Contemplata, a Free Platform for Constituency Treebank Annotation

Jakub Waszczuk, Ilaine Wang, Jean-Yves Antoine, Anaïs Halftermeyer

Contemplata, a Free Platform for Constituency Treebank Annotation

Jakub Waszczuk[1], Ilaine Wang[2,3], Jean-Yves Antoine[2], Anaïs Halftermeyer[3]
[1]Heinrich-Heine-Universität, Düsseldorf, Germany
[2]LIFAT, Université de Tours, France
[3]LIFO, Université d'Orléans, France
waszczuk@hhu.de, jean-yves.antoine@univ-tours.fr,
{ilaine.wang, anais.halftermeyer}@univ-orleans.fr

Abstract

This paper describes Contemplata, an annotation platform that offers a generic solution for treebank building as well as treebank enrichment with relations between syntactic nodes. Contemplata is dedicated to the annotation of constituency trees. The framework includes support for syntactic parsers, which provide automatic annotations to be manually revised. The balanced strategy of annotation between automatic parsing and manual revision allows to reduce the annotator workload, which favours data reliability. The paper presents the software architecture of Contemplata, describes its practical use and eventually gives two examples of annotation projects that were conducted on the platform.

Keywords: treebank, syntactic annotation, spontaneous speech, French language, constituent trees

1. Introduction

This paper presents Contemplata, an annotation platform that was originally created to fulfil the needs of temporal annotation while providing a generic tool for the annotation of constituency treebanks.

The representation and the processing of temporality is important for many understanding NLP tasks. Temporal annotation has benefitted from the normalisation efforts of the ISO TC37/SC4 committee which has led to the definition of the ISO-TimeML standard (ISO, 2012). While originally developed for English, ISO-TimeML has been applied on a large variety of languages (Italian, Korean, Romanian, Chinese, French...) with only slight idiomatic adaptations. The Temporal@ODIL project aims at building a new French corpus annotated with both temporal mentions and temporal relations. The primary motivation behind the project is to focus specifically on spontaneous speech, unlike the French Time Bank (Bittar et al., 2011) which targets written text. This leads Temporal@ODIL to review the ISO-TimeML standard and propose enhancements, while preserving coherence and upward compatibility with the international standard (Lefeuvre-Halftermeyer et al., 2016). In that respect, the project meets a larger variety of needs in NLP and in corpus linguistics, in addition to offering a coverage of spoken language for temporality annotation. What makes the originality of this proposal is that temporal mentions are delimited not by their minimal chunk (lexical head) but by the range of the syntactic subtree that covers the temporal mention. This large-span annotation challenges the annotator's ability to delimit the temporal mentions with a satisfactory reliability. To ease this delimitation, we adopt a solution that was investigated for multi-word expressions in the Prague Dependency Treebank (Bejček and Straňák, 2010): temporal mentions are directly defined on the syntactic structures of the utterances. Thus, the delimitation task boils down to the selection of one specific node. Data reliability is therefore favoured by the reduction of the annotator's cognitive load. In addition, prior knowledge of the syntactic trees also eases the annotation of the semantic relations. For instance, the annotation of most of the subordinate relations (SLINK standing for Subordination Link in the ISO-TimeML standard) can be automatically inferred from the syntactic trees.

Temporal@ODIL adopts an incremental process of annotation: first, the corpus of speech transcripts is parsed into syntactic trees, resulting in the building of a treebank. Only then, a round of semantic annotation consisting in the delimitation of temporal mentions and the definition of relations is carried out. As an annotation platform, Contemplata offers to handle both the syntactic and the semantic annotations, thus offering a generic solution for treebank building and treebank enrichment with the addition of relations between syntactic nodes. The genericity of Contemplata is however restricted by its specialization to constituent trees. This restriction is due to our annotation needs. From a theoretical point of view, the resolution of temporal abstract anaphora often needs the consideration of a whole clause or a whole speech turn (Zinsmeister and Dipper, 2010), which cannot be modelled in a lexical head-based representation like ISO TimeML. From a practical point of view, the pilot experiments we conducted have confirmed that the cognitive load required by the manual annotation of temporality is reduced if the phrase-based structure of the utterances is displayed to the annotator. Cognitively speaking, the annotation of temporality seems to require a syntactic disambiguation in most of the cases. In addition, the definition of temporal relations such as SLINK is directly resolved with constituent trees.

This paper presents Contemplata as an annotation platform, with an emphasis on its ability to provide for any linguistic annotation, including temporal. After shortly describing related work, we proceed with an overall description of the software architecture and the software interface. The following section explains how the platform is used in practice to annotate, describing successively the roles of the administrator, the annotator and the adjudicator. In the last sec-

tion, we show the outcomes of two projects whose annotations were conducted with Contemplata, Temporal@ODIL and RAVIOLI, focusing on temporality annotation for the first, and on the syntactic characterisation of injunctive utterances for the second.

2. Related Work

An existing tool that matches our needs the closest is TrEd[1](Hajič et al., 2001), a highly configurable (via PML schemas, stylesheets, macros, configuration files) tree edition tool. It has been primarily used for dependency annotations, but can be also configured (via extensions) to annotate/modify constituency trees. TrEd allows editing node attributes and assigning complex feature structures to them, as well as adding references from nodes to other nodes. It would thus allow to handle, after developing a dedicated extension, both the multiple annotation layers of Temporal@ODIL and their complex features, and the references between temporal entities (although, in contrast to Contemplata, TrEd does not allow to assign features to arcs between nodes, nor is it able to appropriately visualize references between nodes from different sentences). However, TrEd does not provide any support for spoken data characteristics, such as speech turns and phatic expressions: the former can be merged or split into several sentences, while the latter are discarded in the trees but kept in the context window, see Sec. 3.1.2. Moreover, TrEd is a standalone application and cannot be run through a web-browser, which is a drawback for a large annotation campaign.

3. Software Description

Contemplata is a platform designed to provide the functions necessary for a syntactic and/or semantic annotation campaign. This section describes how the platform is built for this purpose and to what extent it can be adapted for a new campaign. It is noteworthy that while some functions are adapted to speech corpora, Contemplata is not dedicated to speech and does not integrate audio/video playback.

3.1. Software Architecture

The tool is implemented in a client/server architecture. The front-end annotation tool is written in Elm[2], which compiles to JavaScript, thus the tool can be used in any modern internet browser. The client annotation tool communicates with a Haskell[3] server via websockets, with the annotated files serialized to JSON before being sent. Thanks to the use of strongly-typed programming languages, annotation data can be represented with appropriate data types on both sides and many of its well-formedness properties checked already at compile time. This reduced the chance of creating and storing malformed data in the database.

The client/server architecture has a couple of advantages within the context of an annotation project. The annotator does not have to install anything locally, and the server can provide the user with more advanced functionality – for instance, syntactic parsing. In the long run, this architecture should also allow a more collaborative annotation style.

[1] https://ufal.mff.cuni.cz/tred/

[2] http://elm-lang.org/

[3] https://www.haskell.org/

3.1.1. Data Types Specification

Contemplata has the ambition of being easy to adapt to different annotation tasks which involve labeling syntactic nodes and relations between them with structured information. To this end, Contemplata uses Dhall[4] configuration files, which notably define:

- The list of non-terminal and terminal categories which can be assigned to syntactic nodes.
- The annotation entities, i.e., the objects (events, time expressions, etc.) with which the nodes in syntactic trees can be marked. The corresponding attributes, the attributes' types, and potential values, are also defined in the configuration.
- The annotation relations and the corresponding attributes, i.e., the objects which can connect two different nodes belonging to two different syntactic trees.

For instance, the following configuration extract (simplified) defines an entity called `Timex`, which can be assigned to syntactic nodes (or relations, depending on the remaining configuration). It specifies the possible Timex types (`["Date", "Time", "Duration"]`), the default type when the entity is created (`Time`),[5] and the possible attributes of a Timex: free-text `Value`, optional attribute `Mod` whose value has to belong to `["Before", "After"]`, as well as a special anchor attribute called `Anchor`. Anchor attributes function as place-holders for references to other syntactic nodes, to be specified during annotation. Finally, two anchor attributes (`Begin` and `End`) are defined as type dependent – they should be only specified if the type of the Timex is `Duration`. For instance, this allows to capture the relation between the Timex *from 12:00 to 13:00*, on the one hand, and *12:00* (`Begin`) and *13:00* (`End`), on the other hand, in the sentence *he was running from 12:00 to 13:00*.

```
{ name = "Timex"
  -- Possible types
, typ =
    { among = ["Date", "Time", "Duration"]
    , def = ["Time"] : Optional Text
    }
  -- Attributes
, attributes =
    [ { name = "Value"
      , value = Free {def = None Text}
      }
    , { name = "Mod"
      , value = Closed
        { among = ["Before", "After"]
        , def = None Text
        , required = False }
      }
    , { name = "Anchor"
      , value = Anchor
      }
    ]
  -- Type-dependent attributes
, attributesOnType =
```

[4] https://dhall-lang.org/

[5] In practice, the most frequent value is selected to avoid manual revision.

```
    [ { typ = "Duration"
      , attributes =
        [ { name = "Begin"
          , value = Anchor
          }
        , { name = "End"
          , value = Anchor
          }
        ]
      }
    ]
} : ./Entity.typ
```

The configuration is stored on the server and sent to the front-end once the annotator decides to work on a specific file. The front-end annotation tool then ensures that all created entities conform the configuration.[6] Other properties of the framework – e.g., the specification of command invocations (the corresponding menu items, command-line instructions, keyboard shortcuts, help messages, and so on) – are also specified in the configuration files.

3.1.2. Data Model

As a framework adapted to processing and annotation of spoken data, Contemplata allows to:

- Exclude irrelevant tokens (for instance, *ah* and other speech-related interjections) from subsequent (syntactic, temporal, etc.) processing and annotation.[7]
- Split and merge speech turn elements. Merging is useful when a single syntactic unit happens to span several speech turns due to speech overlap, as in Fig. 1. In this example, we notice that the left part of the figure displays a regular syntactic tree, while the right part shows that a verbal nod (*oui* 'yes') actually cuts the unit in two parts. The tree shown is in fact the result of the merge command used on two originally separated parts: *c'est toujours plus* 'it is always more' and *plus vite* 'more fast [faster]'.

Besides, in order to better adapt the input to subsequent automatic processing (parsing, in particular), split and merge operations can be performed over sentence tokens.

All this leads to the data model depicted as a relational diagram in Fig. 2. In this model, the set of `Sentences`[8] in the `File` is divided into `Partitions`, with one syntactic tree (`SynTree`) being assigned to each partition. Each `Sentence` consists of a sequence of `Tokens`. In order to allow for subsequent projection of the annotated data on the original corpus (the set of sentences, grouped into speech `Turns`), relations between syntactic leaves (one kind of `SynNodes`) and `Tokens` are preserved. The set of speech turns in the file, kept independently from the partitions, allows for an adequate visualisation of speech turns. Node

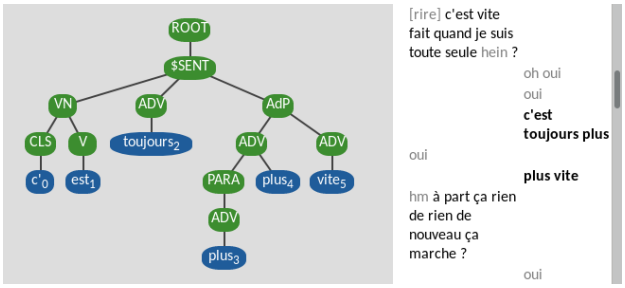

Figure 1: Example of two speech turns (*c'est toujours plus* 'it's always more' and *plus vite* 'more quickly') merged back into one syntactic unit.

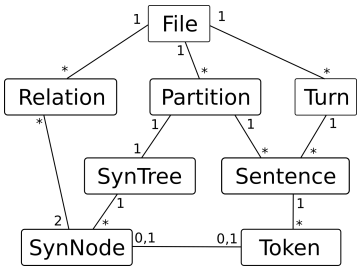

Figure 2: Relational diagram of the data model (excluding annotation entities)

annotations (not depicted) are assigned directly to (non-leaf) syntactic nodes, and `Relation`-related annotations are stored in the scope of the entire file, since they can link nodes occurring in different syntactic trees. Finally, due to the special anchor attributes (see Sec. 3.1.1), additional links (not depicted either) can be defined between syntactic nodes, as well as between (linguistic) relations and nodes.

At the level of implementation, Contemplata defines the data types/structures corresponding to the annotated objects at different levels: files, sentences, syntactic trees, entities, links, etc. For instance, the following extract contains a (simplified) Haskell definition of the constituency tree:[9]

```
-- | Syntactic tree: multi-way tree labeled
-- with syntactic nodes (see below)
type SynTree = Tree SynNode

-- | Syntactic node (internal node or leaf)
data SynNode
  = InternalNode
    { nodeId :: NodeId
      -- ^ Unique ID (within tree scope)
    , nodeVal :: Text
      -- ^ Value assigned to the node
    , nodeEnt :: Maybe Entity
      -- ^ Annotation of the node (if any)
    }
  | LeafNode
    { leafId :: NodeId
      -- ^ Leaf's equivalent of `nodeId`
    , leafVal :: Text
```

[6] It should be noticed that, in order to use Contemplata for two different annotation projects with different annotation configurations, it is necessary to run two instances of the server.

[7] The list of stop words that should be automatically discarded is application-specific and can be specified in the configuration.

[8] Speech utterances, typically, but we call them sentences for the sake of genericity. The framework can be very well applied to written corpora, in which case `Sentence`, `Partition`, and `Turn` would be all conflated to a single entity.

[9] All Haskell (back-end) data types have their Elm (front-end) equivalents, to ensure seamless data exchange. We plan to avoid this code duplication in future versions.

```
    -- ^ Value assigned to the leaf
  , leafPos :: Int
    -- ^ Position of the leaf in the
    -- sentence
  }
```

Each syntactic node (`SynNode`) is associated with a value, non-terminal in case of internal nodes (`nodeVal`) and terminal word forms in case of leaves (`leafVal`). The set of possible non-terminals is specified in the configuration (see Sec. 3.1.1). The forms assigned to leaves can differ from the corresponding tokens – namely, the framework allows to edit them manually to facilitate parsing.

Additionally, each node has an ID (`nodeId` and `leafId`), unique within the scope of the tree. To address syntactic nodes globally (i.e., within the scope of the file), tree IDs are combined with node IDs. Each node can be also associated with an annotation entity (`nodeEnt`), which corresponds to one of the entities specified in the configuration.[10]

For each leaf, the position of the leaf in the sentence is specified. Since not every token is required to be referenced from the tree (we allow to discard tokens, see above), we need this information in order to be able to recover the relationship between leaves and the original sentence tokens. Secondly, this allows to represent non-projective trees (i.e., trees with discontinuous constituents). Namely, the framework allows to change the order of the leaves in the tree while preserving the mapping between them and sentence tokens. For instance, in Fig. 3, the original sentence was *parce que* les *comment vous expliquer les les créanciers* ('because *the* how to explain this to you the the creditors'), as shown in the `context` window in the right part of the figure, where the determiner *les* ('the') first appears detached ahead of the `NP`. However, for the sake of the syntactic annotation, the determiner was interchanged with the `Sint` node. Contemplata automatically detects the parts that are displaced by comparing the original and the final position of each token and marks them in pink. The original order of the tokens is still preserved, as can be seen in the token nodes: *les* ('the') is still in position 110, even though its node is now between leaves 113 and 114.

3.1.3. File Formats

Contemplata uses automatic serialization procedures to generate easily-exchangeable JSON representations of data annotations (see Sec. 3.1.2).

The input should be text (transcribed speech in the case of oral data), either in a GLOZZ (Widlöcher and Mathet, 2012) or a Penn Treebank format. In the first case, the file is automatically converted to a JSON format when uploaded; in the latter case, the file has to be converted beforehand to a JSON format using a command line tool that comes with Contemplata. The generated files are therefore JSON files which can be later converted to an annotation-specific format (e.g., Penn Treebank, CoNLL or XML-based TimeML standard). To ensure that annotated data is consistent with the annotation-specific standard, appropriate configuration (see Sec. 3.1.1) should be used. For now, we provide a Python script to convert Contemplata's JSON data into Penn Treebank's syntactic bracketed format. Other output formats will be available in a future release vesion.

3.1.4. Parsing

One of the main features of Contemplata is that it allows to syntactically re-parse a given sentence in a way which takes the constraints specified by the annotator (e.g. a particular tokenization) into account. This allows a more interactive annotation style, in which the most obvious errors are first corrected manually, thus making the parsing task easier. After re-parsing, the annotator is confronted with a better-quality constituency tree (see also Sec. 4.2).

Currently, two syntactic parsers are integrated in the framework: the Stanford parser (Green et al., 2011) and Disco-DOP (van Cranenburgh et al., 2016). To use a particular parser, it should be installed and run on the server (although using it remotely is also an option). Communication with the Stanford parser relies on Protocol Buffers, with the benefit of having the parsing results automatically converted into the Contemplata's data structures. Parsing results retrieved from the Disco-DOP's web server, on the other hand, follow the Penn Treebank-style bracketed format, which has to be additionally parsed into the Contemplata's tree data structures.

It is possible to specify additional parsing constraints, e.g., a particular tokenization, POS tags, or (in case of Disco-DOP) the set of constituency nodes (span/label pairs) that the resulting tree should contain. Such constraints are sent as a part of the HTTP request to the parser's server together with the sentence to be parsed.

3.2. Software Interface

The interface of the framework is divided into two independent, switchable parts. The first one is the overview page (see Fig. 4), which provides the annotator with an overview of the files that she or he can explore and/or annotate, depending on the assigned access level. The files are divided into three categories: `waiting` (annotation of the file not started), `in progress` (annotation started), and `done` (annotation finished). The annotator can choose to annotate a single file, or select two or more files for comparison and, possibly, adjudication.

The second, core part of the interface is the actual annotation mode. In this mode, the workspace consists of two vertically arranged annotation windows, showing two syntactic trees assigned to two (typically different) sentences (technically, partitions, see Sec. 3.1.2) in a given file, as shown in Fig. 5. To each workspace, the corresponding side window is allocated on the right, which allows to (a) `edit` the attributes of the selected node (specified in the configuration, see 3.1.1), (b) see the `context` of speech turns and sentences in which the given sentence occurs, and (c) inspect the `messages` received from the Contemplata back-end server. The context window in particular consists of two or more columns, depending on the number of speakers, with the currently selected sentence/partition marked in bold. The annotator can merge the selected sentence with another one into a single partition by `CTRL`-

[10] This also illustrates one of the limitations of the current version of framework – it is not possible to assign two or more entities to a single node, nor is it possible to assign an entity to a group of nodes (a workaround using relations is possible, though).

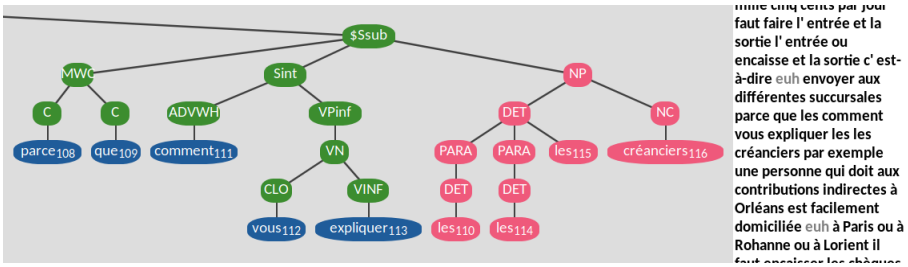

Figure 3: Example of a non-projective tree (*parce que les comment vous expliquer les les créanciers* 'because the how to explain this to you the the creditors')

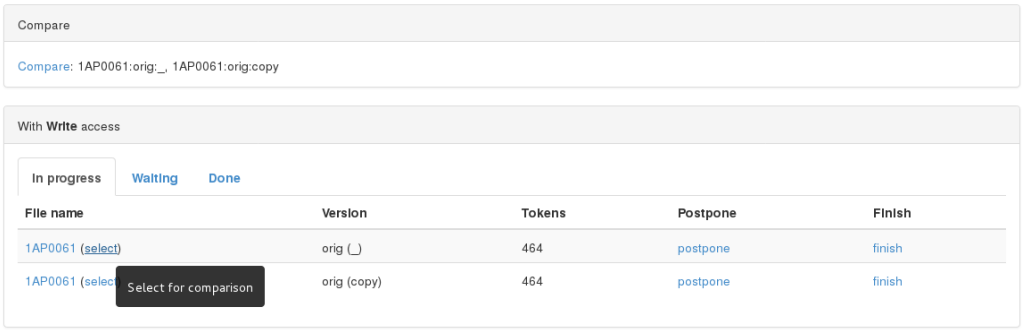

Figure 4: Overview page, showing the files available for annotation and/or inspection.

clicking on the second sentence in the context window. All discarded words, which are not present in the syntactic tree, are displayed in grey.

The size of the four windows can be easily adapted using the arrows: `left` or `right` to move the vertical line separating the workspaces, `up` or `down` to move the horizontal line between the workspaces and the side windows. This allows to, e.g., easily adapt the tool to a single-file annotation mode, in which case the horizontal line can be moved to the very bottom of the page. In case of the comparison mode, the two windows can display syntactic trees from two different files. The files being edited are shown in the bottom-left corners of the respective workspaces, and can be changed by clicking on the corresponding file names.

A menu with the main commands is placed at the top of the upper workspace. Apart from the general commands (`menu` to go to the overview page, `save` to send the file to the back-end server and store it in the annotation database), one of the annotation sub-modes can be selected: `segmentation`, `syntax`, and `temporal`.[11] Each sub-mode comes with its own set of dedicated menu commands, shown on the right. For instance, in Fig. 5, the `segmentation` sub-mode comes with four segmentation-related commands: `restart`, `split sentence`, `split word`, and `join words`.

The division of the menu commands into sub-modes serves to better organize the annotation at different linguistic levels. However, all the commands are also available via the command line, and for the more frequently used commands keyboard shortcuts are provided (e.g., `p` to syntactically `parse` the sentence). The command line is invoked with `space` and allows to type commands literally, with support of auto-completion.

As mentioned before, one of the primary functionalities provided by the tool is to allow manual correction of syntactic trees. To this end, the tool allows the annotator to perform several structure-modifying operations: adding and deleting nodes, changing the parent of a node, changing the position of the node w.r.t. its parent, etc. Only the operations which preserve the well-formedness of syntactic structures are allowed.

All the tree editing operations rely on the concept of node selection. The annotator can select, using the mouse, one primary node and several secondary nodes in each workspace. The semantics of the selection depends on the command. For instance, the `add` command adds a new node (with under-specified non-terminal symbol) over each selected node in the current workspace. In case of the re-attachment command, the sub-tree rooted at the main node gets re-attached to the secondary node.

While the syntax-related editing commands apply to the current workspace, there are also commands which apply to the nodes selected in the two workspaces. For instance, the `TLink` command in the `Temporal` annotation sub-mode allows to connect the two main nodes with a directed temporal relation. The relation itself can be selected and its attributes modified, as shown below in Fig. 6.

Finally, Contemplata keeps track of all the editing operations performed during the annotation of a file (or a set of files). These modifications can be easily reverted using the

[11] In principle, the annotation sub-modes – together with the corresponding commands – could be customizable in the configuration. Currently, however, the sub-modes depend on a particular Contemplata instantiation, adapted to a specific annotation task.

undo command, and restored using the redo command (`z`).

4. Annotating with *Contemplata*

This section describes how Contemplata can be used to annotate a corpus, starting from the different roles to be considered to the actual annotation process.

4.1. Role Distribution

Working on a corpus rarely involves only one person, and even when it does, this single person has to play different parts. Contemplata provides a specific interface for each of those roles. In this section, we first describe the different roles that are implemented, before going through a classic annotation process with Contemplata.

4.1.1. Administrator

Every project needs a leader and every system needs an administrator. In our case, both roles can be fulfilled when using the "admin" account, which has all rights. Under the hat of the administrator, one can:

- create and delete regular users accounts;
- upload files to the database of the project;
- assign files to users and give them either reading or editing rights to those files.

It is noteworthy that on Contemplata, while regular users accounts can be used for either the annotation part or the adjudication part, they cannot be given administration rights. Therefore, if a project has several leaders, all of them have to log in to the admin account to do actions that are permitted only to the administrator.

4.1.2. Annotator(s)

Regular user accounts are to be used by annotators. Unlike the admin account, they do not come with the tool and have to be created by the administrator.
Whether annotators can read or edit the files of the database depends on the rights they were given for each sample of the corpus by the administrator.

4.1.3. Adjudicator(s)

Annotation is a tedious task that requires an acute attention and a full understanding of quite complex guidelines. It is therefore virtually impossible to produce a perfect annotation, but it is possible to reduce the number of errors by reviewing the annotation. To this end, the third role implemented in Contemplata for an annotation process is the role of adjudicators, i.e., users who are in charge of the review of annotations once they are done.
The roles of annotator and adjudicator are not mutually exclusive: any regular user account can be assigned the role of adjudicator on any file, while keeping their role of annotator on other files.

4.2. Corpus Annotation Process

This section describes the actual process of annotation using Contemplata, from the attribution of files to the adjudication of the annotation. At this point, we assume that a new project was created, that corpus samples were uploaded and that user accounts for each participant of the project were created using the Admin account.

4.2.1. File Attribution

The very first step of an annotation process using Contemplata is the attribution of files to annotators and adjudicators. As mentioned before, this can be done only via the Admin account. Leaders of the project therefore have to log in to the Admin account, select each file and assign them one or several users for annotation or adjudication.
When they log in, annotators are redirected to the overview page (Fig. 4). Depending on the rights they were given to a file, they can see it either in the "With Write access" table or the "With Read access" table. To help annotators keep track of their work, each table has three tabs: "In progress", "Waiting" and "Done", as described in Sec. 3.2 Annotators then have to choose a file and click on its name to open the annotation mode (Fig. 5).

4.2.2. Annotation

Contemplata was built to provide a platform for any type of annotation that needs to be grounded on syntactic trees. Before proceeding to any further layer of annotation, the four following steps of syntactic annotation must be completed:

1. a preliminary automatic syntactic annotation - the annotator uses the command `parse` to annotate a speech turn using a syntactic parser.
2. a manual revision of utterance segmentation and POS tagging - where the parser completely fails, the annotator is asked to provide the appropriate revision: she or he can either merge speech turns (see Sec. 3.1.2 and Fig. 1), split a speech turn into several sentences and/or revise the POS tags directly.
3. a final automatic syntactic annotation - using the `parse` command again, the annotator runs a second round of parsing, this time taking into account the newly revised segmentation, as well as the POS tagging (command `parsepos`) if necessary.
4. a manual revision of constituent trees - this last step consists in the revision of the residual errors (if any) in the constituent trees.

In (Wang et al., 2020), the syntactic annotation process is thoroughly depicted with five steps, adding the automatic pre-processing of files as a 'step zero'. This step consists in the sidelining of irrelevant tokens (noises, interjections, and phatic expressions) as described in Sec. 3.1.2. They would appear greyed out in the `Context` window but are not shown in the trees (see Fig. 1).
As soon as the syntactic annotation is validated, the annotators may proceed to a second layer of annotation (semantic relations, temporal entities...) depending on the objectives of the project. We are illustrating two possibilities in Sec. 5: the case of temporal annotations (Temporal-@ODIL project) and the case of the annotation of injunctions (RAVIOLI project).

4.2.3. Adjudication

If needed, once annotators consider that their work is done and hit the Finish button (see Fig. 4) for a given file, this file appears in the 'Done' tab for them but in the 'Waiting' tab

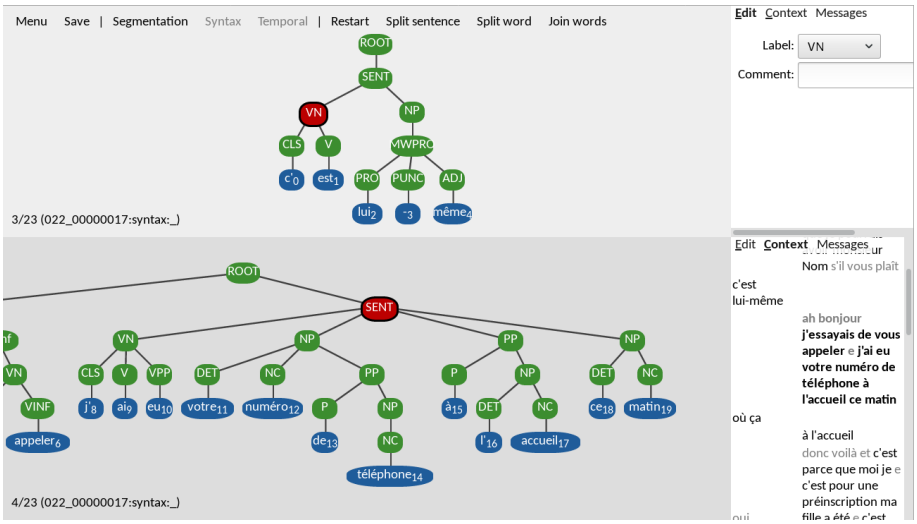

Figure 5: Annotation mode, with two main (top and bottom) workspaces showing the syntactic trees, and two (top and bottom) corresponding side windows.

for the assigned adjudicator(s)[12]. To revise this file, users who were given the role of adjudicator can select the annotated file(s) they were attributed in the overview page (still Fig. 4). If there were more than one annotator for a given file, they may select two annotated files and compare them directly thanks to the interface showing two workspaces simultaneously (Fig. 5).
After the approbation or revisions of the adjudicator(s), annotators can proceed with further annotations if there were several layers of annotation.

5. Current Applications

5.1. Syntactic and temporal Annotation

As mentioned in the introduction of this paper, Contemplata was built to provide an annotation platform that allows temporal annotations to be grounded on constituent trees for a spoken corpus.
Contemplata therefore displays specific features for temporal annotation:

- marks for temporal entities such as events, time expressions (`Timex`) and signals - with specific attributes for each of them;
- links for temporal relations between temporal entities, such as subordination links (`SLINK`) or temporal links (`TLINK`).

Those features can be added to syntactic nodes when relevant, as shown in Fig. 6. In this example, two nodes have been marked as temporal entities (orange nodes): the sentence node as an event (hence the subscripted `EV`) and the `NP` *ce matin* ('this morning') as a `Timex` (subscripted `TI`). On top of that, a red arrow shows that there is a temporal relation (`TLINK`) between the `Event` and the `Timex` (in this case, they are simultaneous as the Event occurred at the time of the Timex).

[12]To be implemented in the next version of Contemplata.

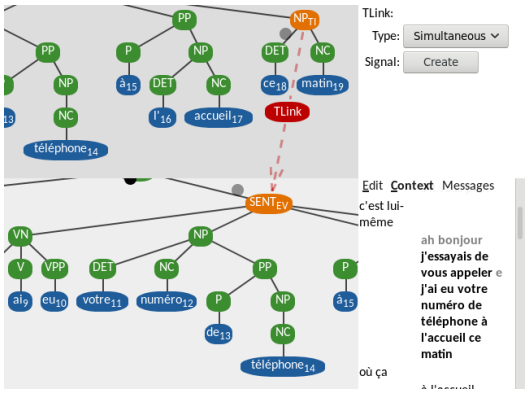

Figure 6: Annotation of an `Event` and `Timex`, linked by a temporal relation (`TLINK`) (*j'ai eu votre numéro de téléphone à l'accueil ce matin* 'I got your phone number at the reception this morning')

5.2. Visualization of Injunctions

The second project that was partly conducted on Contemplata is the RAVIOLI[13] Project. The main objective of this project is to automatically identify, characterize and categorize injunctions, whether they are expressions of an order, an advice, an instruction or a request, using machine learning techniques. To this end, a manually annotated corpus is used as the training corpus of a supervised model which is supposed to use both signal-related features and (morpho)syntactic features. As a matter of fact, one of the research questions is to investigate whether or not injunctions do coincide with syntactic units.
To provide (morpho)syntactic features as well as an answer to the above-mentioned research question, the corpus used

[13]Stands for **R**econnaissance **A**utomatique des **V**aleurs **I**njonctives à l'**O**ral, **L**angue en **i**nteraction, described in `http://tln.li.univ-tours.fr/Tln_Ravioli.html`

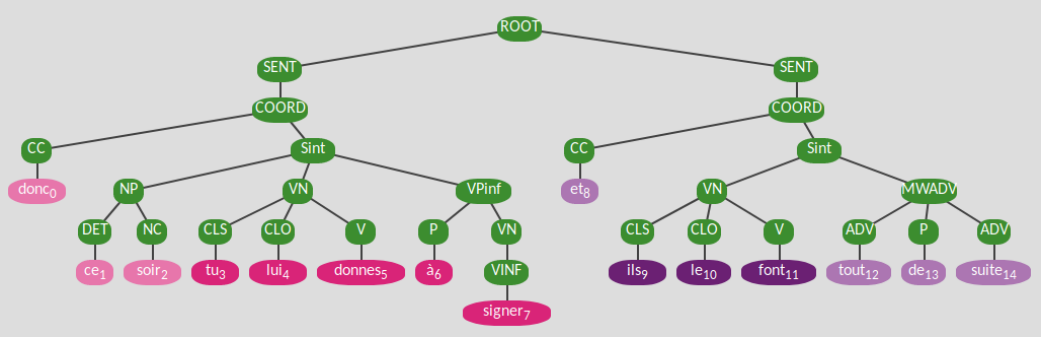

Figure 7: Annotation of two injunctions for the RAVIOLI Project (*donc ce soir tu lui donnes à signer* 'so tonight you give them to sign' and *et ils le font tout de suite* 'and they do it straight away')

in RAVIOLI, which is a spoken French corpus[14], has to be parsed. Contemplata is an interesting annotation platform for RAVIOLI because it is built to handle the particularities of spoken language, is easily adaptable to a new project and offers further possibilities of annotation. For instance, if constituents prove to be interesting units for injunctions, the semantico-pragmatic annotation needed to characterize injunctions can also be carried out on Contemplata.

Fig. 7 shows how Contemplata was used to visualize the semantically annotated corpus of RAVIOLI. In the original corpus, injunctions were annotated with two units: the injunction nucleus (*noyau injonctif*) and the injunction as a whole (*intervention injonctive*), including the nucleus and peripheral elements. One of the main requirements of the use of Contemplata for RAVIOLI was to allow to visually identify these two units at a glance, while preserving a clear distinction between injunctive elements and non-injunctive elements. We therefore used the node coloring function of Contemplata, already implemented for non-projective trees (see Fig. 3), with additional contrasting colors: tokens that are not part of an injunction remain blue as in the original Contemplata configuration (not shown here), while tokens pertaining to an injunction can be alternatively magenta or purple, with the nucleus being bright and the peripheral elements (if any) being toned down.

6. License

Contemplata is freely distributed under the BSD license. The code is available in the dedicated git repository.[15]

7. Ackowledgements

The two projects presented here, Temporal@ODIL and RAVIOLI, were both funded by the Council of the Centre Val de Loire Region (as APR-IA).

8. Bibliographical References

Bejček, E. and Straňák, P. (2010). Annotation of Multiword Expressions in the Prague Dependency Treebank. *Language Resources and Evaluation*, 44(1-2):7–21.

Bittar, A., Amsili, P., Denis, P., and Danlos, L. (2011). French TimeBank: an ISO-TimeML Annotated Reference Corpus. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers (ACL'2011)*, pages 130–134, Portland, USA.

Green, S., de Marneffe, M.-C., Bauer, J., and Manning, C. D. (2011). Multiword expression identification with tree substitution grammars: A parsing tour de force with french. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 725–735, Stroudsburg, PA, USA. Association for Computational Linguistics.

Hajič, J., Vidová-Hladká, B., and Pajas, P. (2001). The prague dependency treebank: Annotation structure and support. In *Proceedings of the IRCS Workshop on Linguistic Databases*, pages 105–114.

ISO. (2012). Language Resource Management - Semantic Annotation Framework (SemAF) - Part 1: Time and Events. ISO 24617-1:2012. *International Organization for Standardization*.

Lefeuvre-Halftermeyer, A., Antoine, J.-Y., Couillault, A., Schang, E., Abouda, L., Savary, A., Maurel, D., Eshkol-Taravella, I., and Battistelli, D. (2016). Covering various Needs in Temporal Annotation: a Proposal of Extension of ISO TimeML that Preserves Upward Compatibility. In *Proceedings of LREC'2016*, pages 3802–3806, Portorož, Slovenia.

van Cranenburgh, A., Scha, R., and Bod, R. (2016). Data-oriented parsing with discontinuous constituents and function tags. *Journal of Language Modelling*, 4(1):57–111.

Wang, I., Pelletier, A., Antoine, J.-Y., and Halftermeyer, A. (2020). ODIL_Syntax, a Free Spontaneous Spoken French Treebank Annotated with Constituent Trees. In *Proceedings of LREC'2020*, Marseille, France.

Widlöcher, A. and Mathet, Y. (2012). The glozz platform: A corpus annotation and mining tool. In *Proceedings of the 2012 ACM symposium on Document engineering*, pages 171–180.

Zinsmeister, H. and Dipper, S. (2010). Towards a Standard for Annotating Abstract Anaphora. In *Proceedings of LREC'2010*, pages 54–59, Valletta, Malta.

[14]The source corpus is ESLO2, the second series of **E**nquêtes **S**ocio**L**inguistiques à **O**rléans.

[15]`https://github.com/contemplata/contemplata`