the 18 NE types, suitable for production use. We have also used 5K sentences at random for testset from the corpus and leaving the rest (90K aprox) for training. The Ontonotes CoNLL 4 NE types with local features model obtains F1 86.21. The Ontonotes 3 NE types with local features configuration obtains F1 89.41. The module allows to format its output in CoNLL style tabulated BIO format as specified in the CoNLL 2003 shared evaluation task.

- **Input**: Lemmatized and POS tagged text
- **Input representation**: NAF tokend and term layers
- **Output**: Named entities
- **Output representation**: NAF entity layer
- **Required modules**: Tokenizer and POS tagger modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: CoNLL 2003 models, Ontonotes 4.0 models, properties file
- **Dependencies**: Java, Maven, NAF Java library, Apache OpenNLP
- **Flexible in- and output**: It accepts lemmatized and POS tagged text in NAF format. The modules allows to output dependencies trees in NAF and CoNLL formats.
- **github address**: `https://github.com/newsreader/ixa-pipes`

4.5 Word Sense Disambiguation

- **Module**: vua-wsd
- **Description of the module**: WSD-VUA is a machine learning system that performs Word Sense Disambiguation for Dutch text. It is based on a supervised machine learning approach: Suppor Vector Machines. The library selected for the low level machine learning engine is lib-svm[33]. A bag-of-words feature model is used for representing the training instances, where tokens and lemmas are considered to build the context for each target word. The training material used has been the corpus manually annotated on the DutchSemCor project.[34] One classifier is build for each target

[33]`https://github.com/cjlin1/libsvm`
[34]`http://www2.let.vu.nl/oz/cltl/dutchsemcor`

lemma, following the one-vs-all approach to deal with the multilabel classification of whe WSD task, as SVM is in principle a binary classifier. The relative frequency of a feature with respect to a certain classifier is considered to filter out to general features, and also for weighting each feature in the training phase. This classifier was evaluated by Cross-fold validation reaching an accuracy of 82.51 for nouns, 84.80 for verbs and 73.62 for adjectives.

- **Input**: tokenized, lemmatized and POS tagged text
- **Input representation**: NAF token and term layers
- **Output**: word sense labels assigned to terms
- **Output representation**: NAF external references in the term layer
- **Required modules**: Tokenizer and POS tagger modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: models trained on DutchSemCor
- **Dependencies**: lib-svm and KafNafParser python libraries
- **Flexible in- and output**: it accepts tokenized, lemmatized and POS tagged text in KAF/NAF format or plain text. The output can be KAF/NAF format of XML format as used in the SemCor corpus.
- **github address**: `https://github.com/cltl/svm_wsd`

4.6 PredicateMatrix tagging

- **Module**: vua-ontotagging
- **Description of the module**: This module takes a predicate matrix file with synsets and predicate matrix mappings and adds the predicate matrix mappings to the synsets listed in the term layer.
- **Input**: terms with wed output
- **Input representation**: term layer
- **Output**: terms with predicate matrix mappings
- **Output representation**: term layer with external references
- **Required modules**: vua-wsd

- **Level of operation**: term level
- **Language dependent**: yes
- **Resources**: PredicateMatrix
- **Dependencies**: Java, Maven, KAF/NAF Saxparser
- **Flexible in- and output**: NAF input stream and NAF output stream
- **github address**: `git@github.com:cltl/OntoTagger`

4.7 Named Entity Disambiguation

- **Module**: ixa-pipe-ned
- **Description of the module**: This module performs the Named Entity Disambiguation task based on DBpedia Spotlight. Assuming that a DBpedia Spotlight Rest server for a given language is locally running, the module will take NAF as input and will perform Named Entity Disambiguation. The module accepts text with named entities in NAF format as standard input, it disambiguates them and outputs them in NAF.
- **Input**: Named entities and sentences
- **Input representation**: NAF entities layer[35]
- **Output**: Disambiguated named entities with dbpedia links
- **Output representation**: dbpedia links on the NAF entity layer
- **Required modules**: NERC module
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: DBpedia spotlight server
- **Dependencies**: Java, Maven, NAF Java library, DBpedia spotlight
- **Flexible in- and output**: no
- **github address**: `https://github.com/newsreader/ned-spotlight`

[35]Note: Linguistic annotations of a particular level always span elements of previous levels. In this particular case, the module also uses the terms layer to obtain the sentences of the given entities.

4.8 Nominal Coreference Resolution

- **Module**: CorefGraph
- **Description of the module**: The module of coreference resolution included in the IXA pipeline that is used for English is also used for Dutch in an adapted form. For convenience, we repeat the description provided above before adding a few remarks on adaptations for Dutch. The coreference module is loosely based on the Stanford Multi Sieve Pass system [?]. The system consists of a number of rule-based sieves. Each sieve pass is applied in a deterministic manner, reusing the information generated by the previous sieve and the mention processing. The order in which the sieves are applied favours a highest precision approach and aims at improving the recall with the subsequent application of each of the sieve passes. This is illustrated by the evaluation results of the CoNLL 2011 Coreference Evaluation task [?; ?], in which the Stanford's system obtained the best results. The results show a pattern which has also been shown in other results reported with other evaluation sets [?], namely, the fact that a large part of the performance of the multi pass sieve system is based on a set of significant sieves. Thus, this module focuses for the time being, on a subset of sieves only, namely, Speaker Match, Exact Match, Precise Constructs, Strict Head Match and Pronoun Match [?]. The algorithm uses constituent trees and morphosyntactic information to determine coreference. We have adapted the original implementation so that it can deal with the richer morphosyntactic output of Alpino that our current pipeline for Dutch requires. Furthermore, constituent trees provided by Alpino are structured differently from those provided by the Stanford parser (which is used as the basis for implementing this algorithm). Minor adaptations were required to process Alpino's constituent trees.
- **Input**: lemma, morphosyntactic information, named-entities, constituents
- **Input representation**: NAF entities, constituency and term layers.
- **Output**: coreferences
- **Output representation**: NAF coreference layer
- **Required modules**: Tokenizer, Alpino POS tagger, lemmatization and parsing and NERC modules
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: none
- **Dependencies**: pyKAF, pycorpus, networkx, pyYALM
- **Flexible in- and output**: no (KAF and NAF in- and output only)

- **github address**: `https://bitbucket.org/Josu/corefgraph`

4.9 Semantic Role Labeling

- **Module**: vua-srl
- **Description of the module**: This module is based on the Semantic Role Labelling system as described in [?]. The original system takes one-file-per-sentence xml files as generated by the Alpino parser and generates comma separated feature vectors that provide the input to the machine learner (TiMBL [?]). This machine learning algorithm tries to predict the role label between the predicate and the dependency that are represented in the feature vector.

  The training data used was annotated in the context of the SoNaR project [?]. For this module, we settled on using only the texts that pertain to newswire or magazines.

  For optimal integration with the NewsReader pipeline and to make the module robust against changes in any of the preceding processing steps (e.g. updates of the parser), we chose to reimplement their feature generator so that it takes a NAF file containing the term, constituents and dependencies layers as input. We have also added identifiers to the start of the feature vectors to make insert the predicted roles with the NAF file easier.
- **Input**: Lemmatized and POS tagged text and syntactic dependencies
- **Input representation**: NAF terms, constituents and dependencies layers
- **Output**: Semantic roles
- **Output representation**: NAF srl layer
- **Required modules**: Tokenizer, POS tagger and Alpino parsing modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: SoNaR SRL training data
- **Dependencies**: Python, TiMBL 6.4.5
- **Flexible in- and output**: no (KAF and NAF in- and output only)
- **github address**: `https://github.com/newsreader/vua-srl-nl`

4.10 FrameNet labelling

- **Module**: vua-framenet-classifier
- **Description of the module**: This module assign FrameNet frames and elements to predicates and roles in the SRL layer. It read the predicates in the SRL layer, matches these with the terms and their wordnet references to find the possible frames and elements as they are given in the PredicateMatrix. Next, it selects the highest ranked frame from all the possible frames and assign it to the predicate. The frame elements linked to that frame are then used to find a proper match with the PropBank roles assigned by the Semantic Role Labeling. If there is a direct match, it is added to the role as an external reference, if not it assigns the frame element to the PropBank role that has the strongest association.
- **Input**: semantic roles and synsets
- **Input representation**: NAF terms, SRL layers
- **Output**: External references to FrameNet added to the SRL layer
- **Output representation**: NAF srl layer
- **Required modules**: WSD and SRL
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: predicate matrix and model for frame element = propbank role associations
- **Dependencies**: Java, Maven, NAF/KAF Saxparser
- **Flexible in- and output**: input stream NAF, output stream NAF
- **github address**: `https://github.com/cltl/OntoTagger`

4.11 Opinions

- **Module**: opinion-miner
- **Description of the module**: This is a module that detects and extracts fine-grained opinions, where one single opinion contains three elements: the opinion expression (the subjective statement itself), the opinion target (what the opinion is about) and the opinion holder (who is stating the opinion). This module has been developed on the OpeNER project,[36] where it has been trained on different domains (hotel

[36]`http://www.opener-project.eu/`

reviews, political news...), and for different languages (Dutch, English, Spanish, Italian, French and German) using corpora manually annotated also during the project. The extraction and tagging of opinions is divided into two steps. First, the detection of opinion entities (holder, target and expression) using Conditional Random Fields using as features the tokens lemmas and pos on the context, as well as syntactic features and features extracted from lexicons. The second step is the opinion entity linking (expression¡-target and expression-¡holder) using binary Support Vector Machines. In this step all the single opinion entities detected are grouped into triples ¡expression,target,holder¿ according to the output of the SVM classifiers. In this case, besides the local context features, dependency features and features capturing the relative location of the opinion elements are included. The models have been trained with a rich set of features, but the opinion tagger can be used with a reduced subset of this features considering that the performance will be affected.

- **Input**: NAF text processed trough the pipeline, this module will use the information provided by all the rest
- **Input representation**: token, term, entity, dependency and constituency NAF layers
- **Output**: fine-grained opinion triples <expression, target, holder>
- **Output representation**: NAF opinion layer
- **Required modules**: tokenizer, lemmatizer, POS tagger, polarity tagger, named entity tagger, constituency parser and dependency parser
- **Level of operation**: document
- **Language dependent**: yes
- **Resources**: models trained on the OpeNER project
- **Dependencies**: CRF library,[37] SVM-Light library,[38] KafNafParser[39] and VUA_pylib library[40]
- **Flexible in- and output**: it takes NAF or KAF input text, and generates NAF or KAF output text enriched with the extracted opinions
- **github address**: `https://github.com/cltl/opinion_miner_deluxe`

[37]`http://www.chokkan.org/software/crfsuite/`
[38]`http://www.chokkan.org/software/crfsuite/`
[39]`https://github.com/cltl/KafNafParserPy`
[40]`https://github.com/cltl/VUA_pylib`

4.12 Event coreference

- **Module**: vua-eventcoreference
- **Description of the module**: This module takes the predicates of the SRL layer as input and matches the predicates semantically. If the predicates are sufficiently similar, a coreference set is created with references to the predicates as coreferring expressions. If there is no match, predicates form a singleton set in the coreference layer.
- **Input**: SRL
- **Input representation**: SRL predicates
- **Output**: Coreference sets for events
- **Output representation**: NAF coref layer
- **Required modules**: vua-srl
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: wordnet-lmf
- **Dependencies**: Java, Maven, KAF/NAF Saxparser, WordnetTools
- **Flexible in- and output**: no
- **github address**: `https://github.com/cltl/EventCoreference`

5 Italian NLP Processing

The NLP processing suite for Italian is based on TextPro ([?]). The suite has been designed so as to integrate and reuse state of the art NLP components developed by researchers at HLT-FBK group. A wrapper program allows for specifying what kind of analysis are requested, and takes into account possible interdependencies between tasks. Internally each module uses a column format (similar to CoNLL tabular format) of the input/output as interchange format. The NAF format is used to get the input text and to provide the final output. The wrapper is written in Java. In the following we describe each module of the Italian pipeline.

5.1 Tokenizer

- **Module**: fbk-tokenpro
- **Description of the module**: TokenPro provides tokenization and sentence segmentation given an Italian raw text. It is a rule based splitter tool and it can be fully customizable from an XML configuration file (putting specific splitting word rules or handling UTF-8 symbols, such as the uncommon apostrophe, quote, dash,...). The sentence splitting is performed when a sentence-ending character (like ., !, or ?) is found and it doesn't belong to an abbreviation. TokenPro is not distributed separately but is included in the TextPro distribution
- **Input**: Raw text
- **Input representation**: NAF raw layer
- **Output**: Tokens and end of sentences.
- **Output representation**: column format
- **Required modules**: None
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: list of Italian abbreviations
- **Dependencies**: Java
- **Flexible in- and output**: It takes plain text or raw text in NAF. It produces tokenized and segmented text in NAF, running text and CoNLL formats.
- **github address**: `https://github.com/hltfbk/TextPro`

5.2 Morphological analysis

- **Module**: fbk-morphopro
- **Description of the module**: MorphoPro is a morphological analyzer (finds all possible morphological analyses of a word in a text). It is based on word-form lists: for Italian it uses 1,878,285 analyses for 149,372 lemmas.
- **Input**: token
- **Input representation**:
- **Output**: all possible morphological analyses for each token

- **Output representation**: list of analyses
- **Required modules**: tokenizer
- **Level of operation**: word level
- **Language dependent**: yes
- **Resources**: word-form list for Italian as finite state automata
- **Dependencies**: C++ library
- **Flexible in- and output**:
- **github address**: `https://github.com/hltfbk/TextPro/tree/master/modules/MorphoPro`

5.3 POS tagging

- **Module**: fbk-tagpro
- **Description of the module**: TagPro is a module for PoS-tagging based on Support Vector Machine. It comes with two language models, Italian and English. The Italian model is trained on a corpus using a subset of the ELRA tagset ([?]). It was the best system at EVALITA 2007 (98.04% F1).
- **Input**: token, morphological analysis
- **Input representation**: column format
- **Output**: POS
- **Output representation**: column format
- **Required modules**: tokenizer, morphological analizer
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: language model
- **Dependencies**: YamCha,[41] TinySVM[42]
- **Flexible in- and output**:
- **github address**: `https://github.com/hltfbk/TextPro/tree/master/modules/TagPro`

[41]`http://chasen.org/~taku/software/yamcha/`
[42]`http://chasen.org/~taku/software/TinySVM/`

5.4 Lemmatization

- **Module**: fbk-lemmapro
- **Description of the module**: This module gets lemma from the morphological analysis/es that is/are compatible with the selected POS tag.
- **Input**: token, morphological analysis, POS
- **Input representation**: column format
- **Output**: lemma
- **Output representation**: column format
- **Required modules**: tokenizer, morphological analizer, POS-tagger
- **Level of operation**: word level
- **Language dependent**: yes
- **Resources**: mapping among morphological features and POS tagset
- **Dependencies**: Java
- **Flexible in- and output**:
- **github address**: `https://github.com/hltfbk/TextPro/tree/master/modules/LemmaPro`

5.5 Named Entity Recognition and Classification

- **Module**: fbk-entitypro
- **Description of the module**: EntityPro performs name entity recognition based on machine learning. It exploits a rich set of linguistic features, and the occurrence in proper nouns gazetteers. The Italian model is trained on ICAB ([?]). The data contains entities of four types: Person (PER), Organization (ORG), Location (LOC) and Geo-Political entity (GPE). It was the best performing at EVALITA 2008 (82.1% F1). The module for system training is included in the distribution, and also the customization through white/black lists is possible.
- **Input**: token, lemma, POS
- **Input representation**: column format
- **Output**: Named entities
- **Output representation**: column format

- **Required modules**: tokenizer, morphological analizer, lemmatizer, POS-tagger
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: language model, Proper nouns gazetteers
- **Dependencies**: YamCha,[43] TinySVM,[44]
- **Flexible in- and output**:
- **github address**: `https://github.com/hltfbk/TextPro/tree/master/modules/EntityPro`

5.6 Chunking

- **Module**: fbk-chunkpro
- **Description of the module**: This module groups words into flat constituents for a syntactic analysis. Chunking for Italian annotates with 2 categories: B-NP (for nominal predicate), B-VX (for verbal predicate).
- **Input**: token, POS
- **Input representation**: column format
- **Output**: chunk
- **Output representation**: column format
- **Required modules**: tokenizer, POS-tagger
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: language model
- **Dependencies**: YamCha,[45] TinySVM[46]
- **Flexible in- and output**:
- **github address**: `https://github.com/hltfbk/TextPro/tree/master/modules/ChunkPro`

[43] `http://chasen.org/~taku/software/yamcha/`
[44] `http://chasen.org/~taku/software/TinySVM/`
[45] `http://chasen.org/~taku/software/yamcha/`
[46] `http://chasen.org/~taku/software/TinySVM/`

5.7 Time expression detection and normalization

- **Module**: fbk-timepro
- **Description of the module**: This module recognizes and normalizes temporal expressions in Italian. It processes the same way as TimePro for English. The core library timenorm ([?]) has been adaptated for Italian. The Italian model is trained on EVENTI-EVALITA 2014 training data. At EVALITA 2014 it was the best performing on time expression recognition (82.7% F1) and class detection (80% F1) This module has been integrated into TextPro pipeline but it also accepts a NAF input file with tokenization, POS-tagging and chunking information and provides a NAF file as output.
- **Input**: token layer, lemma, POS, entity, chunk
- **Input representation**: column format or NAF terms layer
- **Output**: timex3
- **Output representation**: NAF timex layer
- **Required modules**: tokenizer, POS-tagger, named entity recognizer, chunker
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: language model, language dependent rules, Italian grammar for timenorm
- **Dependencies**: timenorm ([?]), YamCha,[47] TinySVM[48]
- **Flexible in- and output**: use also TextPro format (i.e. column format)
- **github address**: `https://github.com/hltfbk/TextPro/tree/master/modules/TimePro`

5.8 Dependency Parser

- **Module**: fbk-syntaxpro
- **Description of the module**: This module implements an Italian Dependency Parser. It is based on Malt Parser ([?]), a language-independent system for data-driven dependency parsing written in Java (open source). It was evaluated at Evalita 2011, using the Turin University Treebank for training (88.62% LAS, 92.85% UAS).
- **Input**: token layer, POS, lemma

[47]`http://chasen.org/~taku/software/yamcha/`
[48]`http://chasen.org/~taku/software/TinySVM/`

- **Input representation**: column format
- **Output**: dependency
- **Output representation**: column format
- **Required modules**: tokenizer, POS-tagger, lemmatizer
- **Level of operation**: sentence level
- **Language dependent**: language model
- **Resources**:
- **Dependencies**: MaltParser `http://www.maltparser.org/`
- **Flexible in- and output**:
- **github address**: `https://github.com/hltfbk/TextPro/tree/master/modules/DepParserPro`

5.9 Event recognizer

- **Module**: fbk-eventpro
- **Description of the module**: EventPro detects event extents and classifies them in one of the 7 classes defined in TimeML. The module is based on machine learning and it uses the Support Vector Machine (SVM) implementation provided by yamcha. The Italian model is trained on EVENTI-EVALITA 2014 data. It was evaluated at EVALITA 2014, and obtained the result of 86.7% F1 for the task of event recognition and 67.1% F1 for event classification.
- **Input**: token layer, lemma, POS, entity, chunk, time expression, morpho
- **Input representation**: NAF terms layer
- **Output**: event
- **Output representation**: NAF event layer
- **Required modules**: tokenizer, POS-tagger, morphological analizer, chunker, named entity recognizer, temporal expressions recognizer and normalizer
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: language model, language dependent rules

- **Dependencies**: Snowball Italian stemmer (`http://snowball.tartarus.org/algorithms/italian/stemmer.html`), MultiWordNet domains (`http://multiwordnet.fbk.eu/english/home.php`), derIvaTario lexicon (`http://derivatario.sns.it/`), YamCha (`http://chasen.org/~taku/software/yamcha/`), TinySVM[49]
- **Flexible in- and output**: use also TextPro format (i.e. column format)
- **github address**:

5.10 Temporal relations extraction

- **Module**: fbk-temprel
- **Description of the module**: TempRel extracts temporal relations between events and time expressions as defined in TimeML. It processes the same way as TempRel for English. The Italian model is trained on EVENTI-EVALITA 2014 data. At EVALITA 2014 on the task of relation classification (identification of the relation type given the relations) it obtained 73.6% F1.
- **Input**: token layer, lemma, POS, entity, morpho, dependency, event, time
- **Input representation**: NAF terms layer
- **Output**: tlink
- **Output representation**: NAF temporal relation layer
- **Required modules**: tokenizer, POS-tagger, morphological analizer, chunker, named entity recognizer, temporal expressions recognizer and normalizer, dependency parser, event recognizer
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: language model, language dependent rules
- **Dependencies**: YamCha (`http://chasen.org/~taku/software/yamcha/`), TinySVM[50]
- **Flexible in- and output**: use also TextPro format (i.e. column format)
- **github address**: `https://github.com/hltfbk/TextPro/tree/master/modules/TempRelPro/`

[49] `http://chasen.org/~taku/software/TinySVM/`
[50] `http://chasen.org/~taku/software/TinySVM/`

5.11 Factuality detection

- **Module**: fbk-factpro
- **Description of the module**: FactPro performs in 3 steps: detection of the polarity of an event (machine learning based), identification of the certainty of an event (machine learning based) and identification of the semantic time (rules based). The Italian models are trained on "Training data EVENTI task - Part 2"[51] annotated with factuality on top of TimeML annotation.
- **Input**: token layer, lemma, POS, entity, chunk, morpho, event
- **Input representation**: NAF terms layer
- **Output**: factuality
- **Output representation**: NAF factuality layer
- **Required modules**: tokenizer, POS-tagger, morphological analizer, chunker, named entity recognizer, event recognizer
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: language model, language dependent lexicons
- **Dependencies**: derIvaTario lexicon (`http://derivatario.sns.it/`), YamCha (`http://chasen.org/~taku/software/yamcha/`)
- **Flexible in- and output**: use also TextPro format (i.e. column format)
- **github address**:

6 Spanish NLP Processing

The NLP processing for Spanish is similar to the English pipeline as they both share various modules to perform the processing. In this section, we follow the same structure as in Section 2 to present the modules. The descriptions of the modules are provided along with some technical information: input, output, required modules, level of operation, language dependency, resources, dependencies and github address.

[51] `https://sites.google.com/site/eventievalita2014/data-tools`

6.1 Tokenizer

- **Module**: ixa-pipe-tok
- **Description of the module**: This module provides Sentence Segmentation and Tokenization for English and Spanish and other languages such as Dutch, German, English, French, Italian and Spanish. It implements a rule-based segmenter and tokenizer originally inspired by the Stanford English Penn Treebank tokenizer[52] but with several modifications and improvements. These include tokenization for other languages such as Italian, normalization according the Spanish Ancora Corpus [?], paragraph treatment, and more comprehensive gazeteers of non breaking prefixes. The tokenizer depends on a JFlex[53] specification file which compiles in seconds and performs at a very reasonable speed (around 250K word/second, and much quicker with Java multithreading). JFlex is a lexical analyser generator. The module is part of the IXA pipes [?],[54] a modular set of Natural Language Processing tools (or pipes) which provide easy access to NLP technology for English and Spanish.
- **Input**: Raw text
- **Input representation**: NAF raw layer
- **Output**: Tokens and sentences.
- **Output representation**: NAF text layer
- **Required modules**: None
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: None
- **Dependencies**: Java, Maven, NAF Java library, JFlex
- **Flexible in- and output**: It takes plain text or raw text in NAF. It produces tokenized and segmented text in NAF, running text and CoNLL formats.
- **github address**: `https://github.com/newsreader/ixa-pipes`

[52]`http://www-nlp.stanford.edu/software/tokenizer.shtml`
[53]`http://jflex.de/`
[54]`http://ixa2.si.ehu.es/ixa-pipes/`

6.2 POS tagging

- **Module**: ixa-pipe-pos
- **Description of the module**: This module provides POS tagging and lemmatization for English and Spanish. The module is part of the IXA pipes. We have obtained the best results so far with *Maximum Entropy* models and the same featureset as in [?]. The models ave been trained and evaluated for Spanish using the Ancora corpus; it was randomly divided in 90% for training and 10% for testing. This corresponds to 440K words used for training and 70K words for testing. We obtain a performance of 98.88% (the corpus partitions are available for reproducibility). [?] report 98.86%, although they train and test on a different subset of the Ancora corpus.

  For Spanish, lemmatization is currently performed via 2 different dictionary lookup methods: a) *Simple Lemmatizer*: It is based on HashMap lookups on a plain text dictionary. Currently we use dictionaries from the LanguageTool project[55] under their distribution licenses; b) Morfologik-stemming:[56] The Morfologik library provides routines to produce binary dictionaries, from dictionaries such as the one used by the Simple Lemmatizer above, as finite state automata. This method is convenient whenever lookups on very large dictionaries are required because it reduces the memory footprint to 10% of the memory required for the equivalent plain text dictionary. By default, the module accepts tokenized text in NAF format as standard input and outputs NAF.
- **Input**: Tokens
- **Input representation**: NAF text layer
- **Output**: Lemmas and POS-tags
- **Output representation**: NAF terms layer
- **Required modules**: Tokenizer module
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: POS model; Lemmatizer dictionaries: plain text dictionary and morfologik-stemming dictionary.
- **Dependencies**: Java, Maven, NAF Java library, JWNL API, Apache OpenNLP.
- **Flexible in- and output**: It accepts tokenized text in NAF. It outputs NAF or CoNLL formats.
- **github address**: `https://github.com/newsreader/ixa-pipes`

[55] `http://languagetool.org/`
[56] `https://github.com/morfologik/morfologik-stemming`

6.3 Constituency Parser

- **Module**: ixa-pipe-parse
- **Description of the module**: This module provides statistical constituent parsing for English and Spanish. The module is part of the IXA pipes. Maximum Entropy models are trained to build shift reduce bottom up parsers [?] as provided by the Apache OpenNLP Machine Learning API. Parsing models for Spanish have been trained using the Ancora corpus [?]. Furthermore, ixa-pipe-parse provides two methods of headword finders: one based on Collins' head rules as defined in his PhD thesis [?], and another one based on Stanford's parser Semantic Head Rules.[57] The latter are a modification of Collins' head rules according to lexical and semantic criteria. We obtain a F1 88.40%. As far as we know, and although previous approaches ex- ist [?], ixa-pipe-parse provides the first publicly available statistical parser for Spanish. The module accepts lemmatized and POS tagged text in NAF format as standard input and outputs NAF.
- **Input**: Lemmatized and POS tagged text
- **Input representation**: NAF terms layer
- **Output**: Constituents; Syntactic tree of sentences.
- **Output representation**: NAF constituency layer
- **Required modules**: Tokenizer and POS tagger modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: Parsing model
- **Dependencies**: Java, Maven, NAF Java library, Apache OpenNLP
- **Flexible in- and output**: It accepts lemmatized and POS tagged text in NAF format. In addition to NAF output, ixa-pipe-parse also allows to output the parse trees into Penn Treebank bracketing style.
- **github address**: `https://github.com/newsreader/ixa-pipes`

[57] `http://www-nlp.stanford.edu/software/lex-parser.shtml`

6.4 Dependency Parser

- **Module**: ixa-pipe-srl
- **Description of the module**: This module is based on the MATE-tools [?], a pipeline of linguistic processors that performs lemmatization, part-of-speech tagging, dependency parsing, and semantic role labeling of a sentence. As the input of the module is a NAF file that includes lemmatization and pos-tagging, the module only implements the dependency parser [?]. The module is ready to work with Spanish and English.
- **Input**: Lemmatized and POS tagged text
- **Input representation**: NAF terms layer
- **Output**: Dependencies
- **Output representation**: NAF deps layer
- **Required modules**: Tokenizer and POS tagger modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: mate-tools package, dependency parsing model[58]
- **Dependencies**: Java, Maven, NAF Java library, mate-tools
- **Flexible in- and output**: It accepts lemmatized and POS tagged text in NAF format. The modules allows to output dependencies trees in NAF and CoNLL formats.
- **github address**: `https://github.com/newsreader/ixa-pipe-srl`

6.5 Time Expressions

- **Module**: ixa-heideltime
- **Description of the module**: This module identifies time expressions in text and normalizes them. The core component of the module is HeidelTime [?], a temporal tagger supporting English, German, Dutch, Vietnamese, Arabic, Spanish, Italian, French, Chinese and Russian.[59] HeidelTime identifies temporal expressions based on language specific patterns. Identified temporal expressions are normalized and represented according to TIMEX annotations [?]. This module is currently under development.

[58]The module use additional resources to perform the semantic role labeling.

[59]HeidelTime source code is available here: `https://code.google.com/p/heideltime/`

- **Input**:
- **Input representation**:
- **Output**: Normalized time expressions in timex
- **Output representation**: NAF time expression layer
- **Required modules**: Tokenizer module
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: language specific rules
- **Dependencies**: Python
- **Flexible in- and output**:
- **github address**: `https://github.com/ixa-ehu`

6.6 Named Entity Recognition and Classification

- **Module**: ixa-pipe-nerc
- **Description of the module**: This module is a multilingual Named Entity Recognition and Classification tagger. ixa-pipe-nerc is part of IXA pipes. The named entity types are based on: a) the CONLL 2002[60] and 2003[61] tasks which were focused on language-independent supervised named entity recognition for four types of named entities: persons, locations, organizations and names of miscellaneous entities that do not belong to the previous three groups. We currently provide two very fast language independent models using a rather simple baseline feature set. It is based on the features presented by [**?**] with several differences: We do not use POS tags, chunking or gazetteers in our baseline models but we do use bigrams as a feature.

  For Spanish we currently obtain best results training Maximum Entropy models on the CoNLL 2002 dataset. Our best model obtains 80.16 F1 vs 81.39 F1 of [**?**], the best result so far on this dataset. Their result uses external knowledge and without it, they system obtains 79.28 F1. The module allows to format its output in CoNLL style tabulated BIO format as specified in the CoNLL 2003 shared evaluation task.
- **Input**: Lemmatized and POS tagged text
- **Input representation**: NAF terms layer

[60]`http://www.clips.ua.ac.be/conll2002/ner/`
[61]`http://www.clips.ua.ac.be/conll2003/ner/`

- **Output**: Named entities
- **Output representation**: NAF entities layer
- **Required modules**: Tokenizer and POS tagger modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: CoNLL 2003 models, properties file
- **Dependencies**: Java, Maven, NAF Java library, Apache OpenNLP
- **Flexible in- and output**: It accepts lemmatized and POS tagged text in NAF format. The modules allows to output dependencies trees in NAF and CoNLL formats.
- **github address**: `https://github.com/newsreader/ixa-pipes`

6.7 Word Sense Disambiguation

- **Module**: wsd-ukb
- **Description of the module**: UKB is a collection of programs for performing graph-based Word Sense Disambiguation. UKB applies the so-called Personalized PageRank on a Lexical Knowledge Base (LKB) to rank the vertices of the LKB and thus perform disambiguation. UKB has been developed by the IXA group. The module accepts lemmatized and POS tagged text in NAF format as standard input and outputs NAF.
- **Input**: Lemmatized and POS tagged text
- **Input representation**: NAF terms layer
- **Output**: Synsets
- **Output representation**: NAF terms layer
- **Required modules**: Tokenizer and POS tagger modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: Spanish WordNet
- **Dependencies**: C++, boost libraries
- **Flexible in- and output**: no
- **github address**: `https://github.com/ixa-ehu/ukb`

6.8 Named Entity Disambiguation

- **Module**: ixa-pipe-ned
- **Description of the module**: This module performs the Named Entity Disambiguation task based on DBpedia Spotlight. Assuming that a DBpedia Spotlight Rest server for a given language is locally running, the module will take NAF as input (containing elements) and perform Named Entity Disambiguation. . The module offers the "disambiguate" and "candidates" service endpoints. The former takes the spotted text input and it returns the identifier for each entity. The later is similar to disambiguate, but returns a ranked list of candidates. For Spanish, given a disambiguate entity, it is also possible to return the corresponding English dbpedia-entry. The module accepts text with named entities in NAF format as standard input, it disambiguates them and outputs them in NAF.
- **Input**: Named entities and sentences
- **Input representation**: NAF entities layer
- **Output**: Disambiguated named entities
- **Output representation**: NAF entities layer
- **Required modules**: NERC module
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: DBpedia spotlight server
- **Dependencies**: Java, Maven, MapDB, NAF Java library, DBpedia spotlight
- **Flexible in- and output**: no
- **github address**: `https://github.com/newsreader/ned-spotlight`

6.9 Coreference Resolution

- **Module**: corefgraph
- **Description of the module**: The module of coreference resolution included in the IXA pipeline is loosely based on the Stanford Multi Sieve Pass system [?]. The system consists of a number of rule-based sieves. Each sieve pass is applied in a deterministic manner, reusing the information generated by the previous sieve and the mention processing. The order in which the sieves are applied favours a highest precision approach and aims at improving the recall with the subsequent application

of each of the sieve passes. This is illustrated by the evaluation results of the CoNLL 2011 Coreference Evaluation task [?; ?], in which the Stanford's system obtained the best results. The results show a pattern which has also been shown in other results reported with other evaluation sets [?], namely, the fact that a large part of the performance of the multi pass sieve system is based on a set of significant sieves. Thus, this module focuses for the time being, on a subset of sieves only, namely, Speaker Match, Exact Match, Precise Constructs, Strict Head Match and Pronoun Match [?].

- **Input**: lemma, POS, named-entities, constituents
- **Input representation**: NAF entities, constituency layers
- **Output**: coreferences
- **Output representation**: NAF coreferences layer
- **Required modules**: Tokenizer, POS-tagger and NERC modules
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: none
- **Dependencies**: pyKAF, pycorpus, networkx, pyYALM
- **Flexible in- and output**: It accepts lemmatized and POS tagged text, entities and constituents in NAF format. The modules allows to output coreference clusters in NAF and CoNLL formats.
- **github address**: `https://bitbucket.org/Josu/corefgraph`

6.10 Semantic Role Labeling

- **Module**: ixa-pipe-srl
- **Description of the module**: This module is based on the MATE-tools [?], a pipeline of linguistic processors that performs lemmatization, part-of-speech tagging, dependency parsing, and semantic role labeling of a sentence. They report on the CoNLL 2009 Shared Task a labelled semantic F1 of 85.63 for English and 79.91 for Spanish. As the input of the module is a NAF file that includes lemmatization, pos-tagging and dependency parsing, the module only implements the semantic role labeler [?]. The module is ready to work with Spanish and English. By default, the module accepts parsed text in NAF format as standard input and outputs the enriched text in NAF.

- **Input**: Lemmatized and POS tagged text and syntactic dependencies
- **Input representation**: NAF terms, deps layers
- **Output**: Semantic roles
- **Output representation**: NAF srl layer
- **Required modules**: Tokenizer, POS tagger and Dependency parsing modules
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: mate-tools package, PredicateMatrix
- **Dependencies**: Java, Maven, NAF Java library, mate-tools
- **Flexible in- and output**: It accepts lemmatized and POS tagged text and syntactic dependencies in NAF format. The modules allows to output semantic roles in NAF and CoNLL formats.
- **github address**: `https://github.com/newsreader/ixa-pipe-srl`

6.11 Event coreference

- **Module**: vua-eventcoreference
- **Description of the module**: This module takes the predicates of the SRL layer as input and matches the predicates semantically. If the predicates are sufficiently similar, a coreference set is created with references to the predicates as coreferring expressions. If there is no match, predicates form a singleton set in the coreference layer.
- **Input**: SRL
- **Input representation**: NAF srl layer
- **Output**: Coreference sets for events
- **Output representation**: NAF coref layer
- **Required modules**: ixa-pipe-srl
- **Level of operation**: sentence level
- **Language dependent**: yes
- **Resources**: wordnet-lmf

- **Dependencies**: Java, Maven, KAF/NAF Saxparser, WordnetTools
- **Flexible in- and output**: no
- **github address**: `https://github.com/cltl/EventCoreference`

7 Text Classification

In this second cycle of the project, we have decided to automatically set the topic of each document. This task is part of WP4 and it is applicable in all the languages of the project. We have implemented one module which works for the four languages of the project and it runs per set of documents. Given a set of documents, the module can be seen either as a pre-processing or post-processing step.

Document descriptors are useful in NewsReader to perform event coreference. The topic determines the domain of the document and this information, among other features, is used for event coreference resolution [?].

The module is based on the Multilingual Eurovoc thesaurus descriptors and it makes use of the JRC Eurovoc Indexer JEX [?]. As the rest of the available modules for text processing, this module reads from the standard input a NAF file and it writes the new version with the topic information in NAF. The topic information is represented in the <topic> layer in the NAF document and it corresponds to the whole document.

The technical information regarding this module can be summarized as follows:

- **Input**: text
- **Input representation**:
- **Output**: topic
- **Output representation**: NAF topic layer
- **Required modules**: None
- **Level of operation**: document level
- **Language dependent**: yes
- **Resources**: None
- **Dependencies**: Java, Maven, NAF Java library, JEX
- **Flexible in- and output**: no
- **github address**: `https://github.com/newsreader/`

We are currently studing an alternative version of the module that processes one document per time. If the processing time is reasonable, we will integrate it into the pipelines.

8 Scaling of Text Processing

In the first cycle of event detection, we performed some initial experiments with the goal of analyzing the scaling capabilities of our English NLP processing pipeline. As a result of the analysis made in WP2, we chose the distributed framework called STORM to integrate the pipeline into it. For that we use virtual machines which are described in [?]. The results of these experiments are described in the deliverable 4.2.1 "Event detection, version 1".

In the second cycle of the project, we have continued collecting and processing different data. So far, we have stored the following data: LN car company news (EN) : 63K articles (6M); TechCrunch (EN) : 43K articles; WikiNews (EN, ES) : 19K English and 7K Spanish; ECB+ (EN) : 984 articles; FIFA World Cup (Hackathon Londres): LexisNexis, BBC, The Guardian (200K news).

The analyzed documents from the FIFA 2014 World Cup domain were the basis from which to create the RDF triplets as used in the First NewsReader Hackaton held in London on the 10th June, 2014. The documents were processed within a 15 day time-frame using 30 copies of the virtual machines. FBK has set up and run the NLP processing pipeline on the FBK cluster. Once again, we tested the pipeline as regards the scaling of text processing. Due to time constraints, the 200K news articles were processed using the processing pipeline set up at the end of Y1.

9 Conclusions

This deliverable describes the second version of the **Event Detection** framework developed in NewsReader to process large and continuous streams of English, Dutch, Spanish and Italian news articles.

During the second cycle of the NewsReader project (Event Detection, version 2) we focused on providing generic pipelines for English, Dutch, Spanish and Italian. We also improved the modules for English event detection. For instance, several modules of the English NLP pipeline have been improved, including those for event detection, named entity recognition, coreference resolution and semantic role labeling. Some of these modules have been also adpated to the financial domain. We have been also deploying new generic pipelines and modules for Dutch, Spanish and Italian.

In order to achieve cross-lingual semantic interoperability, entity and event mentions have been projected to language independent knowledge representations. Thus, named entities are linked as much as possible to external sources such as DBpedia entity identifiers while event mentions are aligned to abstract event representations thanks to the Predicate Matrix [?]. We are also developing new techniques and resources to achieve interoperable semantic interpretation for English, Dutch, Spanish and Italian thanks to DBpedia cross-lingual links and multilingual semantic projections through local wordnets of the Predicate Matrix.

The benchmarks for the four languages will be available in the last trimester of the second year of the project. We will use this datasets to evaluate our four pipelines. The evaluation results will be presented in deliverable D3.3.2 “Annotated data.”

Five datasets covering different topics have been processed by the NLP pipeline. Together these sets consists of approximately 326K articles in English and 7K articles in Spanish.

We will continue researching on the improvement of the event detection systems by analysing the interacting tasks such as NERC, NED and coreference. We are also planning to define an experiment to start exploiting the KnowledgeStore provided in WP06 to try to improve the event detection system.

A Output representation

This section presents the output of each module presented in sections 2, 3, 4 and 5 in the NAF representation format. Each module produces linguistic information at one layer defined in NAF. The complete NAF representation is available at `http://wordpress.let.vupr.nl/naf/` and `https://github.com/newsreader/NAF`.

- ixa-pipe-tok

  The ixa-pipe-tok provides sentence segmentation and tokenization. It annotates word forms within the <text> element. Each form is enclosed by a <wf> element and it has the following attributes: word id, sentence id, paragraph id, the offset and length of the word form. Example:

```
    <wf id="w1" length="9" offset="21" para="1" sent="1">President</wf>
```

- ixa-pipe-pos

  The ixa-pipe-pos provides POS tagging and lemmatization. It annotates terms within the <terms> element. Each term is enclosed by a <term> element and it has the following attributes: term id, type, lemma, pos, morphofeat and the span sub-element which is used to identify the tokens that the term spans. Example:

```
    <term id="t1" lemma="President" morphofeat="NNP" pos="R" type="close">
      <span>
        <!--President-->
        <target id="w1"/>
      </span>
    </term>
```

- ixa-pipe-parse

  The ixa-pipe-parse provides constituent parsing. It annotates constituents within the <constituency> element, and each sentence (parse tree) is represented by a <tree> element. Inside each <tree>, there are three types of elements: a) <nt> elements representing non-terminal nodes; b) <t> elements representing terminal nodes; and c) <edge> elements representing in-tree edges. The <nt> element has the id and label attributes. The <t> element has the id attribute and the <span> element pointing to the term layer. Finally, the <edge> element has the id, from, to and head attributes. Example:

```
<constituency>
    <tree>
      <t id="ter1">
        <!--Mariano-->
        <span>
          <target id="t1"/>
        </span>
      </t>
      <t id="ter2">
        <!--Rajoy-->
        <span>
          <target id="t2"/>
        </span>
      </t>
      <t id="ter3">
        <!--is-->
        <span>
          <target id="t3"/>
        </span>
      </t>
      <t id="ter4">
        <!--the-->
        <span>
          <target id="t4"/>
        </span>
      </t>
      <t id="ter5">
        <!--President-->
        <span>
          <target id="t5"/>
        </span>
      </t>
      <t id="ter6">
        <!--of-->
        <span>
          <target id="t6"/>
        </span>
      </t>
      <t id="ter7">
        <!--Spain-->
        <span>
          <target id="t7"/>
        </span>
      </t>
      <t id="ter8">
        <!--.-->
        <span>
          <target id="t8"/>
        </span>
      </t>
      <nt id="nter1" label="TOP"/>
      <nt id="nter2" label="S"/>
      <nt id="nter3" label="NP"/>
      <nt id="nter4" label="NNP"/>
      <nt id="nter5" label="NNP"/>
      <nt id="nter6" label="VP"/>
      <nt id="nter7" label="VBZ"/>
      <nt id="nter8" label="NP"/>
      <nt id="nter9" label="NP"/>
      <nt id="nter10" label="DT"/>
      <nt id="nter11" label="NNP"/>
      <nt id="nter12" label="PP"/>
      <nt id="nter13" label="IN"/>
      <nt id="nter14" label="NP"/>
      <nt id="nter15" label="NNP"/>
      <nt id="nter16" label="."/>
      <edge from="nter2" head="yes" id="tre2" to="nter1"/>
      <edge from="nter3" id="tre3" to="nter2"/>
      <edge from="nter4" id="tre4" to="nter3"/>
      <edge from="ter1" id="tre5" to="nter4"/>
      <edge from="nter5" head="yes" id="tre6" to="nter3"/>
```

```
      <edge from="ter2" id="tre7" to="nter5"/>
      <edge from="nter6" head="yes" id="tre8" to="nter2"/>
      <edge from="nter7" id="tre9" to="nter6"/>
      <edge from="ter3" id="tre10" to="nter7"/>
      <edge from="nter8" head="yes" id="tre11" to="nter6"/>
      <edge from="nter9" head="yes" id="tre12" to="nter8"/>
      <edge from="nter10" id="tre13" to="nter9"/>
      <edge from="ter4" id="tre14" to="nter10"/>
      <edge from="nter11" head="yes" id="tre15" to="nter9"/>
      <edge from="ter5" id="tre16" to="nter11"/>
      <edge from="nter12" id="tre17" to="nter8"/>
      <edge from="nter13" head="yes" id="tre18" to="nter12"/>
      <edge from="ter6" id="tre19" to="nter13"/>
      <edge from="nter14" id="tre20" to="nter12"/>
      <edge from="nter15" head="yes" id="tre21" to="nter14"/>
      <edge from="ter7" id="tre22" to="nter15"/>
      <edge from="nter16" id="tre23" to="nter2"/>
      <edge from="ter8" id="tre24" to="nter16"/>
    </tree>
  </constituency>
```

- ixa-pipe-nerc

  The ixa-pipe-nerc module provides named entities. The <entity> element is used to represent a named entity in the document. The <entity> element has the id and type attributes. It also has the <references> sub-element which contains one or more <span> element, each one spaning terms. Example:

```
  <entities>
    <entity id="e1" type="PERSON">
      <references>
        <span>
          <!--Mariano Rajoy-->
          <target id="t1"/>
          <target id="t2"/>
        </span>
      </references>
      <externalReferences>
        <externalRef reference="http://dbpedia.org/resource/Mariano_Rajoy" ↩
            resource="spotlight_v1"/>
      </externalReferences>
    </entity>
```

- ixa-pipe-ned

  The ixa-pipe-ned module provides external references to named entities. Named entities are linked to an external resources using the <externalRef> element. The <externalRef> has the resource, reference and confidence attributes. Example:

```
  <entities>
    <entity id="e1" type="PERSON">
     ...
      <externalReferences>
        <externalRef reference="http://dbpedia.org/resource/Mariano_Rajoy" ↩
            resource="spotlight_v1" confidence="0.7"/>
      </externalReferences>
    </entity>
```

For Spanish, if the corresponding English dbpedia-entry is provided, the <externalRef> does not have the confidence attribute.

```
    <entity id="e1" type="PERSON">
      <references>
        <!--Mariano Rajoy-->
        <span>
          <target id="t1" />
          <target id="t2" />
        </span>
      </references>
      <externalReferences>
        <externalRef resource="spotlight_v1" reference="http://es.dbpedia.org/↩
            resource/Mariano_Rajoy" confidence="0.9999986" />
        <externalRef resource="wikipedia-db-esEn" reference="http://dbpedia.org/↩
            resource/Mariano_Rajoy" />
      </externalReferences>
    </entity>
```

- ixa-pipe-srl

  The ixa-pipe-srl provides dependency parsing. It annotates dependency relations among terms. Each dependency is represented by an empty <dep> element and span previous terms. The <dep> element has the from, to and rfunc attributes. Example:

```
  <deps>
    <!--NAME(Rajoy, Mariano)-->
    <dep from="t2" rfunc="NAME" to="t1"/>
    <!--SBJ(is, Rajoy)-->
    <dep from="t3" rfunc="SBJ" to="t2"/>
    <!--NAME(Spain, the)-->
    <dep from="t7" rfunc="NAME" to="t4"/>
    <!--NAME(Spain, President)-->
    <dep from="t7" rfunc="NAME" to="t5"/>
    <!--NAME(Spain, of)-->
    <dep from="t7" rfunc="NAME" to="t6"/>
    <!--PRD(is, Spain)-->
    <dep from="t3" rfunc="PRD" to="t7"/>
    <!--P(is, .)-->
    <dep from="t3" rfunc="P" to="t8"/>
  </deps>
```

The ixa-pipe-srl also provides semantic roles. Each annotate predicate is represented by an <predicate> element. The <predicate> element has the id attribute. It also has the <externalReferences>, <span> and <role> elements. The <span> element contains one or more <target> elements with the id attribute. The <role> element represents filler of a particular argument of the predicate and it has the id attribute and the <externalReferences> and <span> sub-elements.

```
    <predicate id="pr1">
      <!--elected-->
      <externalReferences>
        <externalRef reference="elect.01" resource="PropBank"/>
        <externalRef reference="appoint-29.1" resource="VerbNet"/>
        <externalRef reference="Change_of_leadership" resource="FrameNet"/>
        <externalRef reference="elect.01" resource="PropBank"/>
        <externalRef reference="contextual" resource="EventType"/>
      </externalReferences>
      <span>
        <target id="t4"/>
      </span>
      <role id="rl1" semRole="A1">
        <!--Mariano Rajoy-->
        <externalReferences>
          <externalRef reference="appoint-29.1#Theme" resource="VerbNet"/>
          <externalRef reference="Change_of_leadership-New_leader" resource="↩
              FrameNet"/>
          <externalRef reference="elect.01#1" resource="PropBank"/>
        </externalReferences>
        <span>
          <target id="t1"/>
          <target head="yes" id="t2"/>
        </span>
      </role>
      <role id="rl2" semRole="AM-TMP">
        <!--on 20 November 2011-->
        <span>
          <target head="yes" id="t5"/>
          <target id="t6"/>
          <target id="t7"/>
          <target id="t8"/>
        </span>
      </role>
    </predicate>
```

- wsd-vua

  The wsd-vua module provides word sense disambiguation. It associate terms to WordNet resource. It consists of several <externalRef> elements, one per association. The module returns the <externalRef> element with resource, reference and confidence attributes. Example:

```
    <term id="t9" lemma="man" morphofeat="NN" pos="N" type="open">
      <span>
        <!--man-->
        <target id="w9"/>
      </span>
      <externalReferences>
        <externalRef confidence="0.020081263" reference="r_n-23112" resource="↩
            Cornetto"/>
        <externalRef confidence="0.048200976" reference="r_n-23113" resource="↩
            Cornetto"/>
        <externalRef confidence="0.16325809" reference="r_n-23111" resource="↩
            Cornetto"/>
      </externalReferences>
    </term>
```

- wsd-ukb

The wsd-ukb module provides word sense disambiguation. It associate terms to WordNet resource. It consists of several <externalRef> elements, one per association. The module returns the <externalRef> element with resource, reference and confidence attributes. Example:

```
  <term id="t5" type="open" lemma="presidente" pos="N" morphofeat="NCMS000">
    <span>
      <target id="w5" />
    </span>
    <externalReferences>
      <externalRef resource="wn30sp.bin64" reference="eng-30-10467179-n" ↩
          confidence="0.266266" />
      <externalRef resource="wn30sp.bin64" reference="eng-30-10468559-n" ↩
          confidence="0.253603" />
      <externalRef resource="wn30sp.bin64" reference="eng-30-10467395-n" ↩
          confidence="0.245211" />
      <externalRef resource="wn30sp.bin64" reference="eng-30-10468962-n" ↩
          confidence="0.23492" />
    </externalReferences>
  </term>
```

- corefgraph

  The corefgraph module provides clusters of terms which share the same referent. The <coref> element has the id attribute and it also contains <span> elements. Each <span> contains one or more <target> element, each one with the id attribute. Example:

```
 <coref id="co1">
    <span>
      <!--the Prime Minister of Spain-->
      <target id="t4"/>
      <target id="t5"/>
      <target id="t6"/>
      <target id="t7"/>
      <target id="t8"/>
    </span>
    <span>
      <!--leader of the People s Party-->
      <target id="t19"/>
      <target id="t20"/>
      <target id="t21"/>
      <target id="t22"/>
      <target id="t23"/>
      <target id="t24"/>
    </span>
    <span>
      <!--Mariano Rajoy-->
      <target id="t1"/>
      <target id="t2"/>
    </span>
    <span>
      <!--He-->
      <target id="t16"/>
    </span>
  </coref>
```

- TimePro

  The TimePro module provides temporal expressions and their normalization. The <timex3> element is used to represent a temporal expression in the document. The <timex3> element has 5 attributes: id, type and value. This element is either empty (for example to represent the date of creation of the document) or contains a <span> element spaning words. Example:

```
<timeExpressions>
  <timex3 id="tmx0" type="DATE" value="2013-03-22" functionInDocument="↩
      CREATION_TIME" />
  <timex3 id="tmx1" type="DURATION" value="P1M">
<!-- a month -->
    <span>
      <target id="w26" />
      <target id="w27" />
    </span>
  </timex3>
</timeExpressions>
```

- TempRel

  The TempRel module provides temporal relations between events and time expressions. The <tlink> element is used to represent a temporal relation in the document. The <tlink> is an empty element which has 6 attributes: id, from, to, fromType, toType and relType. Example:

```
<temporalRelations>
<tlink from="coevent10" to="coevent11" relType="BEFORE" fromType="event" toType="↩
    event" id="tlink0"/>
    <tlink from="coevent1" to="tmx0" relType="AFTER" fromType="event" toType="↩
        timex" id="tlink1"/>
</temporalRelations>
```

- EventPro

  The EventPro module provides events. The <event> element is used to represent a event in the document. The <event> element has the id and class attributes. It also has the <references> sub-element which contains one or more <span> element, each one spanning terms. Example:

```
  <events>
    <event id="ev1" class="REPORTING">
      <references>
        <span>
          <!-- announcement -->
          <target id="t10"/>
        </span>
      </references>
    </event>
```

- CausalRel

  The CausalRel module provides causal relations between events. The <clink> element is used to represent a causal relation in the document. The <clink> is an empty element which has 4 attributes: id, from, to and relType. Example:

```
<causalRelations>
  <clink from="coevent10" to="coevent11" relType="CAUSE" id="clink0"/>
</causalRelations>
```