

# Glozz

## User's Manual

<http://www.glozz.org>

Version : **1.0**  
Date : May the 14'th, 2011  
Author : Yann Mathet

*Glozz was created within the french ANR project Annodis by (in alphabetic order) Yann Mathet and Antoine Widlöcher  
mathet@unicaen.fr, widlocher@unicaen.fr*

# Table of contents

1.	Introducing Glozz.....	5
1.1.	Meta-Model : the Unit-Relation-Schema generic model .....	5
1.1.1	Units .....	5
1.1.2	Relations .....	6
1.1.3	Schemas.....	6
1.2.	Annotation Model : a Unit-Relation-Schema instantiation for a given annotation campaign.....	7
1.3.	Feature-Sets.....	8
1.4.	Annotation IDs in Glozz : « real IDs » versus « friendly IDs ».....	9
1.4.1	Real IDs.....	9
1.4.2	Friendly IDs.....	9
2.	User interface .....	10
2.1.	overview .....	10
2.2.	Main and macro views: viewing and navigating.....	10
2.3.	Annotating how-to: adding and editing annotations.....	11
2.3.1	Units .....	12
2.3.2	Relations .....	14
2.3.3	Schemas.....	15
2.3.4	Working with « real IDs » or « friendly IDs » .....	17
2.4.	Annotations as predicates (Frame 6) .....	18
2.4.1	Overview .....	18
2.4.2	Navigation and selection .....	19
2.4.3	Sorting the list.....	20
2.4.4	Managing visibility for each individual annotation.....	20
2.4.5	Command line feature: creating annotations via a predicate entered with the keyboard.....	20
3.	Installing and getting started how-to .....	22
3.1.	Download and unpack Glozz .....	22
3.2.	Launch Glozz.....	22
3.3.	Create a shortcut on your desktop .....	22
3.4.	Choose a login and ask for a key .....	22
3.5.	Creating your own data folder(s).....	23
4.	File types overview.....	24
5.	Creating a corpus in Glozz format.....	25
5.1.	Manual corpus creation from a txt file via « import txt » menu.....	25
5.2.	Automatic corpus creation via a custom program .....	29
6.	Annotation Models : types, feature-sets, groups.....	31
6.1.	Overview .....	31
6.2.	Units .....	31
6.3.	Relations.....	32
6.4.	Schemas.....	33
6.5.	Groups.....	33
7.	Styles.....	35
7.1.	Overview .....	35
7.2.	Creating styles .....	35
7.3.	Unit styles.....	36
7.4.	Relation styles .....	38

7.5.	Schema styles .....	38
7.6.	Special features.....	39
7.6.1	StyleSheet mode.....	39
7.6.2	Individual colors modes .....	39
7.6.3	Co-reference chain color mode.....	40
8.	GlozzQL : Glozz Query Language.....	42
8.1.	GlozzQL fundamentals .....	42
8.1.1	Constraint .....	42
8.1.2	ConstrainedAnnotation.....	43
8.1.3	Incremental creation of Constraints and ConstrainedAnnotations .....	43
8.2.	GlozzQL Graphical User Interface.....	44
8.3.	Some examples .....	45
8.3.1	Units containing and not containing a text .....	45
8.3.2	Focusing on one particular annotator.....	48
8.3.3	Deeper queries with schemas and relations.....	49
8.4.	Saving and loading GlozzQL queries .....	52
8.5.	GlozzQL Basket .....	53
8.5.1	Feeding the basket.....	53
8.5.2	Save basket content as .aa file .....	54
8.5.3	Erase basket content from current data .....	55
8.6.	Advanced concepts .....	55
8.6.1	Two ways constraints.....	55
8.6.2	Unification mechanism .....	56
8.7.	Use cases .....	60
8.7.1	Splitting annotations by authors, types, etc .....	60
8.7.2	Looking for mistakes.....	60
8.7.3	Basic statistics.....	60
9.	« Grapher » : annotations shown as a graph .....	61
9.1.	Overview .....	61
9.2.	Interface.....	61
9.2.1	Launching Grapher .....	61
9.2.2	Auto-selection.....	62
9.2.3	Zoom-in, Zoom-out.....	62
9.2.4	Showing embedded text .....	62
9.3.	SDRT layout.....	62
9.4.	Co-reference chains layouts .....	63
9.4.1	Co-reference using relations.....	63
9.4.2	Co-reference using schemas .....	63
10.	« Glozz Aligner » : Alignment and agreement tool for multi-annotated texts.....	65
10.1.	Principles .....	65
10.2.	Aligner special view .....	65
10.2.1	overview .....	66
10.2.2	Zooming-in, zooming-out.....	66
10.2.3	Moving to the right or to the left.....	67
10.2.4	Changing the order the annotators appear .....	67
10.2.5	Adjusting units positions .....	68
10.3.	Alignment tool.....	69
10.3.1	Auto alignment.....	69
10.3.2	Manual alignment .....	70
10.3.2.1	Choosing special mode .....	70

10.3.2.2	Creating a new alignment.....	70
10.3.2.3	Editing an alignment.....	71
10.4.	Agreement measurement.....	71
11.	Additionnal tools .....	73
11.1.	Depth selector .....	73
11.2.	Simple search tools .....	74
11.2.1	« Find Text » Tool .....	74
11.2.2	« Find units » tool .....	75
11.3.	« TimePlayer ».....	76
11.3.1	Main principle.....	76
11.3.2	True time option.....	77
11.3.3	Auto-replay.....	77
11.3.4	Caution .....	77
Appendices.....		78
Appendice 1 : hashcode algorithm in Java .....		78
References.....		79

Remark : what we call « corpus » in Glozz and in this manual is an annotated text, not a collection of annotated texts, as it usually means.

## 1. Introducing Glozz

Glozz is a multi-purpose annotation tool, which can be set to cope with most of paradigms. It has been developed since september of year 2008 within the french ANR Annodis project (involving Clle-Erss, Greyc and Irit laboratories), by, in alphabetic order, Yann Mathet and Antoine Widlöcher from the Greyc. A third developper, engineer Jérôme Chauveau, has joined the development team since october 2010 for some months.

It comes with a fully WYSIWYG<sup>1</sup> interface which makes it possible to annotate texts with rich annotation models, and provides additionnal features such as the query language GlozzQL and some exports features (SQL, CSV).

### 1.1. Meta-Model : the Unit-Relation-Schema generic model

Glozz relies on the URS (Unit-Relation-Schema) meta model from Widlöcher. This model defines 3 meta types of elements as follows :

#### 1.1.1 Units

A unit is a contiguous span of text starting at one character position and finishing at another one. Units can overlap each other or even cover others :

lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Ut velit mauris, egestas sed, gravida nec, ornare ut, mi. Aenean ut orci vel massa suscipit pulvinar. Nulla sollicitudin. Fusce varius, ligula non tempus aliquam, nunc turpis ullamcorper nibh, in tempus sapien eros vitae ligula. Pellentesque rhoncus nunc et augue. Integer id felis.

In the figure above, we can see first two separate units, then two overlapping ones, and at last some covering ones. Note that when a unit covers another one, it is visually shown in Glozz through a covering frontier. Hence it is possible to see which unit contains which others.

---

<sup>1</sup> WYSIWYG : what you see is what you get, or, in other words, you directly work on the screen with the data as they will appear at final stage.

Units can be considered as the first type of elements from which others (relations and schemas) can be built.

### 1.1.2 Relations

A relation is a link, oriented or not, from one element of the URS model (a unit, a relation or a schema), to another element.

Typically, it may involve two units as follows :

Lorem ipsum dolor sit amet, **consectetuer** adipiscing elit. Sed non risus. Suspendisse  
lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum  
ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin  
porttitor, orci nec nonummy molestie, enim **est eleifend mi, non fermentum** diam nisl sit

But it may link a unit to a relation :

Lorem ipsum dolor sit amet, **consectetuer** adipiscing elit. Sed non risus. Suspendisse  
lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum  
ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin  
porttitor, orci nec **nonummy** molestie, enim **est eleifend mi, non fermentum** diam nisl sit

Or a schema to another schema (see next section to see what a schema is) :

Lorem ipsum dolor sit amet, **consectetuer** adipiscing elit. Sed non risus. Suspendisse  
lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum  
ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin  
porttitor, orci nec **nonummy** molestie, enim **est eleifend mi, non fermentum** diam nisl sit  
amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim.  
Pellentesque congue. Ut in risus volutpat **libero** pharetra tempor. Cras vestibulum  
bibendum **augue. Praesent** egestas leo in pede. Praesent blandit odio eu enim. Pellentesque  
sed dui ut augue blandit sodales. Vestibulum **ante** ipsum primis in faucibus orci **luctus et**

And so on : Relation to Relation, Relation to Schema, Unit to Schema...

### 1.1.3 Schemas

A schema is a set of as many URS elements as wished. Hence, a given schema can contain some units, but also some relations, and even some other schemas. This enable to construct recursively deep structures.

Let's see some possible configurations :

Lorem ipsum dolor sit amet, **consectetuer** adipiscing elit. Sed non risus. Suspendisse  
 lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum  
 ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin  
 porttitor, orci nec **nonummy** molestie, enim **est eleifend mi, non fermentum** diam nisl sit  
 amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim.  
 a schema embedding 3 units

Lorem ipsum dolor sit amet, **consectetuer** adipiscing elit. Sed **non** risus. Suspendisse  
 lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum  
 ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, **mi**. Proin  
 porttitor, orci nec **nonummy** molestie, enim **est eleifend mi, non fermentum** diam nisl sit  
 amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim.  
 a schema embedding 3 units and a relation

Lorem ipsum dolor sit amet, **consectetuer** adipiscing elit. Sed **non** risus. Suspendisse  
 lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum  
 ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, **mi**. Proin  
 porttitor, orci nec **nonummy** molestie, enim **est eleifend mi, non fermentum** diam nisl sit  
 amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim.  
 Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum  
 bibendum augue. **Praesent** **egestas leo, in pede. Praesent blandit odio** eu enim. Pellentesque

a schema embedding 3 units, a relation and a schema

## 1.2. Annotation Model : a Unit-Relation-Schema instantiation for a given annotation campaign

The 3 kinds of elements which are used in Glozz are those just presented above, Units, Relations and Schemas.

However, for a given annotation campaign, we will always rely on **a specific instantiation** of this meta-model, not directly on this generic level. A specific instantiation in Glozz is simply called an « **annotation model** ».

Such instantiation of the U-R-S meta-model is merely, for each of this meta-categories (U, R and S), the list of different types we can use to annotate. In other words, when we'll have to annotate something as a Unit, we will never just say this thing is a Unit, but we will choose among different types of Units provided by the specific instantiation of the model we are using for this campaign.

Let's see an example, which will be practically studied in section 6. Assume we need an annotation model for co-reference annotating. We could define :

- two kinds of units, *noun* and *pronoun*,
- two kinds of relations, *part-of* and *relationship*,
- one kind of schema, *reference-chain*.

With this annotation model, we can annotate all nouns and pronouns of a text with the two kinds of units. Then create as many reference chain as needed, each of them containing some nouns and pronouns units. Then draw relations between some reference-chains schemas.

Of course, this is a very simple example. Some campaigns use a lot of types for each of the 3 meta-types, some others use a few, or even use units and relations only, or units and schemas only, etc. This deeply depends on the annotation task, and the way we modelize it.

As you can see, Glozz is not devoted to any special paradigm. It can be configured to any task by defining a relevant annotation model.

It is also possible to mix different models within a given campaign, for instance syntax and semantics, either defining a whole annotation model containing syntactic and semantic categories, or defining two annotation models, one for syntax, the other for semantics, which would be used respectively at a first stage and at a second stage of the annotation process.

### 1.3. Feature-Sets

Each annotation element can be associated a feature-set, i.e. a set of couples (feature-name, feature-value), which provides additional **individual information**.

For an individual element, the set of the features it will embed are defined by the type of element it belongs to. But for each of these features, this individual element has its own value. So, In addition to what we've just seen in last sub-section, an annotation model defines, for each of its types, a feature-set model to be filled by its elements.

For instance, if we consider again our example of annotation model regarding co-reference, we could define for the *noun* Unit type (and the same for *pronoun*) the feature set model as follows :

Feature name	Possible values
Gender	<i>Male, Female, Neuter</i>
Count	<i>Singular, Plural</i>

Suppose now we annotate « John » as a *noun* in a text, we will fill its feature set as follows :

Feature name	value
Gender	<i>Male</i>
Count	<i>Singular</i>

And suppose we annotate « the cars », the feature set will be :

Feature name	value
Gender	<i>Neuter</i>
Count	<i>Plural</i>

Even if our examples concern units, feature-sets can be used exactly in the same way for Relations and Schemas.

Note that at the time this manual is written (Glozz version. 1.0), feature-sets are not recursive (a value can't be itself a feature set), but this may be added in the future.

## **1.4. Annotation IDs in Glozz : « real IDs » versus « friendly IDs »**

### **1.4.1 Real IDs**

Each annotation in Glozz has its own ID which identifies it from any other one in the world. This is very important so that no conflict may appear when, for instance, merging several resources.

To do so, Glozz defines an ID by concatenating the annotator's login and the exact date of creation of this annotation. Since each annotator in the world has a unique login (thanks to the login attribution procedure), and the exact date is expressed in millisecond, there is no risk two annotations will be given the same ID. This date is number of milliseconds elapsed since 1970 January the first, as it is used namely in Java language.

Here is an example of a real ID in Glozz (with annotator's login ymathet) :  
ymathet\_1290167040405

### **1.4.2 Friendly IDs**

However, these real IDs are not user-friendly, being far too long. Hence, Glozz uses « friendly IDs », which is a parallel system of ID simply being numbers from 1 to the number of annotations of the current annotated text. Hence, the first annotation of the text has the friendly ID 1, the second has the friendly ID 2, etc.

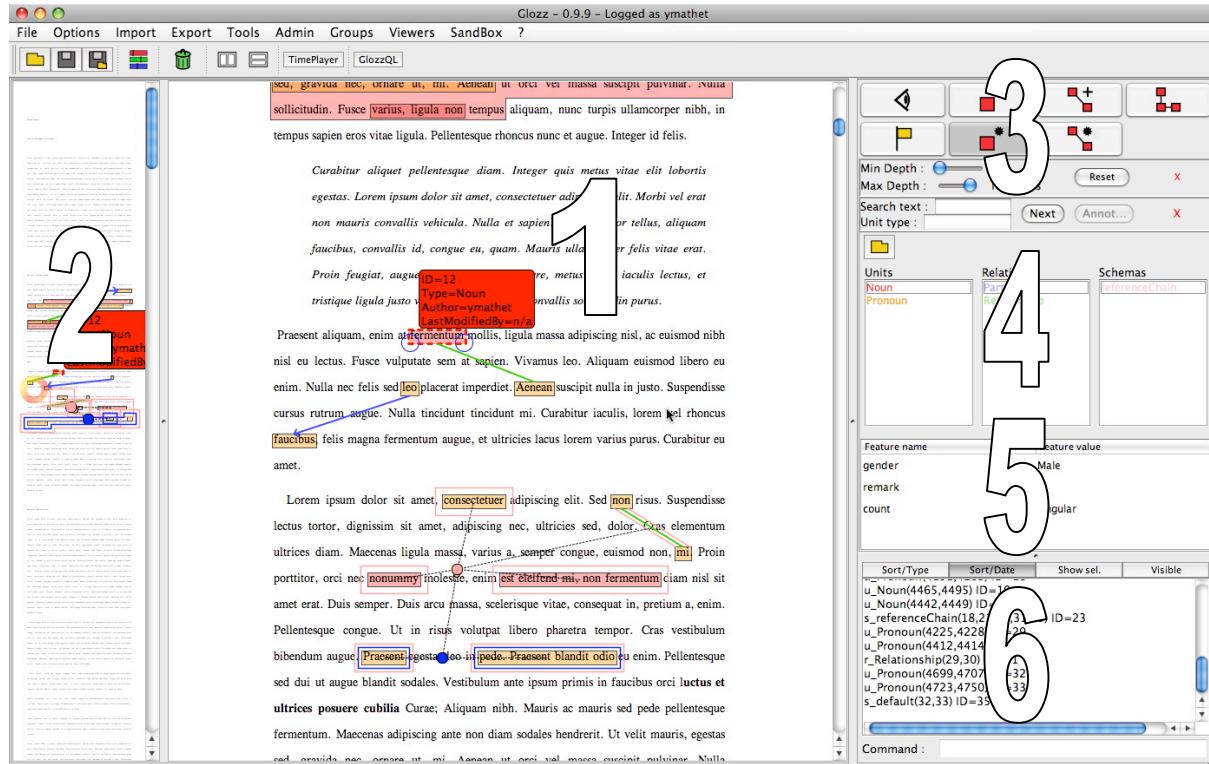
This is much more easy to handle, but, be careful, this makes sense only within a Glozz session, and there is absolutely no warranty that an annotation which is given, for example, the friendly ID number 312 will have the same number the next time. Consequently, you should never communicate to other people nor store for yourself friendly IDs, but only real IDs. Of course, this is what is done when saving your annotation in a file with Glozz. Even if you've used « friendly IDs » when working, the real ones will be stored.

Please refer to the User Interface chapter, **section 2.3.4** to see how to choose between real and friendly IDs when working.

## 2. User interface

### 2.1. overview

The main interface comes with 6 main frames:



- 1 : **main view**, where we can see the annotated text, and directly add or edit annotations.
- 2 : **macro view**, it's a view on the same annotated text as the main view does, text but in macro mode, enabling to have a global view on the annotated text, and to navigate quickly through it.
- 3 : **mode** buttons, in order to set the current mode (adding units, editing units, and so on).
- 4 : **annotation model**, where we can see the list of all available types (one column for units, one for relations, and one for schemas).
- 5 : **feature sets** table, which shows the features values of the selected element.
- 6 : **annotation as text** table, where each element is shown in a logical predicate.

Let's see in details each of these frames (and how to use them) in the next sub-sections.

### 2.2. Main and macro views: viewing and navigating

This point concerns frames 1 and 2.

These two views are representing the current text, with current annotations. They only differ on two aspects :

- the view number 2 is a macro view of the document, i.e. it is devoted to show the annotations at a large scope, not to read the text.

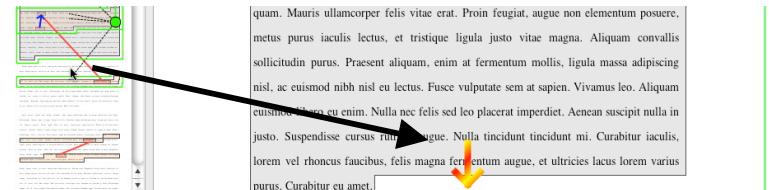
- the view number 1 shows the text in a readable font-size, and enables the user to **create or edit annotations**.

The point is that these two views are linked together, so that it is possible to navigate easily in the text using the macro view (frame 2), and then watch and work the annotated text using the main view (frame 1). To do so, click anywhere in the frame 2, and immediately the frame 1 will be positionned at this exact point, and conversely.

Moreover, an option makes it possible to show the correlated positions between frame 1 and frame 2. To activate or desactive it, use the *option* menu, and the *viewer* tab :



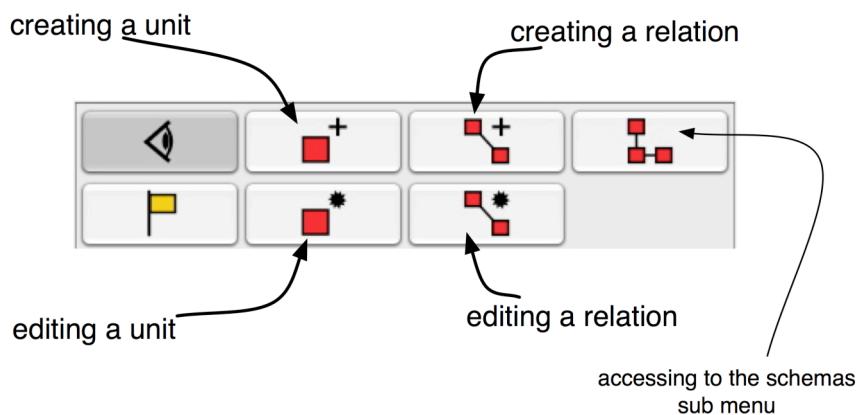
Then, when the mouse points somewhere in the frame 2, the frame 1 shows a cursor corresponding to the same position if it is within the current zone, or either an arrow pointing to the top if the position is *supra*, or to the bottom, like in the next screenshot, if the position is *infra* :



### 2.3. Annotating how-to: adding and editing annotations

This point concerns frames 1, 3, 4 and optionnally 5.

A toolbar is prodived in frame 3 as follows, which enables to choose the current mode of annotation :



Choosing a given mode will result in a specific behavior of the the frame 1, as it is detailed in the next sections.

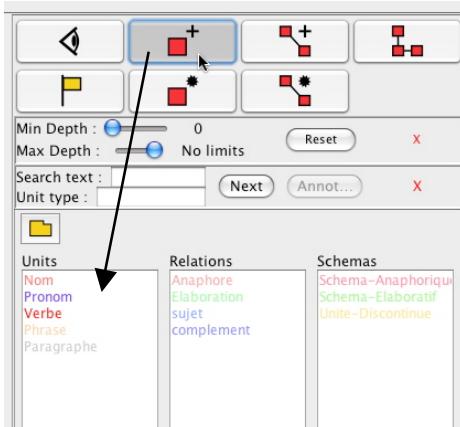
**Important** : before creating or editing annotations, you may **load an annotation model** so that you can operate with the types deditacted to your annotation task. Refer to section 1.2 to see what is an annotation model. To load one, click on the yellow button of the frame 4 :



### 2.3.1 Units

#### Creating a unit

First, click on the unit creation mode button. This results in highlighting this button (showing that this mode is now active), but also in activating the Units part of the annotation model selector (frame 4) in the left :



Then, select in the Units model the type you want to give to the next created unit(s), for instance verb in our example below :



Once in unit creation mode, you can add new units directly with the mouse, with two fashions :

- drag&drop : put the mouse on the position of the start position of the future unit, then press the button and keep it pressed while moving the mouse to the end position. When the mouse is released, the unit is fully created.
- Two clicks : click on the start position. Then a « begin » flag is shown. Then move the mouse towards the end position. While moving, a « end » flag is following the mouse position. Then, once on the end position, click again.



While creating a unit or just after, you may want to **cancel** what you're doing (or what you've just done). To do so, press the « backspace » button of your keyboard.

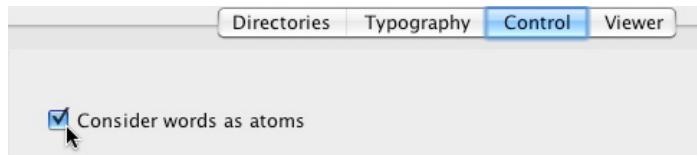
### Attributing values to feature sets

Once the unit is created, it appears as a colored box (red in the example below), which color is given by the stylesheet, according to the chosen type in the annotation model. As soon as it is created, its feature-set is shown in frame 5, and can be edited immediately. To do so, click on the value you want to edit in the « feature value » column. Then the corresponding row is highlighted (« groupe » feature below), and the value can be re-set :

Feature name	Feature value
temps	present
groupe	Premier

### Choosing between character or word atoms option

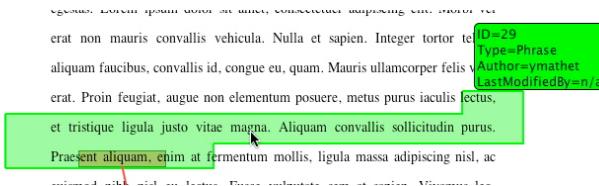
In some cases, you may want to work at a character level (for instance if some of your units may begin or end within a word), whereas in other cases, you may prefer to work at a word level. You can choose between two corresponding modes, in the options, as shown below :



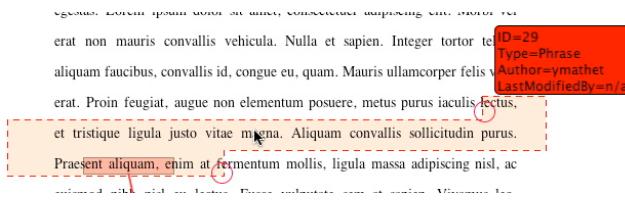
With « words as atoms » selected, each click on a word will result on positionning a frontier just before (for a begin frontier) or just after (for a end frontier) the word. Moreover, if you double click on a word, this will create a unit just surrounding this latter.

### Selecting & Editing units

Click on the dedicated button in the mode toolbar, as shown at the beginning of section 2.3. Then, when moving the mouse over a unit with the mouse will pre-select it, this latter appearing in green color, and comes with an additional information panel showing its ID, its type, its author, and the last author if it has been modified :



Then, if you click, it will really select this unit, resulting in a red and dotted frontier :



As long as this unit is selected, you can move its begin point and end point, by drag&drop. To do so, put the mouse above the green circle drawn at the begin or end position, press the mouse, move it to the new position, and release. In the example below, we drag&drop the end position of the unit (it is the same procedure for the begin one) :



Besides, you can edit the feature set associated to this unit, just clicking on the values. Indeed, as soon as a unit is selected, the feature set controller (frame 5) is automatically set to it (showing its features, and enabling edition).

### Deleting a Unit

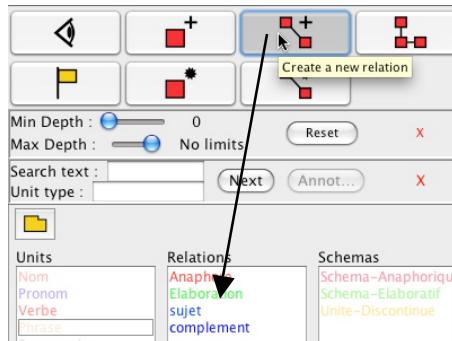
To delete a Unit, select it, then you can either press the « Delete » button (or « CTRL-Backspace » on a Mac) of your keyboard, or click on the trash icon on the top of the screen.

## 2.3.2 Relations

The main procedure is very close to the one concerning units, just shown in previous section. Hence, we will show here the specific points only.

### Creating a Relation

First, click on the « create a new relation » button mode, which activates this mode and also the related part of the annotation model selector :



Then, the procedure is very simple :

- reminder : a relation is always a link between two annotation elements (and cannot point on a part of text not having annotations), these annotations being units, relations or schemas.
- put the mouse over the start element (it becomes red)

- click
- put the mouse over the end element (it becomes red)
- click

Here is an example where start and end elements are units, but would be the same with relations or schemas.

s a. <b>semper</b> congue, euismod non, m est eleifend mi, non fermentum diam elerisque vitae, <b>consequat</b> in, pretium : t libero pharetra tempor. Cras ves e. <b>Praesent blandit</b> odio eu enim. Pelle a ante ipsum primis in faucibus orci l  going to the start	s a. <b>semper</b> congue, euismod non, m est eleifend mi, non fermentum diam elerisque vitae, <b>consequat</b> in, pretium : t libero pharetra tempor. Cras ves e. <b>Praesent blandit</b> odio eu enim. Pelle a ante ipsum primis in faucibus orci l  clicking the start	s a. <b>semper</b> congue, euismod non, m est eleifend mi, non fermentum diam elerisque vitae, <b>consequat</b> in, pretium : t libero pharetra tempor. Cras ves e. <b>Praesent blandit</b> odio eu enim. Pelle a ante ipsum primis in faucibus orci l  moving to the end	s a. <b>semper</b> congue, euismod non, m est eleifend mi, non fermentum diam elerisque vitae, <b>consequat</b> in, pretium : t libero pharetra tempor. Cras ves e. <b>Praesent blandit</b> odio eu enim. Pelle a ante ipsum primis in faucibus orci l  clicking the end
---	---	--	---

Note : For an element being itself a **relation**, you can put the mouse over any point of its line, but for an element being a **schema**, you have to put the mouse over the main circle of this latter.

### Selecting & Editing relations

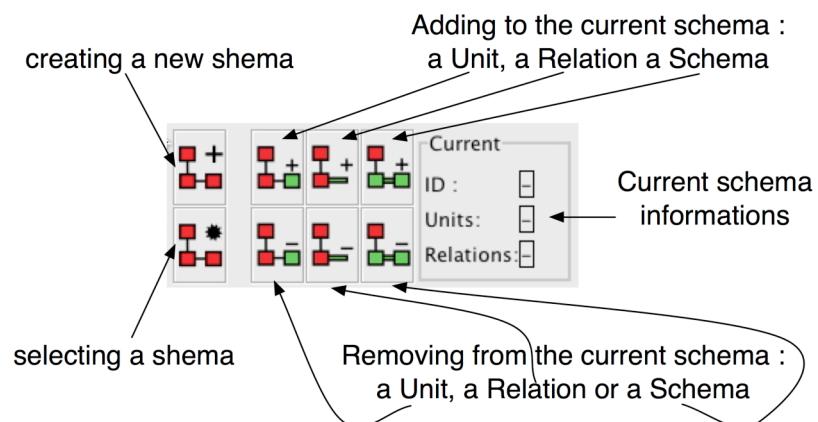
It is the exact same procedure as for Units. Please refer to previous section.

### Deleting a relation

It is the exact same procedure as for Units. Please refer to previous section.

### 2.3.3 Schemas

Schemas being richer structures than units and relations, they require a dedicated toolbar. To make this bar appearing, click on the special button as shown at the beginning of section 2.3. Then, the special menu will appear as follows :

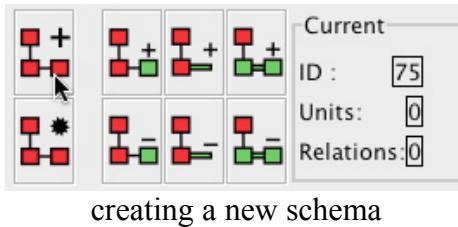


On the right, the panel named « current » shows some informations about the currently edited schema :

- its ID (a number which identifies it)
- Units : the number of units contained in this schema
- Relations : the number of relations contained in this schema

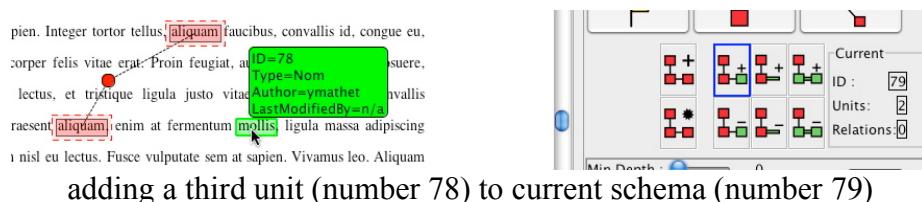
## Creating a new schema

To create a new schema, click on the dedicated button in the top left corner. Immediately, a new schema is created in Glozz, and its ID appears in the « current » panel. It is, at this stage, an empty schema, since it behaves no element at all. In the example below, a new schema is created with ID number 75, and, of course, 0 Unit and 0 Relation :



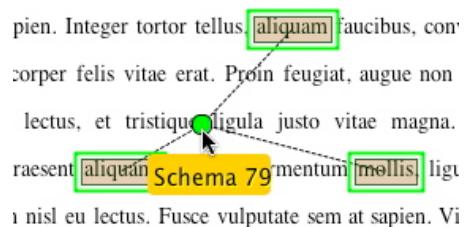
Remark: If you happen to click again on the creation button before you add any element to the new one, Glozz won't create a new one, and you will still stay on the one just created and still empty.

Then you can add as many Units, Relations or Schemas to the current schema, using the three dedicated buttons (icons with « + » character). For instance, to add a unit, once you've clicked on the « adding a unit » button, move the mouse over the unit you want to add in frame 1, and then click on it. As long as you stay in this mode, you can add as many units as you want by clicking them.



## Editing a schema

To edit a schema, at first click on the « selecting a schema » button. Then, in the frame 1, click on the schema you want to select (it appears in green color when the mouse is over, then in red once selected by clicking). Remark : you have to click on the circle of the schema to select it, not on one of its components. Its informations (ID, number of units and relations) appear in the right panel :



Use the 3 buttons (with a « + » icon) to add units, relations or schemas, and the 3 buttons (with a « - » icon) to remove units relations or schemas.

### Deleting a schema

Select it, then click on the garbage icon, or press DEL (or Function-Backspace on a Mac).

#### 2.3.4 Working with « real IDs » or « friendly IDs »

Please refer to **section 1.4** before reading this.

As we've seen in the previous chapter, each annotation is given an ID. Its value appears, as we've seen, when it is selected.

For instance, in this screenshot, the selected annotation is shown with ID=26. It is its friendly ID, in the frame 1, but also in the frame 6 :

The screenshot shows a text editor with some Latin text. An annotation in the first frame is highlighted with a red box and has a tooltip showing its properties: Type=Verbe, Author=ymathet, LastModifiedBy=n/a, and ID=26. A second annotation in the same frame has a tooltip showing ID=30. A third annotation in the same frame has a tooltip showing ID=30. In the sidebar, there is a table titled 'Sort/Type' with several rows of annotations, one of which has a red box around it and is labeled 'ID=26'. A black arrow points from the 'ID=26' in the table to the 'ID=26' in the tooltip of the first annotation.

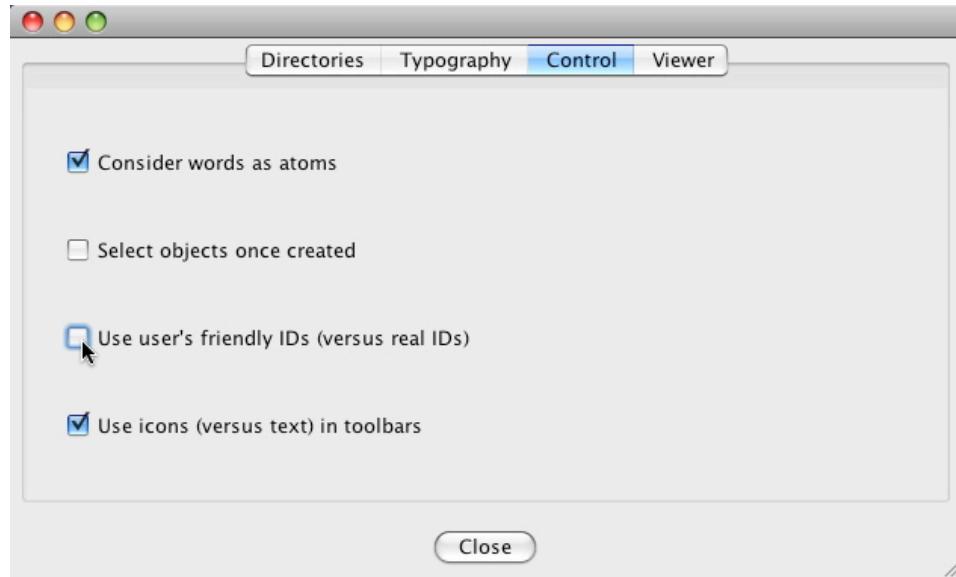
Sort/Type	Sort/Date
u_verbe(4552,4544)	ID=20
u_Verbe(4831,4837)	ID=21
u_Verbe(7267,7276)	ID=22
u_Verbe(7606,7617)	ID=23
u_Verbe(6461,6469)	ID=24
u_Verbe(9888,9902)	ID=25
u_Verbe(10191,10201)	ID=26
u_Verbe(8709,8727)	ID=27
u_Phrase(8619,8737)	ID=28
u_Phrase(9799,9910)	ID=29
u_Phrase(10178,10337)	ID=30
u_Phrase(41_102)	ID=31

In the next screenshot, the same annotation is now shown with its real ID :

The screenshot shows the same text editor and sidebar table as the previous one. However, the annotation in the first frame now has a tooltip showing its real ID: ID=ymathet\_124655791464. A black arrow points from this real ID in the tooltip to the real ID in the table on the right. The table now includes a 'Show sel.' column, and the row for the selected annotation has a red box around it and is labeled 'ID=ymathet\_1246557914641'.

Sort/Type	Sort/Date	Show sel.
u_verbe(4552,4544)	ID=ymathet_124655791453	
u_Verbe(4831,4837)	ID=ymathet_1246557906501	
u_Verbe(7267,7276)	ID=ymathet_1246557908371	
u_Verbe(7606,7617)	ID=ymathet_1246557909522	
u_Verbe(6461,6469)	ID=ymathet_1246557911009	
u_Verbe(9888,9902)	ID=ymathet_1246557913763	
u_Verbe(10191,10201)	ID=ymathet_1246557914641	
u_Verbe(8709,8727)	ID=ymathet_1246557916100	
u_Phrase(8619,8737)	ID=ymathet_1246557922026	
u_Phrase(9799,9910)	ID=ymathet_1246557925098	
u_Phrase(10178,10337)	ID=ymathet_1246557928837	

This is an option to choose, at any moment, as often as necessary, via the option panel, by checking or unchecking the dedicated checkbox :



## 2.4. Annotations as predicates (Frame 6)

### 2.4.1 Overview

This tool is another way of viewing and creating annotations, and works simultaneously to others views of the application.

It shows each annotation as a predicate, with its name and arguments, and so, consists in a complete list of the current loaded annotations.

Let's see an overview of this module in the following screenshot.

Sort/Type	Sort/Date	Show sel.	Visible
u_Phrase(4438,4950) ID=38			
u_Phrase(7239,7650) ID=39			
u_Paragraphe(4163,6213) ID=40			
u_Paragraphe(2113,4163) ID=41			
u_Paragraphe(6237,8286) ID=42			
r_Anaphore(4,6) ID=45			
r_Anaphore(15,14) ID=46			
r_Anaphore(22,27) ID=47			
r_Anaphore(25,26) ID=48			
r_sujet(1,3) ID=49			
r_sujet(8,9) ID=50			
Command : <input type="text"/>			

Remark : if there are too many annotations to list, a vertical scrollbar appears in the right of the window.

**Units** : for instance, the first visible line ***u\_Phrase(4438,4950) ID=38*** is related to the annotation whose ID is **38**, which is a Unit since it starts with ***u\_***, whose category is ***Phrase***, which starts at character ***4438***, and ends at character ***4950***.

**Relations** : The last visible line shows the relation(since it starts with ***r\_***) whose ID is **50**, whose category is ***sujet***, and which links annotation of ID **8** to annotation of ID **9**.

**Schemas** : A schema predicate starts with `s_`, and its arguments are the ID of the nested annotations of the related schema.

## 2.4.2 Navigation and selection

This tool is reactive : when you put the mouse over any part of predicate (no need to click), and shows immediately, in the Frames 1 and 2, where the related annotation is. In the example below, it is shown that the Phrase unit of ID 38 is located below in the text of frame 1.

Sort/Type	Sort/Date
u_Phrase(4438,4950) ID=38	
u_Phrase(7239,7650) ID=39	
u_Paragraphe(4163,6213) ID=40	
u_Paragraphe(2113,4163) ID=41	
u_Paragraphe(6237,8286) ID=42	
r_Anaphore(4,6) ID=45	
r_Anaphore(15,14) ID=46	
r_Anaphore(22,27) ID=47	
r_Anaphore(25,26) ID=48	
r_sujet(1,3) ID=49	
r_sujet(8,9) ID=50	

Then, if you click, the related annotation will be selected, and shown in the frames 1 and 2 :

Units	Relations
Nom	Anaphore
Pronom	Elaboration
Verbe	sujet
Phrase	complement
Paragraphe	

The same can be done with the arguments of a predicate :

```
| r_Anaphore(4,6) ID=45
| r_Anaphore(15,14) ID=46
| r_Anaphore(22,27) ID=47
```

Reversely, if an annotation is currently selected in Glozz (whatever the way), and you want to have it shown in the list, click on « Show Sel. » button :

Sort/Type	Sort/Date	Show sel.	Visible
r_Anaphore(22,27) ID=47			
r_Anaphore(25,26) ID=48			
r_sujet(1,3) ID=49			
r_sujet(8,9) ID=50			
r_sujet(20,21) ID=51			
r_complement(19,18) ID=52			
r_complement(23,22) ID=53			
r_Elaboration(34,11) ID=54			
r_Elaboration(17,35) ID=55			
u_Nom(1402,1412) ID=43			
u_Nom(1508,1515) ID=44			
s_Unite_Discontinue(43,44) ID=57			
current selected annotation id 38 not visible in the list			
list is scrolled so that current selected annotation id 38 is shown			

#### 2.4.3 Sorting the list

The list is sorted either chronologically (from the oldest annotation to the youngest), or by type (Units, then Relations, the Schemas). Use the two buttons is the top to do so.

#### 2.4.4 Managing visibility for each individual annotation

This tool can also be used to hide or show the annotations individually (contrary to the use of styles or groups, which concern a set of annotations). To hide an annotation in the frames 1 and 2, double click on it int the list. Then, it is hidden in the other views, and appears hatched in the list :

Sort/Type	Sort/Date	Show sel.	Visible
r_sujet(8,9) ID=50			
r_sujet(20,21) ID=51			
r_complement(19,18) ID=52			
r_complement(23,22) ID=53			
r_Elaboration(34,11) ID=54			
r_Elaboration(17,35) ID=55			
r_Elaboration(61,60) ID=56			
s_Unite_Discontinue(43,44) ID=57			
s_Unite_Discontinue(36,37,19) ID=58			
s_Schema-Anaphorique(33,34,35,8,15,57) ID=59			
s_Schema-Elaboratif(39,42,28,24) ID=60			

To have an annotation back to visible, double click again on it in the list.

If you want to have all hidden annotation visible again, you can click on the « Visible » button (instead of clicking individually on each of them).

#### 2.4.5 Command line feature: creating annotations via a predicate entered with the keyboard

It is possible to create annotations by typing a predicate directly with the keyboard. To activate this feature, click in the command line, and it appears in green background color :

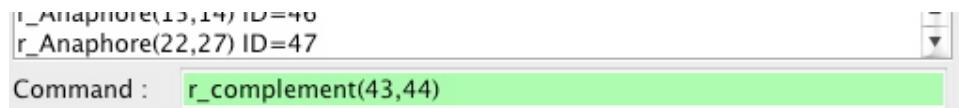
r_Anaphore(13,14) ID=40
r_Anaphore(22,27) ID=47
Command : <input type="text"/>

Then, you just have to write the predicate in the very exact manner as it is in the list. This feature provides auto-completion, and moreover, works with auto-completion only, so that it is impossible to write a wrong predicate : at any moment, if you type a character which is not compatible with a possible predicate, the character is not entered ; on the contrary, if it is a way to complete the predicate being built, it will be taken into account, and automatically completed with additional characters if there is no other possibility.

Let's take an example. Assume we want to create a relation of type « complement » between annotation 43 and 44. The associated predicate will be « r\_complement(43,44) ». In fact, what we have to type on the keyboard is only : « rc4344 ». Indeed, here is what we type and what we get via auto-completion :

Keyboard	Displayed command	Explanation
r	r_	at this stage, only <i>u</i> , <i>r</i> and <i>s</i> were possible
c	<i>r_complement</i> (	among current relation types, only <i>complement</i> begins with <i>c</i> character
4	<i>r_complement</i> (4	Several annotation IDs begin with 4 : 4, and 41 to 49
3	<i>r_complement</i> (43,	No other ID than 43 begins with 43, hence the comma
4	<i>r_complement</i> (43,4	Same remark
4	<i>r_complement</i> (43,44)	Same remark, hence the predicate completed

The result is as follows:



Important : the auto-completion relies on the current loaded annotation model to complete type names, and on the current available annotation to complete the IDs.

Once the predicate complete, press **ENTER** to have the related annotation created.

At any moment, press **BACKSPACE** to cancel the effect of the last typed character.

### 3. Installing and getting started how-to

#### 3.1. Download and unpack Glozz

Download the latest version of Glozz from the website <http://www.glozz.org>, as .tgz archive. Unpack this .tgz, and you get a folder containing the distribution :



#### 3.2. Launch Glozz

The distribution contains the application as « **glozz-platform.jar** » file. It is a Java program which can be launched directly by double-clicking it. If it doesn't, you probably haven't Java correctly installed or configured on your machine. Please use Google with the words « Java download » and the name of your system to get Java installed.

If you're running windows, you can take advantage of launching Glozz via « **StartGlozz.bat** » file. It launches the .jar application file with more memory which is better when dealing with big files.

#### 3.3. Create a shortcut on your desktop

You may want to have Glozz appearing directly on your desktop. Take care **not to move** the .jar nor the .bat files from the distribution folder. You should rather **make a shortcut** (right-click and choose « make a shortcut ») of one of these files, and then **move this shortcut** to your desktop. This is important because Glozz has to be launched from its original folder in order to have access to its data files.

#### 3.4. Choose a login and ask for a key

At this step, you can play with Glozz to test it (you can log as anonymous), but you won't be able to save your annotations.

To really and fully use Glozz, you need to be logged, which means having a login and the associated key. Indeed, in order to guarantee that each annotation is systematically assigned to a unique annotator (worldwide), each user must be authentified with a unique login before saving his data.

To get it, please send a mail to the authors (use the link of the website) with the login you want to use (it is advised to use the first letter of your first name followed by your last name). If this login is free, you'll receive the associated key. Otherwise, you will be proposed another login and its associated key.

Then, the next time you launch Glozz, you can log in, and your system will remember it for next sessions.

### ***3.5. Creating your own data folder(s)***

Please read next section to understand what kind of data is involved in Glozz, and how to create your associated folders.

## 4. File types overview

Glozz uses 5 file types:

Two of them are dedicated to store corpus. Each corpus is stored via a pair of files, one .ac and one .aa.

.**ac** is a text file containing all the characters of the corpus including space characters and punctuations, but no line feed. Consequently, it appears on one (very long) single line, with no typography, and should never be modified since other files will rely on it.

.**aa** is an xml annotation file, constructed and updated relying on a given .ac file. It contains all annotation marks, including typographic ones (titles, paragraphs, lists, etc.), and of course manual annotations which will be made. Of course, a .aa file must be used only with its associated .ac file, but glozz will prevent you from doing any mistake since each .aa remembers which its associated .ac is (through a hashcode).

Note that for a given .ac, you may create as many .aa as you wish. It will be the case in particular when a corpus is annotated by several annotators, each one of them working on his own .aa.

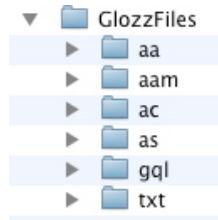
.**aam** is an xml file describing an instance of what will be called « annotation model ». It's where all types of entities will be available for a given annotation campaign. One .aam file may be used for several corpus.

.**aas** is an xml file describing a stylesheet for a given annotation model. You can define and use as many .aas as you wish for a given .aam file, providing you different views on the same corpus. For instance, if the .aam of a campaign copes with syntax and semantics, you can create one .aas file showing syntax only, and a second one showing semantics only, in order to focus on one phenomenon at a time when annotating or observing annotations.

.**gql** is an xml file storing queries expressed through the dedicated « Glozz Querry Language » GlozzQL. You can create as many gql files as you wish for a given campaign, or even more generic queries applying on any entities whatever the .aam.

Besides, Glozz can generate automatically corpus in Glozz format (.ac + .aa) from a .txt file, so, you may store in a specific folder your .txt files too.

Since Glozz memorises the folders where you save and load each type of file, we strongly recommand that, at least for a given campaing, you create a dedicated folder organized as follows :



## 5. Creating a corpus in Glozz format.

### 5.1. Manual corpus creation from a txt file via « import txt » menu

Assume we want to pick an article from Wikipedia to work with in Glozz. Here, the biography of the painter Henri de Toulouse-Lautrec :

Henri de Toulouse-Lautrec  
From Wikipedia, the free encyclopedia

Henri Marie Raymond de Toulouse-Lautrec-Monfa or simply Henri de Toulouse-Lautrec (French pronunciation: [ɑ̃lɪ də tuluz lɔ̃tʁɛk]) (24 November 1864 – 9 September 1901) was a French painter, printmaker, draughtsman and illustrator, known for his colorful and theatrical life of fin de siècle Paris yielded an array of exciting, elegant and provocative images of the modern and sometimes decadent life of those times. Toulouse-Lautrec is known along with Cézanne, Van Gogh, and Gauguin as one of the greatest painters of the Post-Impressionist period. In a 2008 auction at Christie's auction house a new record was set when "Le blanchisseuse", an early painting of a young laundress, sold for \$22.4 million U.S.<sup>[1]</sup>

**Contents** [edit]  
1 Biography  
1.1 Youth  
1.2 Disability and health problems  
1.3 Paris  
1.4 London  
1.5 Alcoholism  
1.6 Death  
2 Art  
3 Selected works  
3.1 Signature  
4 Movies  
5 References  
6 External links

**Biography**

**Youth**

Henri Marie Raymond de Toulouse-Lautrec-Monfa was born in Albi, Tarn in the Midi-Pyrénées region of France, the firstborn child of Comte Alphonse and Comtesse Adèle de Toulouse-Lautrec. He was therefore a member of an aristocratic family (descendants of the Counts of Toulouse and Lautrec and the Viscounts of Monfa, a village and commune of the Tarn department of southern France). A younger brother was also born to the family on 28 August 1867, but died the following year.

After the death of his brother his parents separated and a nanny took care of Henri through this time.<sup>[2]</sup> At the age of 8, Henri left to live with his mother in Paris. Here he started to draw his first sketches and caricatures on his exercise workbooks. The family quickly came to realize that Henri's talent lay with drawing and painting, and a friend of his father named Dr. Georges-Antoine Baune sometimes gave informal lessons. Some of Henri's early paintings are of horses, a specialty of Princetown, and something that he would later visit with his "Circus Paintings".<sup>[2][3]</sup>

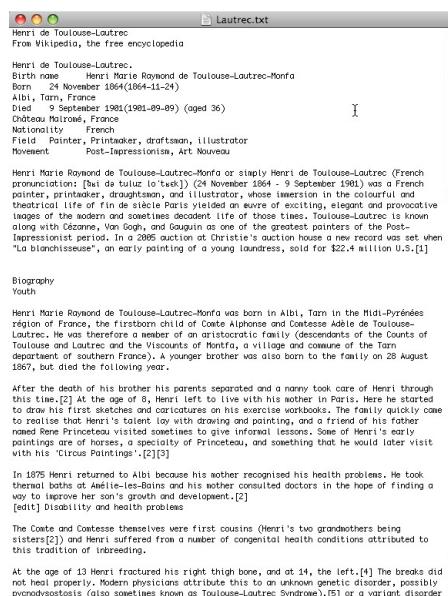
In 1875 Henri returned to Albi because his mother recognized his health problems. He took thermal baths at Aude-les-Bains and his mother consulted doctors in the hope of finding a way to improve his son's growth and development.<sup>[2]</sup>

**[edit] Disability and health problems**

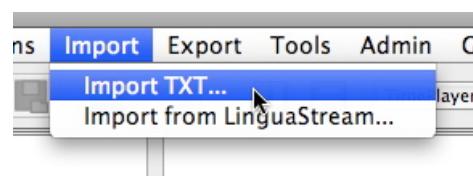
The Comte and Comtesse themselves were first cousins (Henri's two grandmothers being sisters<sup>[2]</sup>) and Henri suffered from a number of congenital health conditions attributed to this tradition of inbreeding.

At the age of 13 Henri fractured his right thigh bone, and at 14, the left.<sup>[4]</sup> The breaks did not heal properly. Modern physicians attribute this to an unknown genetic disorder, possibly pycnodysostosis (also sometimes known as Toulouse-Lautrec Syndrome),<sup>[5]</sup> or a variant disorder

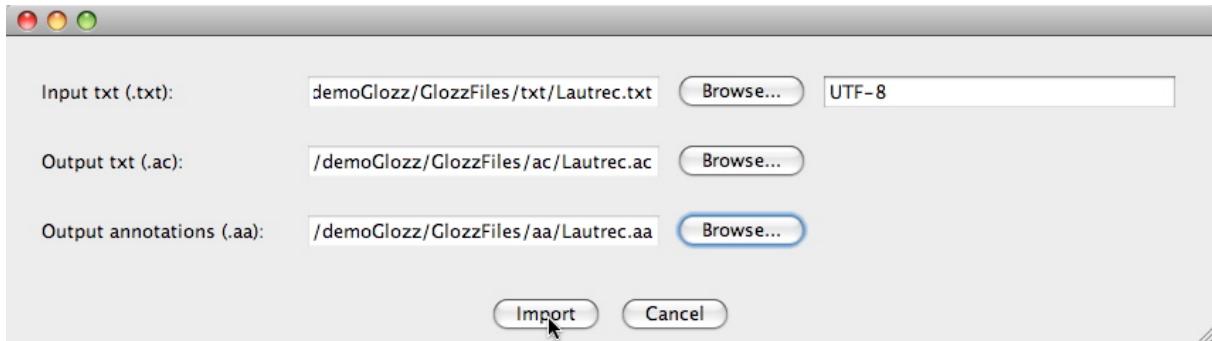
First, we copy the content, and paste it in a text editor. Then, after some cleaning if necessary, we save the file as .txt (Lautrec.txt), encoded in UTF-8.



Now, we can launch Glozz, an use the « import txt » facility :



and we have to fill-in 3 paths, and the encoding of the .txt :

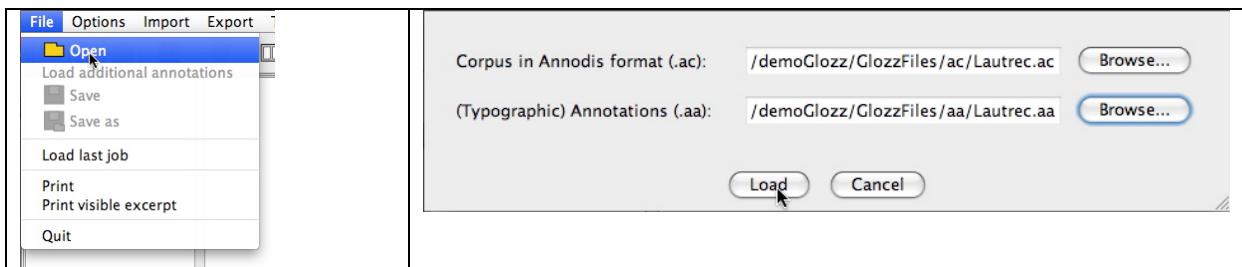


To do so, for each field, we can use the « Browse » button instead of writing the paths manually. Since it's the first time we store or load this type of data in Glozz, we have to browse through our folders in order to go to the correct places as shown in fig. 1. Next times, these folders will be proposed as soon as you'll click on the « Browse » buttons.

So, in our example, we choose the input text Lautrec.txt we've just created. Since it was generated in UTF-8, we leave the default UTF-8 value as it is in the next field.

For the outputs, we browse till we're respectively in the .../GlozzFiles /ac/ and .../GlozzFiles/aa/ folders, and then complete the names via the keyboard with *Lautrec.ac* and *Lautrec.aa*. Note that Glozz will complete automatically the file names with the correct suffixes if you write only *Lautrec* instead of *Lautrec.ac* and *Lautrec.aa*.

Once done, you can open this new corpus via the File menu or the shortcut button :

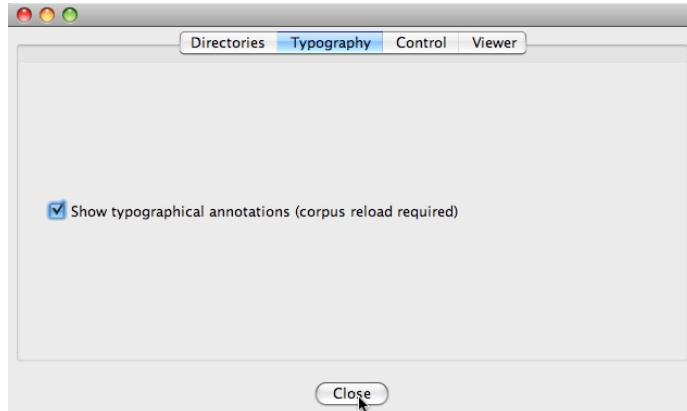


What we get is the text organized in simple paragraphs :

We can go further and use some more styling for this corpus.

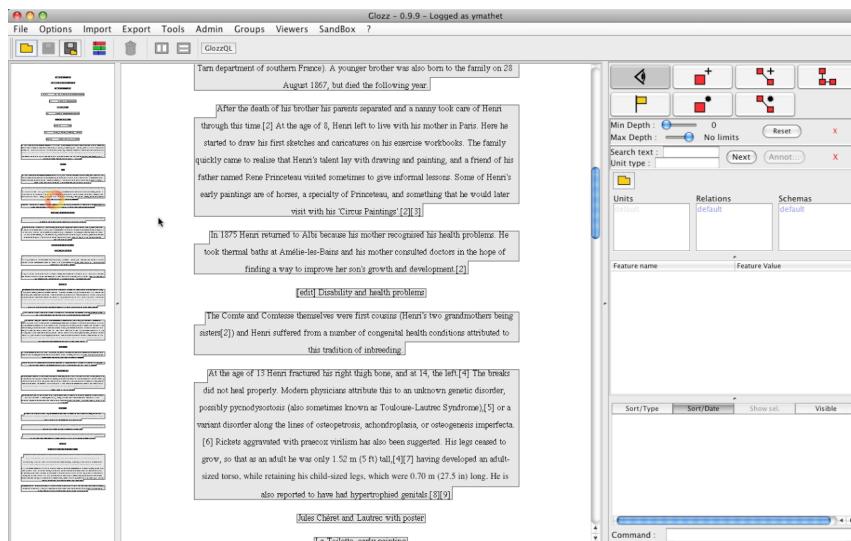
To do so, we first have to (temporary) choose which will show typographical annotations. Indeed, at present time, paragraphs are annotated as special units, and the Glozz renderer take this information into account to add line feeds on the screen. But, for the while, the paragraph units are hidden.

So, we go to Menu-->Options-->Preferences-->Typography :



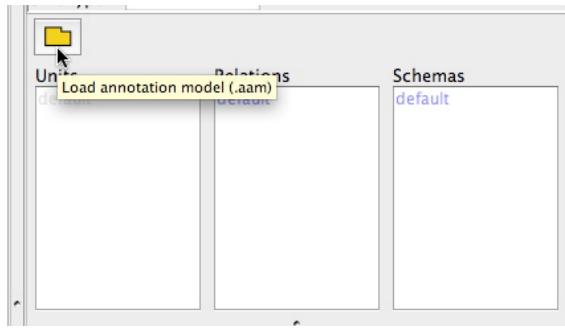
and then check the box « Show typographical annotations ».

We have then, as stated, to reload the corpus. To do so, we can now use the user-friendly « load last job » in the File menu, which will save us choosing again the same .ac and the same.aa. Then, we get each paragraph appearing with a frame around it. It means it is shown as it is really in Glozz, an annotated unit with the type « paragraph » :



Besides, we have to load the annotation model dedicated to typography.

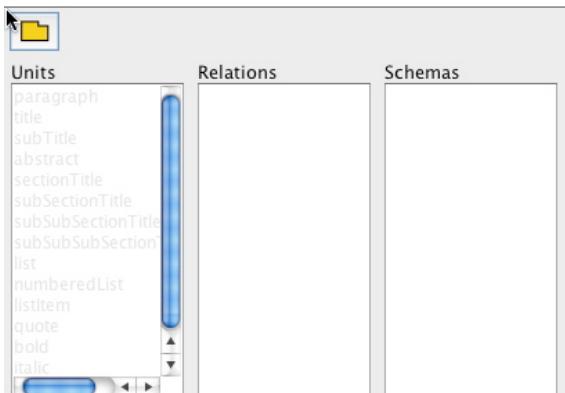
To load an annotation model .aam, we click on the yellow button just above the annotation model table :



And then browse to the special file provided in the Glozz distribution, in `.../data/annotationModels/structure-typo.aam`:

Nom	Date de modification
argumentation.aam	22 mai 2010, 11:31
lorem-ipsum-withComplexAnnotations.aam	22 mai 2010, 11:31
possibleValues.aam	22 mai 2010, 11:31
<b>structure-typo.aam</b>	<b>22 mai 2010, 11:31</b>
validation.aam	22 mai 2010, 11:31

And the unit model is now fed as follows (see « paragraph », « title », « subtitle » and so on) :



Please refer to the specific section of this manual to see how to change the style of an unit. Then, using the `structure-typo.aam` style, you will be able to change any paragraph to a title, a list item, and so on. You can also move the frontiers of a paragrah, define new paragraphs or remove some, etc.

Note that each time you want to **watch the new typographic result, you have to save and reload the current corpus**. No update will be done automatically.

The next screenshot is the result of our text with some work on typo (one main title, then two sub titles, then a list with 4 items, and then paragraphs), and once the « show typographic annotation » option is removed :



## Henri de Toulouse-Lautrec

From Wikipedia, the free encyclopedia

### Henri de Toulouse-Lautrec.

- Birth name Henri Marie Raymond de Toulouse-Lautrec-Monfa

- Born 24 November 1864(1864-11-24)

- Albi, Tarn, France

- Died 9 September 1901(1901-09-09) (aged 36)

Château Malromé, France

Nationality French

Field Painter, Printmaker, draftsman, illustrator

Movement Post-Impressionism, Art Nouveau

Henri Marie Raymond de Toulouse-Lautrec-Monfa or simply Henri de Toulouse-Lautrec  
(French pronunciation: [ɑ̃ʁi da tuluz lo'tʁɛk]) (24 November 1864 – 9 September 1901)  
was a French painter, printmaker, draughtsman, and illustrator, whose immersion in the  
colourful and theatrical life of fin de siècle Paris yielded an œuvre of exciting, elegant and  
provocative images of the modern and sometimes decadent life of those times. Toulouse-

The corpus is now created. As the corpus provider you are now, you'll have to provide to your annotators both the *Lautrec.ac* and the *Lautrec.aa* files built as we've just seen. It is very important **all your annotators start their work from these files**, and then modify *Lautrec.aa* with their own annotations. Indeed, the « import from txt » file, then cleaning, then adding or modifying typography must be done **once for all** in order that annotations done by different annotators can be compared, merged, etc.

## 5.2. Automatic corpus creation via a custom program

You may want to automatize the creation of your corpora in Glozz format. To do so, you may use some software provided by other people, or you may need to develop your own application. We provide in this section some more precise data about the Glozz .ac and .aa formats.

As we've seen in « File types » section, you will have to create two combined files. Assume we want to create a corpus named Lautrec, the two files will be *Lautrec.ac* and *Lautrec.aa*.

Here are two excerpts of these files :

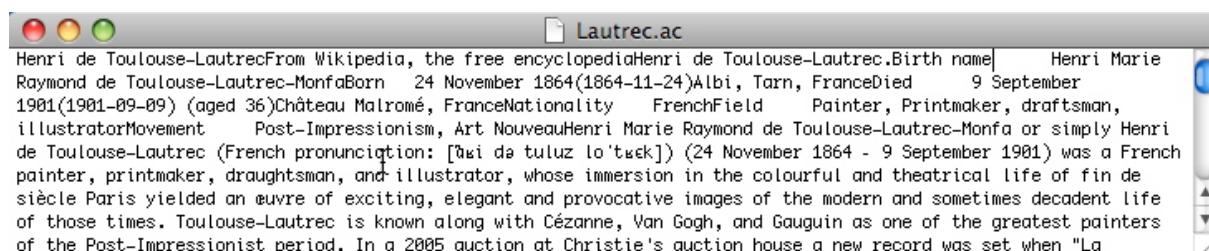


fig. 1 : an excerpt of the Lautrec.ac file

```

- <annotations>
  <metadata corpusHashCode="10680-2919708"/>
  - <unit id="TXT_IMPORTER_1290167040404">
    - <metadata>
      <author>TXT_IMPORTER</author>
      <creation-date>1290167040404</creation-date>
      <lastModifier>n/a</lastModifier>
      <lastModificationDate>0</lastModificationDate>
    </metadata>
    - <characterisation>
      <type>title</type>
      <featureSet/>
    </characterisation>
    - <positioning>
      - <start>
        <singlePosition index="0"/>
      </start>
      - <end>
        <singlePosition index="25"/>
      </end>
    </positioning>
  </unit>
  - <unit id="TXT_IMPORTER_1290167040405">
    - <metadata>
      <author>TXT_IMPORTER</author>

```

fig. 2 : an excerpt of the Lautrec.aa file

As you can see in fig.1, the .ac file should only contain text characters, including spaces and punctuations, but no line feed mark.

Let's see the main points of the .aa file in fig.2 :

- the main node is **annotations**
- it contains the metadata **corpusHashCode** which is a code enabling to identify what .ac file corresponds to this .aa file. It relies on a specific algorithm which uses the length of the text (10680 in our example) for the first number, and a modulo of the content of the text for the second number. The algorithm is provided in the appendices of this manual. If you do not whish to implement this algorithm, you can create your file with no corpusHashCode node, then open the corpus in Glozz, then save it. The hashcode will be automatically created and stored when saved.
- Then come some units nodes. The id parameter is composed of the login of the creator (*TXT\_IMPORTER* in our example, since this .aa file was created automatically by the Glozz text importer), a ‘\_’ character and the date (the number of milliseconds since 1970, given by *(new Date()).getTime()* in Java). For you own program, you should choose your own creator name (as *TXT\_IMPORTER* in our example). Last modifier nodes will be of no use till someone makes some changes in the annotation. So, at this step, we suggest you put resp. *n/a* and *0* values.
- In the characterisation node, you have to put two sub nodes : the type name, *title* in our example, which should correspond to an annotation model (*structure-typo.aam* in our example, see above, in order to format the text), and the content of the feature set. There is no feature set in our example, hence the empty node.
- Then come the positionning, which is always given, for a unit, by two singlePosition nodes. Each single position is the index of the character in the .ac file, starting at 0. In our example, there is a title unit starting at position 0 and finishing at position 25 (corresponding to « Henri de Toulouse-Lautrec »).

To be completed with feature sets, and then with relations and schemas...

## 6. Annotation Models : types, feature-sets, groups

Note : this section needs to have some (elementary) knowledge in XML. To create an annotation model, it is recommended to use a text editor with XML capabilities so that the document can be validated before testing in Glozz. It's much more simple to find an error with an XML editor/validator than just reading the xml code in a mere text editor.

As we've seen in section 1, an annotation model can be considered as an instance of the meta-model « Units – Relations – Schemas » from Widlöcher. It states, for a given annotation campaign, the kinds of Units, Relations and Schemas which will be available, and, for each of these kinds, its the feature set structure.

In Glozz, it is an XML file, with the file name extension .aam. It is strongly recommended that, besides reading this chapter, you also have a look at the different .aam files provided in the distribution.

We will assume here we want to define a (quite simple) model for an coreference annotation campaign.

### 6.1. Overview

We create a file named *coreference.aam* (and store it in our .../aam folder). The main structure is as follows, containing 3 main nodes : units, relations, and schemas.

```
<?xml version="1.0" encoding="UTF-8"?>  
<annotationModel>  
    <units>          (...)  
    </units>  
  
    <relations>        (...)  
    </relations>  
  
    <schemas>          (...)  
    </schemas>  
  
</annotationModel>
```

We'll see in details what to put in units, relations and schemas node in the next subsections.

### 6.2. Units

We would like to have two kinds of units : nouns and pronouns. Each of them will have a gender (male/female/no), a count (plural/singular/no), and a field for an additionnal remark.

Each kind of unit comes in a *type* node. The first one of our example is *Noun*. Then, we define the featureSet, which contains as many feature nodes as needed. At the moment, there are two kinds of feature types :

- with a *possibleValues* node, we define a value to be selected among several predefinite values, each of them defined in a sub node called *value*. Here, for *gender*, there are 3 possible values *Male*, *Female* and *No*. Note that we can also define a default value, here *Male*, which will be automatically set if no value is chosen by the annotator for this field.
- with a <value type="free" default="" />, we can define a free text to be entered, and, if needed, a default value. It's the case in our example for the *remark* type.

Here is the xml code :

```

<units>
    <type name="Noun">
        <featureSet>
            <feature name="gender">
                <possibleValues default="Male">
                    <value>Male</value>
                    <value>Female</value>
                    <value>No</value>
                </possibleValues>
            </feature>
            <feature name="count">
                <possibleValues default="Singular">
                    <value>Singular</value>
                    <value>Plural</value>
                    <value>No</value>
                </possibleValues>
            </feature>
            <feature name="remark">
                <value type="free" default="" />
            </feature>
        </featureSet>
    </type>

    <type name="Pronoun">
        <featureSet>
            <feature name="gender">
                <possibleValues default="Male">
                    <value>Male</value>
                    <value>Female</value>
                    <value>No</value>
                </possibleValues>
            </feature>
            <feature name="count">
                <possibleValues default="Singular">
                    <value>Singular</value>
                    <value>Plural</value>
                    <value>No</value>
                </possibleValues>
            </feature>
        </featureSet>
    </type>

</units>

```

### 6.3. Relations

Creating relation types is very similar to creating unit types. The main difference is that a relation can be oriented or not. This is set in the *oriented* attribute of the type value. An oriented relation will be shown in the interface via an arrow instead of a simple line in the case of a non oriented one. In our example, we create a *PartOf* relation which is oriented, and a *Relationship* one which is not. Hence the respective *false* and *true* values in the xml code

below. Besides, the *Relationship* relation comes with a feature set containing one feature, and the *PartOf* relation comes with no feature set.

```
<relations>

    <type name="PartOf" oriented="true">
        </type>

        <type name="Relationship" oriented="false">
            <featureSet>
                <feature name="kind">
                    <possibleValues default="Other">
                        <value>Family</value>
                        <value>Friend</value>
                        <value>Colleague</value>
                        <value>Other</value>
                    </possibleValues>
                </feature>
            </featureSet>
        </type>
    </relations>
```

## 6.4. Schemas

Once again, creating schema types is almost the same thing as creating unit types. Here, we create the *referenceChain* schema type only, with a free feature named *Remarks*.

```
<schemas>
    <type name="referenceChain">
        <featureSet>
            <feature name="Remarks">
                <value type="free" default="" />
            </feature>
        </featureSet>
    </type>
</schemas>
```

## 6.5. Groups

The notion of « group » is transversal to units, relations and schemas. The idea is to create as many groups as needed, each of them corresponding to a paradigm of the campaign, and then to set which kinds of units, relations and schemas belong to which group.

For instance, if a campaign deals both with syntax and semantics, it will be interesting to create two dedicated groups, and to set for each type to which group(s) it belongs. Indeed, a given type may belong to as many groups as needed, or to no group.

This will be very convenient, once in Glozz, to have such groups, because it will be possible to show/hide all elements whose type belong to a given group. For instance, we will be able to see only annotations dealing with syntax, then only annotations dealing with semantics, and so on.

Groups are defined in the .aam files, in an implicit manner. Indeed, there is no xml node to create a group. We only have to say, for a given type, to which group(s) it belongs, via the *groups* attribute, and the corresponding groups will be automatically created.

In our simple example, let's use two groups, named group1 and group2, as follows (we show only the modified part of the code) :

```
<units>
    <type name="Noun" groups="group1, group2"groups="group1"groups="group2"groups="group2"
```

If we want a type to belong to several groups, like *Noun* in our example, we list all the groups separated by a comma.

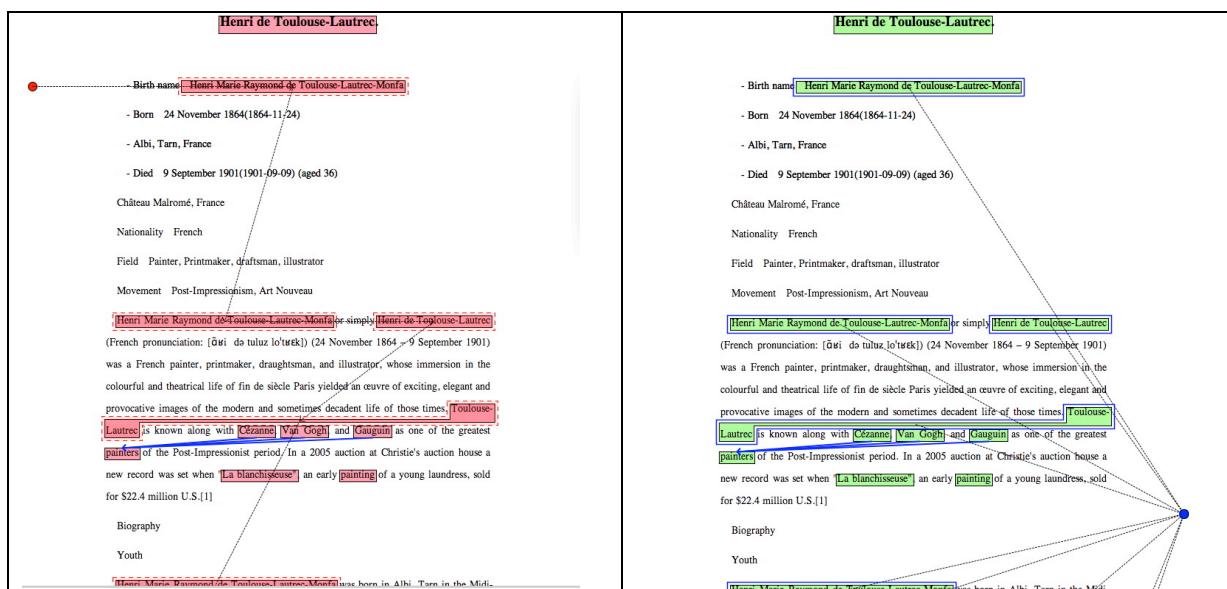
With our modified .aam file, we've set two groups, one named group1 and containing *Noun*, *Pronoun*, one named group2 and containing *Noun*, *ParfOf* and *referenceChain*.

## 7. Styles

### 7.1. Overview

A style sheet defines how each type will appear. It includes color and visibility for units, relations and schemas, and shape for schemas.

For a given annotation model (.aam file), it is possible to define as many style sheets as needed (.as files). Of course, in main cases, we define only one style sheet for a given model, but in some cases it is interesting to provide several ways to observe the annotations. Below are two screenshots of the same annotated text using two different style sheets :

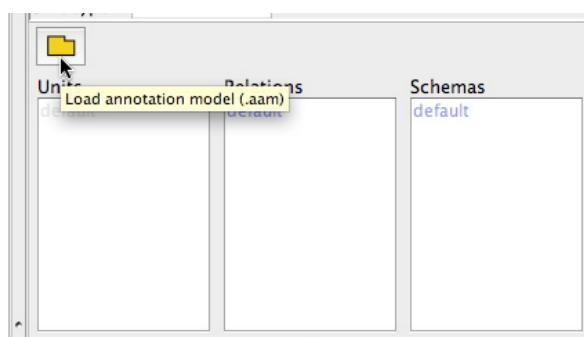


two different styles applied on the same corpus

### 7.2. Creating styles

Glozz provide a full wysiwyg interface to create and modify styles. Then, there is no need to know how it is stored in an .as file (for your information, it is in XML format).

First of all, we load the annotation model for which we want to create the styles :



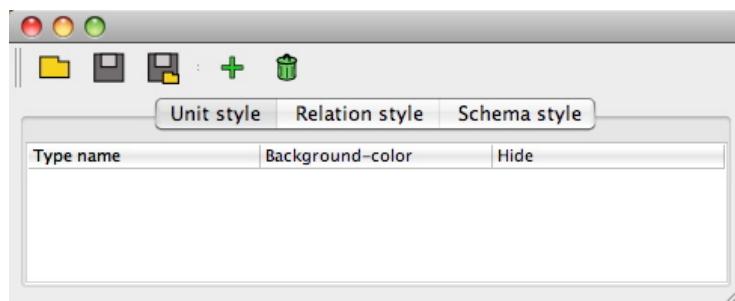
Let's take the coreference.aam file we've created in the last section. Since no style file is loaded, all the style names appear with the same color :

Units	Relations	Schemas
Noun Pronoun	PartOf Relationship	referenceChain

Now, we open the style editor :



And get a new window as follows :

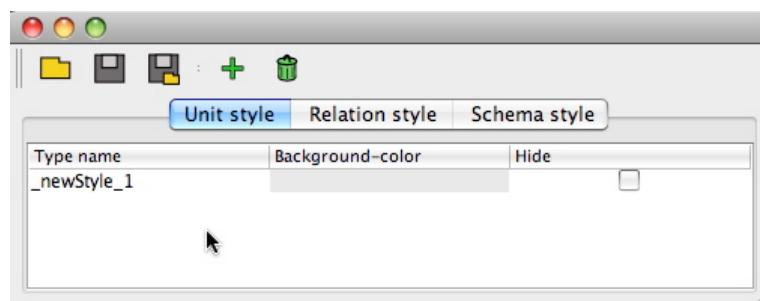


This editor provides a tabbed pane with 3 tabs : one for Units, one for Relations and one for Schemas. The buttons are as follows, from left to right :

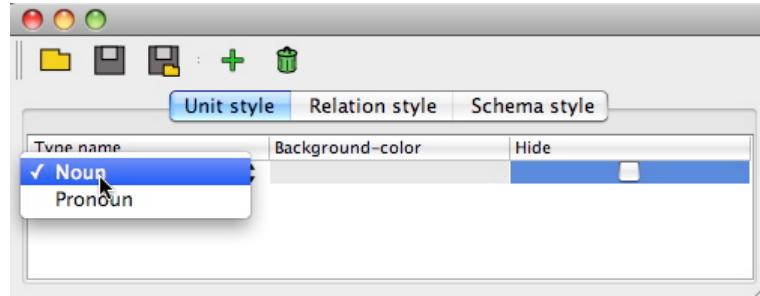
- open a .as style file
- save the current styles
- save the current styles as a new .as file
- add a new style
- remove selected style

### 7.3. Unit styles

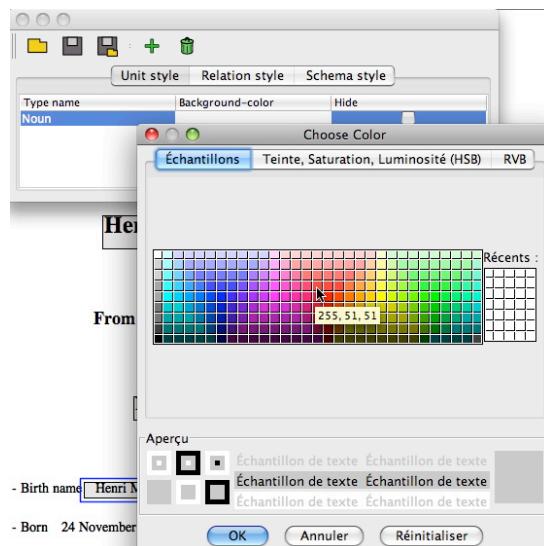
Let's first create two unit styles. We select the « Unit style » tab, then click on the **+** button. We get a new style, with default values (name is \_newStyle\_1, color is gray, and Hide is unchecked) as follows :



Then we can change this default values to the relevant ones. First, we have to modify the name, just clicking on it. It's very important that an annotation model is already loaded so that when clicking on the name, the possible unit type names are automatically proposed (here Noun or Pronoun) :



Then we can click on the color, in the next column, to set the Background-color :



And in the third column, we can set if the units of this style should be shown (by default), or hidden (when the box is checked). In main cases, the style files will be saved with all Hide fields unchecked, and this boxes will be checked/unchecked while working on the annotations to see only what we need. But of course, the style can be saved with some checked Hide boxes.

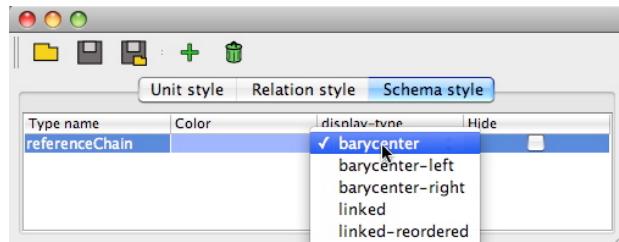
As a result, as soon as we've set some styles, the corresponding annotations are immediately styled, and even the annotation model appears with the associated colors (see in the right of the screenshot below, where Noun appears in red, and Pronoun in orange) :

## 7.4. Relation styles

Relation styles are almost the same as unit styles. The only difference is that in the second column, of course, line-color is concerned instead of background-color... So, we can apply the same method as in the previous section

## 7.5. Schema styles

Schema styles come with an additional field concerning the shape to give to the graphs, called display-type, in the third column.



Let's see how it behaves in the following table :

barycenter	barycenter-left	barycenter-right	linked	linked-reordered

- barycenter will show the schema as a star, with a central disc and a link to each element.
- barycenter-left and barycenter-right do the same, but put the disc in resp. the left margin and the right margin. This is quite convenient when using several types of schemas, in order not to have them all on the same side...
- linked and linked-reordered will show the schema as a path, from one element to the next one. The linked mode will use the natural order of the elements (the order they were added to the schema), whereas the linked-reordered will use the textual order, from top to bottom.

## 7.6. Special features

Besides the use of style sheets as we've just seen, some other special features may help for specific tasks. For this purpose, a « Color mode » chooser is provided in the main toolbar, set by default to « stylesheet » :



### 7.6.1 StyleSheet mode

By default, as just shown, the « Color mode » is set to « STYLESHEET », which means that each annotation will be shown with the color associated to its type with respect to the current stylesheet.

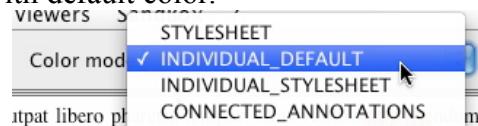
### 7.6.2 Individual colors modes

Sometimes, it may help to give to a certain annotation a given color, for some reason, whatever the stylesheet. This is possible using some special values in the **annotation model**. Indeed, when an annotation feature set contains a specific color element, then it is possible to use the color it specifies via one of the two dedicated modes.

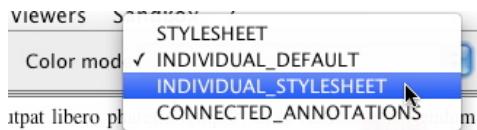
To do so, you have to add a `glozz:color` feature in each type of annotation you want in the annotation model. Then, for a given annotation, you can change its `glozz:color` feature value to one of the possible colors, as follows :

<b>Feature name</b>	<b>Feature value</b>	
glozz:color	green	<b>Lorem ips</b>
<b>Feature name</b>	<b>Feature value</b>	
glozz:color	red	<b>Lorem ipsu</b>

First of the two modes, named « INDIVIDUAL\_DEFAULT », will show each annotation having an individual color set in its feature-set with the given color, and the annotations not having one will be shown with default color.



Second mode, named « INDIVIDUAL\_STYLESHEET », will do the same, with the difference that annotations not having individual color will be shown according to the current stylesheet.



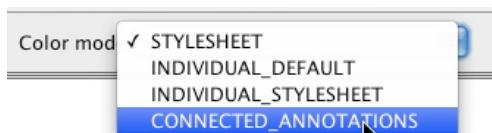
### 7.6.3 Co-reference chain color mode

When annotating co-reference chain, it is sometimes difficult to identify which annotation to which chain in the main view :

*Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisi sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum.*

Of course the « Grapher » can be used with its dedicated layout, but it is also possible to identify the chains through colors directly in the main views.

To do so, choose the option as follows in the toolbar :



And get each element of a connected chain given a same color (3 chains, hence 3 colors, in our example) :

*Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus.  
  Suspendisse lectus tortor, dignissim sit amet adipiscing nec, ultricies sed,  
  dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a,  
  semper congue euismod non, mi. Proin porttitor, orci nec nonummy  
  molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis  
  semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim.  
  Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras  
  vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit  
   odio eu enim, Pellentesque sed dui ut augue blandit sodales. Vestibulum  
  ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae;  
  Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum.*

## 8. GlozzQL : Glozz Query Language

GlozzQL is query language dedicated to Glozz annotations (Units, Relations, Schemas). It relies on simple concepts, and comes with a full graphical interface to create queries and observe results. It is available at any moment, even while annotating.

### 8.1. GlozzQL fundamentals

GlozzQL relies on two main concepts, which interoperate : **Constraint** and **ConstrainedAnnotation**.

#### 8.1.1 Constraint

A constraint expresses a condition an annotation must satisfy to be selected. 20 kinds of constraints are defined (more should come in next versions), which enable a large request scope. Some constraints concern a specific type of entity (Unit, Relation or Schema), whereas others are universal. Three special constraints concern constraints themselves (Not, Or, And) in order to combine them.

##### Universal constraints :

**Feature** : a certain feature must match a certain value (e.g. *gender = female*)

**Type name** : the type name must match a certain value (e.g. *pronoun*)

**Type** : specifies the type (among Unit, Relation and Schema)

**Author** : specifies the author's login. Useful for multi-annotated texts

**Last Author** : specifies the login of the last person who has modified the annotation

**Distance < x** : specifies a maximum distance, in characters, between this annotation and another specified entity

**Distance > x** : the same, but for a minimum distance

**After** : the annotation must come after (when reading) another specified one

**Before** : the same, but before

**Free** : specifies, in fact, no constraint ! Used to find, for instance, all the Units, or all the Relations, or Schemas.

##### Unit constraints :

**Text contains** : the text covered by the unit must contain a given string

**Regexp** : the text covered by the unit must match a given regular expression

**Covers** : the unit must cover another given unit

**Covered by** : the unit must be covered by another given unit

##### Relation constraints :

**Start** : the first argument of the relation must contain a given annotation. This constraint is settable in two ways : 1, we can specify whether this contain condition is recursive (search in depth) or not (first level search) ; 2, we can specify the minimum and maximum utterances of contained elements.

**Target** : the same with the second argument of the relation

#### Relation or Schema constraints :

**Contains** : it works the same way as the Start and the Target constraint (see above), but it may concern also a Schema, and if it concerns a Relation, then it will be fulfilled if the first OR the second argument fulfill the constraint.

#### Logic constraints :

**And** : concerns two given constraints, and combine them with the boolean operator « and ».

**Or** : concerns two given constraints, and combine them with the boolean operator « or ».

**Not** : concerns one given constraint, and applies the boolean operator « not » on it.

Let's finish this section with a very simple example to see how constrained are written :

C1 = *Text\_Contains* : *isotope* (scope = Unit)

This constraint has C1 for name, concerns Units only (scope), and asks a unit to contain the text « isotope ».

### 8.1.2 ConstrainedAnnotation

A ConstrainedAnnotation is a **set** of annotations which fulfill a given Constraint.

For instance,

Unit1 : C1

defines the set of Units which fulfill the C1 constraint, and which is called Unit1. Hence, Unit1 is the set of all units (of the current annotations) which contain the text « isotope ».

### 8.1.3 Incremental creation of Constraints and ConstrainedAnnotations

By definition, a ConstrainedAnnotation depends on a given Constraint.

Reversely, some Constraints depend on given Constraints. It's the case, for instance, of the **Contains** constraint :

C2 = Contains(Unit1) (scope = Relation, Schema)

is a constraint which expresses the fact, for a Relation or a Schema, to contain a Unit which contains the text « isotope ». For instance, to get the set of schemas which fulfill this constraint, we can create the ConstrainedAnnotation :

Schema1 : C2,

And to get the set of relations which fulfill this constraint, we create :

Relation1 : C2.

Hence, it is possible to build, step by step, richer and richer constrained annotations, in an incremental process.

## 8.2. GlozzQL Graphical User Interface

To open the GlozzQL GUI, click on the dedicated button in the main toolbar :



It makes the interface open at the first click, or show it again if it has disappeared :



Since GlozzQL relies on two main concepts, Constraints and ConstrainedAnnotations, the interface comes with two corresponding panels (numbers correspond to the screenshot):

- On the left side, the ConstrainedAnnotations panel embeds 3 frames :

1. Four buttons to create a ConstrainedAnnotation, among Unit, Relation and Schema, or even Any (which mean unspecified).
2. The list of ConstrainedAnnotations created (still empty in the screenshot), with three columns : the name of the ConstrainedAnnotation ; the Constraint it is based on ; the number of matches according to the current annotated text. A click on one item in the list will result in
3. The list of the annotations belonging to the selected ConstrainedAnnotation of frame 2. This panel works exactly the same way as the « annotations as predicates » tool seen in section 2.4.

Additionnaly, 4 buttons are available below panel 3, and will be introduced later

- On the right side, the Constraints panel :

4. The 20 constraints are available through 20 buttons, organized in 5 categories (Any, Units, Relations, Relations/Schemas, Logic)
5. The list of the constraints (still empty in the screenshot), with three columns : the ID of the constraint, its content (a short description), and its scope (what kind of ConstrainedAnnotations may rely on it).

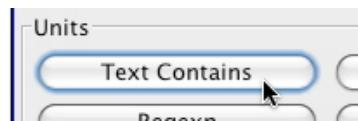
Additionnaly, two buttons are available below panel 5 : « remove selected » removes the constraint currently selected in panel 5 (click on a row to select it) ; « remove all » removes all the current constraints.

### 8.3. Some examples

#### 8.3.1 Units containing and not containing a text

Assume we load the second text of the « sandbox » menu, based on the « lorem ipsum ». We want to find all the Units which contain the text « sit ».

First, we have to create a « Text Contains » constraint, clicking the corresponding button in panel 4:



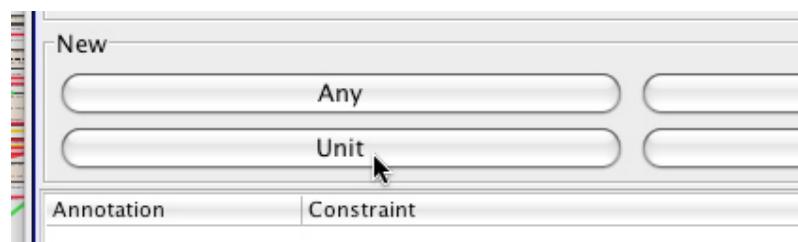
We specify the contained text as « sit », then click OK:



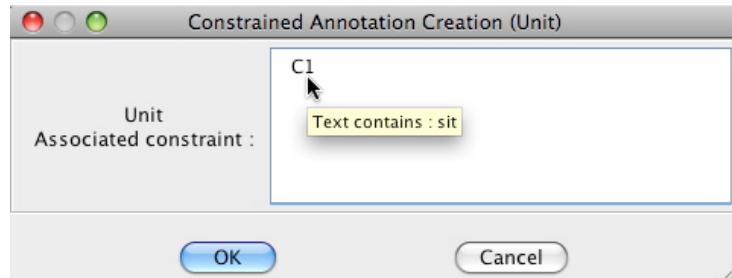
A new constraint is created, and appears in the constraints list, with ID « C1 », the content « Text contains : sit », and the scope being Unit (only Units may rely on this constraint):

ConstraintID	Content	Scope
C1	Text contains : sit	Unit

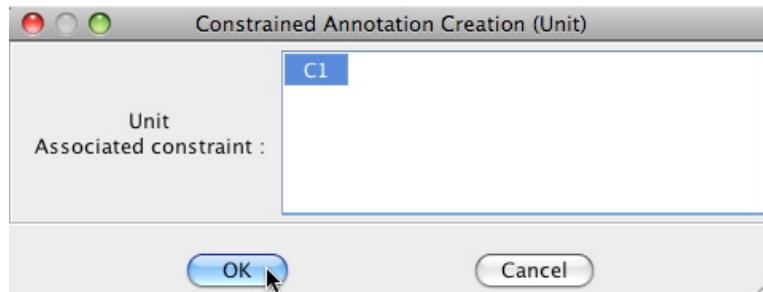
Now, we can create a ConstrainedAnnotation, of type Unit, click the button in panel 2 as follows :



A box appears, and shows all the possible constraints which can be used, that is to say all the constraints whose scope is Unit. At the moment, only C1 is available:



Note that when the mouse is over C1, its content appears as a tooltip (« Text contains : sit »), which makes it easier to find the correct constraint among all. Then, we click on C1, and click OK :



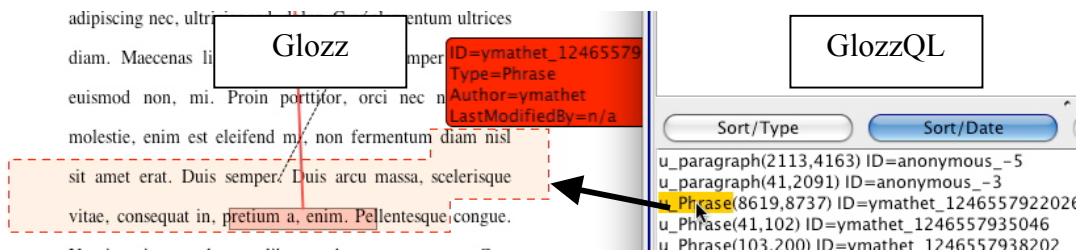
A new ConstrainedAnnotation is created, named Unit1, which constraint is « C1-->Text contains : sit », and which concerns 59 matches, as it appears in the list:

Annotation	Constraint	Matches
Unit1	C1-->Text contains : sit	59

We can click on the corresponding row, which makes all the utterances appearing in the panel below (panel 3) :

Annotation	Constraint	Matches
Unit1	C1-->Text contains : sit	59
<hr/>		
<input type="button" value="Sort/Type"/> <input type="button" value="Sort/Date"/> <input type="button" value="Show sel."/> <input type="button" value="Visible"/>		
u_paragraph(90605,92655) ID=anonymous_-66 u_paragraph(88555,90605) ID=anonymous_-65 u_paragraph(86505,88555) ID=anonymous_-64 u_paragraph(84430,86480) ID=anonymous_-62 u_paragraph(82380,84430) ID=anonymous_-61 u_paragraph(80330,82380) ID=anonymous_-60 u_paragraph(78255,80305) ID=anonymous_-58 u_paragraph(76205,78255) ID=anonymous_-57 u_paragraph(74155,76205) ID=anonymous_-56 u_paragraph(72080,74130) ID=anonymous_-54 u_paragraph(70030,72080) ID=anonymous_-53 u_paragraph(67980,70030) ID=anonymous_-52 u_paragraph(65905,67955) ID=anonymous_-50		

Then, we can click on any item in the list, to have it selected in Glozz :



So, we've just seen how to get all the results and, for each of them, how to have it selected in Glozz (in order to see it, to edit it, to remove it, and so on).

Now, let's try to get all the Units not containing this text. We already have the C1 constraint expressing a unit contains it. Let's create the opposite, using the NOT constraint :



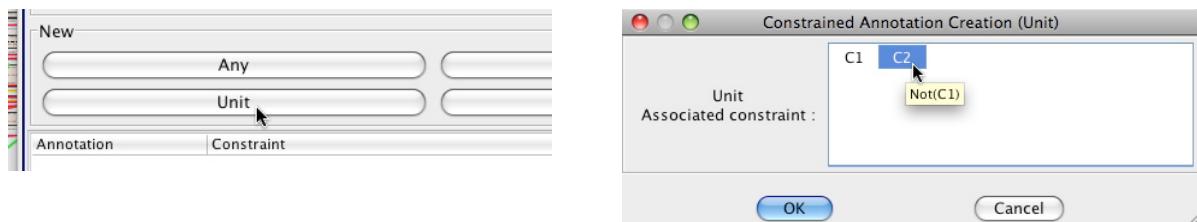
We apply it on C1 (being in fact the only possibility at the moment):



We get a new Constraint named C2 :

ConstraintID	Content	Scope
C1	Text contains : sit	Unit
C2	Not(C1)	Unit

Again, we create a ConstrainedAnnotation of type Unit, based on C2 :



And we get Unit2, with 51 matches :

Annotation	Constraint	Matches
Unit1	C1-->Text contains : sit	59
Unit2	C2-->Not(C1)	51

### 8.3.2 Focusing on one particular annotator

The queries we've just created offer respectively 59 and 51 matches, which concern units from any annotators. In fact, this annotated text contains annotations from a human annotator, and from a machine, the TXT\_IMPORTER.

We're going to get all the units containing « sit », like does Unit1, but from annotator whose login is « ymathet » only.

We click on « Author » in frame 4 :



We enter the requested login. Here we write the full login « ymathet », but it is possible to use a part of it only, and check « Limit to 'contains expr' ». For example, in this case, « ym » would work :



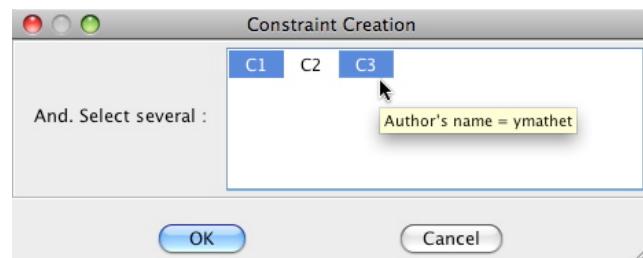
C3 is created. Note that this time, its scope is « Any », since it can be applied to a Unit, but also to a Relation or a Schema:

ConstraintID	Content	Scope
C1	Text contains : sit	Unit
C2	Not(C1)	Unit
C3	Author's name = ymathet	Any

We create now a « And » logic constraint, to combine C1 and C3:



A box appears where we have to choose at least two constraints. In our case, we click on C1 and on C2, in whatever order :



A new Constraint, C4, is created. Note that its scope is automatically computed by restriction of the different scopes of its contained constraints. Here, combining the scope Unit with the scope Any will result in the scope Unit :

ConstraintID	Content	Scope
C1	Text contains : sit	Unit
C2	Not(C1)	Unit
C3	Author's name = ymathet	Any
C4	And(C1,C3)	Unit

Now, to create the ConstrainedAnnotation associated to C4, we can process as we've done previously, clicking on Unit button, and so on.

However, we're going to use a **shortcut** : indeed, when a constraint is of a specific scope, i.e. **Unit, Relation or Schema**, we can create the associated ConstrainedAnnotation just double-clicking it in the list, as we do here clicking on the C4 row. If the scope is Any and you want to create a ConstrainedAnnotation of a specific type (for instance Unit), you have to use the usual way.

ConstraintID	Content	Scope
C1	Text contains : sit	Unit
C2	Not(C1)	Unit
C3	Author's name = ymathet	Any
C4	And(C1,C3)	Unit

This time, we get only 10 matches (41 others, in Unit2, do not belong to this annotator) :

Annotation	Constraint	Matches
Unit1	C1-->Text contains : sit	59
Unit2	C2-->Not(C1)	51
Unit3	C4-->And(C1,C3)	10

```

u_Phrase(8619,8737) ID=ymathet_1246557922026
u_Phrase(41,102) ID=ymathet_1246557935046
u_Phrase(103,200) ID=ymathet_1246557938202
u_Phrase(865,1279) ID=ymathet_1246557946798
u_Phrase(2214,2358) ID=ymathet_1246557953609
u_Phrase(4438,4950) ID=ymathet_1246557960954
u_Phrase(7239,7650) ID=ymathet_1246557965878
u_Paragraphe(4163,6213) ID=ymathet_1246557977596
u_Paragraphe(2113,4163) ID=ymathet_1246557984190
u_Paragraphe(6237,8286) ID=ymathet_1246557991290

```

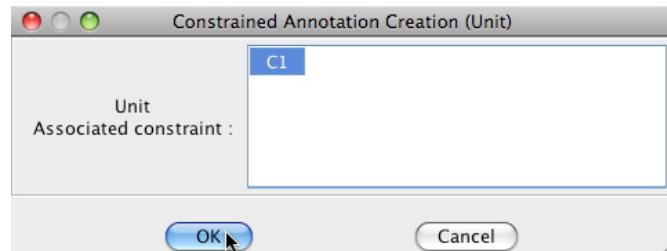
### 8.3.3 Deeper queries with schemas and relations

Let's see some more complex queries, which rely *Contains* and *Target* constraints and involve deep structures.

Assume we want to get all the Relations whose target contains a Schema which contains a Unit whose type is « Verbe ». To do so, we first create C1 as follows :

ConstraintID	Content	Scope
C1	TypeName = Verbe	Any

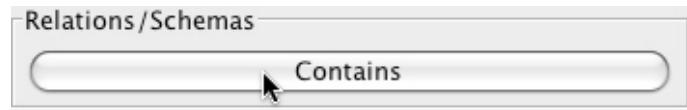
Then we create Unit1, the set of units whose type name is « Verbe » :



We get it in the list :

Annotation	Constraint	Matches
Unit1	C1-->TypeName = Verbe	16

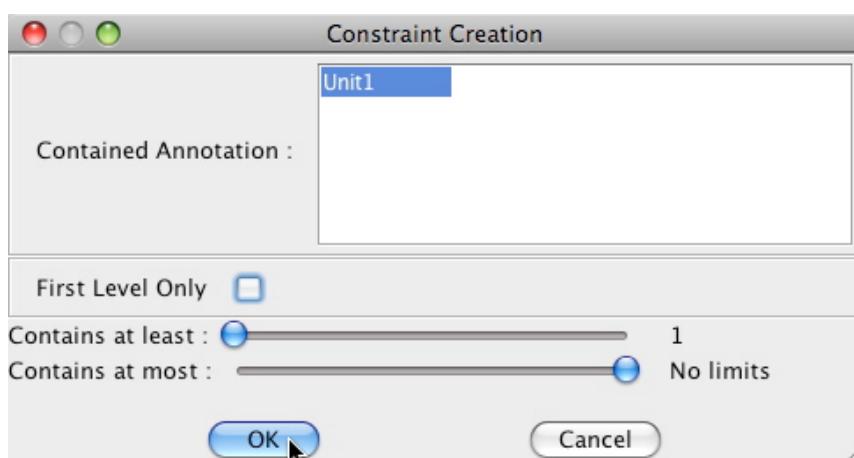
Now we create a Contains constrained by clicking this button :



Then, we're ask to choose the ConstrainedAnnotation we want to be contained. Here, only Unit1 is available, and we choose it. This constraint can be set by three parameters :

- First Level Only : if checked, means we want the annotation to be contained at first level, not deeply ; if unchecked, it allows for instance the annotation to be contained to be itself contained in another annotation, this latter being itself contained.
- Contains at least : the minimum number of utterances being contained
- Contains at most : the same for maximum

In our case, we've unchecked First Level Only, in order to enable a deep search, and leave the default values for least and most, so that any number of utterances is allowed :



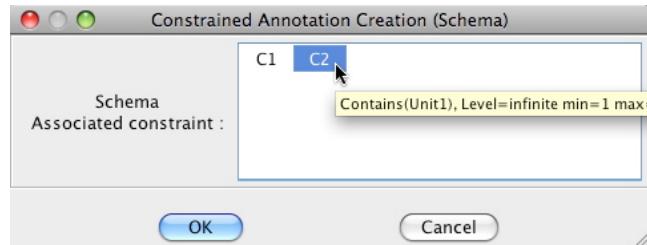
We get then C2 in the list :

ConstraintID	Content	Scope
C1	TypeName = Verbe	Any
C2	Contains(Unit1), Level=infinite mi...	Rel/Sc...

We ask to create a Schema based on C2 :



We choose C2 among the two possible constraints which scope is compliant :



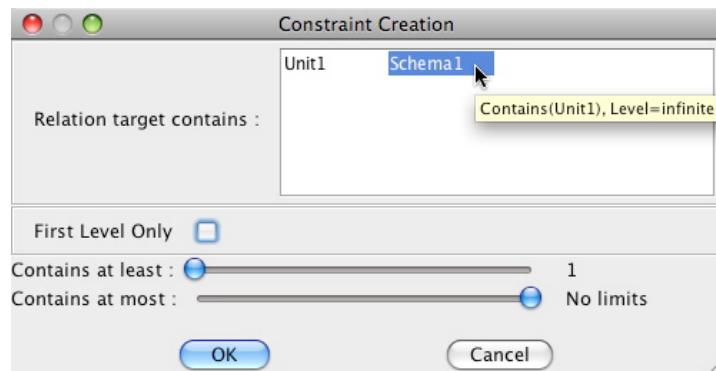
and get Schema1 in the list, with 4 utterances :

Annotation	Constraint	Matches
Unit1	C1-->TypeName = Verbe	16
Schema1	C2-->Contains(Unit1), Level=infinite min=1 max=No limits	4

Now we ask to create a *Target* constraint :



whose associated ConstrainedAnnotation is Schema1 (with First Level Only unchecked) :



and get Relation1, with one utterance, being an « Elaboration » :

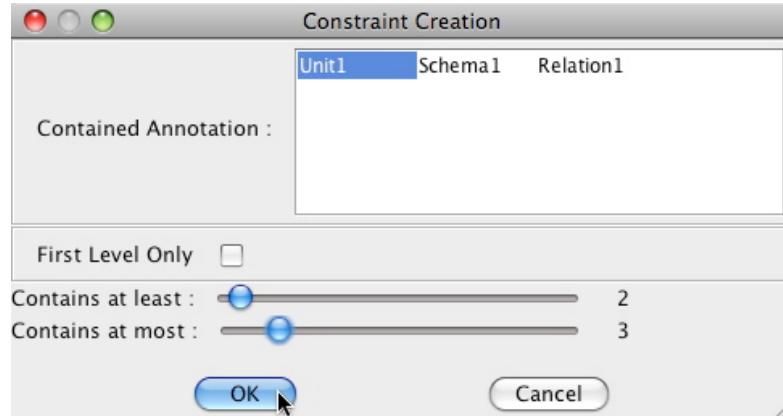
Annotation	Constraint	Matches
Unit1	C1-->TypeName = Verbe	16
Schema1	C2-->Contains(Unit1), Level=infinite min=1 max=No limits	4
Relation1	C3-->TargetContains(Schema1), Level=infinite min=1 ma...	1

Buttons at the bottom: Sort/Type, Sort/Date, Show sel., Visible.

The status bar at the bottom shows: r\_Elaboration(ymathet\_1246558357351,ymathet\_1246558330689) ID=ymathet\_1246558398644

One and only one relation matches this complex query.

Note that if we've had specified a range of number of utterances concerning the Schema, for instance minimum 2 and maximum 3 :



we would have got, with this corpus, no match at all for associated Schemas (see Schema2 below) :

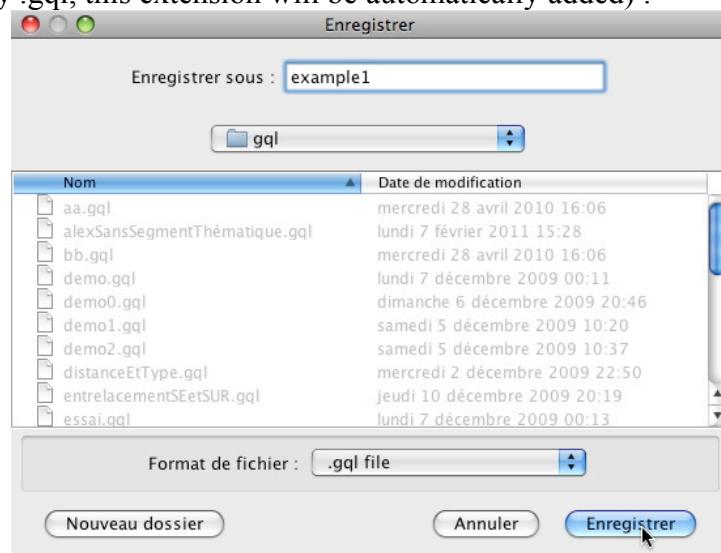
Annotation	Constraint	Matches
Unit1	C1-->TypeName = Verbe	16
Schema1	C2-->Contains(Unit1), Level=infinite min=1 max=No limits	4
Relation1	C3-->TargetContains(Schema1), Level=infinite min=1 ma... 1	
Schema2	C4-->Contains(Unit1), Level=infinite min=2 max=3	0

## 8.4. Saving and loading GlozzQL queries

At any moment, it is possible to save the current set of queries. This will save the list of Constraints, and the list of ConstrainedAnnotations, as a .gql file (in XML format).

This won't save the results, but of course you can later load again your corpus and your .gql file, and then get the results again.

To save, click on the « save » button in top left of the window, and enter a name (if the name doesn't finished by .gql, this extension will be automatically added) :



Loading is as simple as saving. Be careful, loading a .gql file will result in erasing all current Constraints and ConstrainedAnnotations before loading new ones. If needed, save them before.

If you are a developer, you may be interested in generating queries in gql format from your own programs. No DTD is provided at the moment, but it is very simple XML. Of course, if needed, do not hesitate to contact the authors for further details.

## 8.5. GlozzQL Basket

GlozzQL provides a « basket » in which it is possible to store results from one or several queries, in order to do some specific action on these annotations, either saving them as a new .aa file, or removing them from current annotations.

### 8.5.1 Feeding the basket

Each time you click on a ConstrainedAnnotation in the list of frame 3, you can then click on « Add results to Basket » button, just below.

For instance, with our last example, we can click on Schema1, and get the 4 utterances appearing in the frame 4 :

Annotation	Constraint	Matches
Unit1	C1-->TypeName = Verbe	16
Schema1	C2-->Contains(Unit1), Level=infinite min=1 max=No limits	4
Relation1	C3-->TargetContains(Schema1), Level=infinite min=1 ma...	1

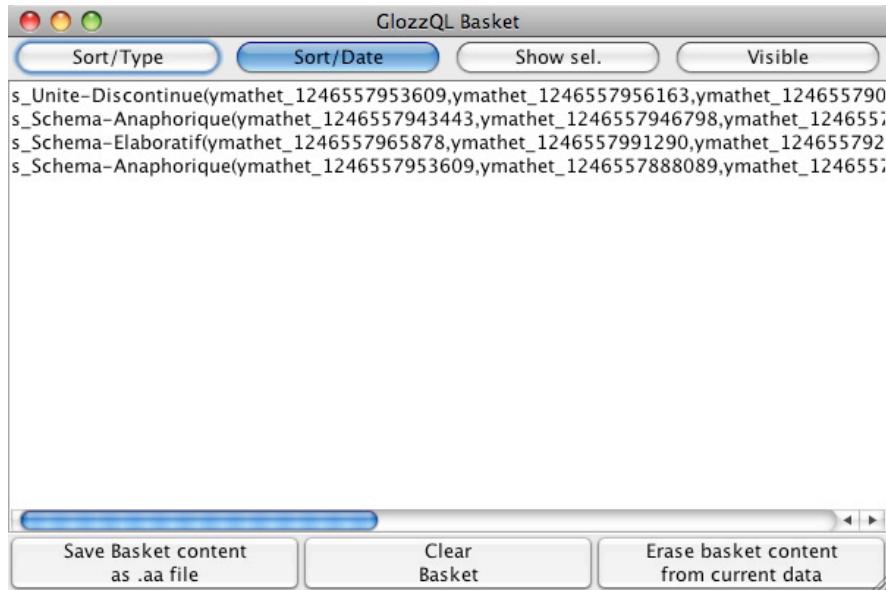
  

Sort/Type	Sort/Date	Show sel.	Visible
s_Unite-Discontinue(ymathet_1246557953609,ymathet_1246557956163,ymathet_124655790398) s_Schema-Anaphorique(ymathet_1246557943443,ymathet_1246557946798,ymathet_1246557950) s_Schema-Elaboratif(ymathet_1246557965878,ymathet_1246557991290,ymathet_124655792202) s_Schema-Anaphorique(ymathet_1246557953609,ymathet_1246557888089,ymathet_1246557956)			

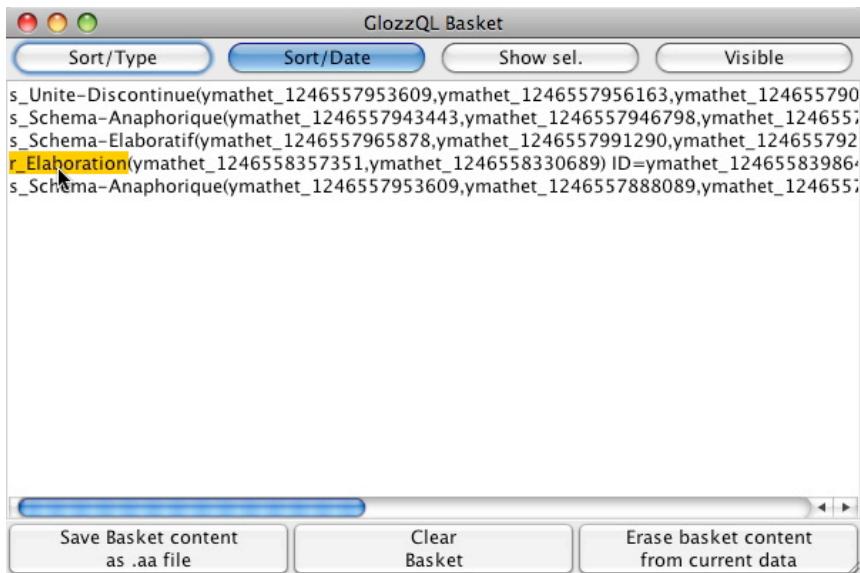
  

Recompute All	Remove Selected	Add results to Basket	Unify Variables
---------------	-----------------	-----------------------	-----------------

Then, clicking on « Add results to Basket », we get a new window, which shows the basket content :



Now, back to the GlozzQL main window, we can click on Relation1 in frame 3 (which owns 1 utterance), and the click again on « Add results to Basket ». Now, the basket owns a fifth annotation :



We can go on as long as needed, and add to the basket as many annotations as we want.

### 8.5.2 Save basket content as .aa file

In the bottom-left corner of the basket window, the button « save basket content as .aa file » enables to create a new annotation file containing only the annotations currently present in the basket.

When doing so, you will be asked to enter a file name. This is useful for instance to split the annotations in several files depending on a criterion such as the author's name or so.

### 8.5.3 Erase basket content from current data

The second action the basket enables is to remove the annotations currently present in the basket from the current loaded annotation in Glozz. This action combined to the other one (save basket) make it possible to reorganize completely an annotated corpus.

## 8.6. Advanced concepts

Two advanced concepts, very easy to launch, but maybe more difficult to understand, enhance the GlozzQL capabilities.

### 8.6.1 Two ways constraints

The « two ways constraints » option of GlozzQL means that when a constraint argument is a constrainedAnnotation, then this constrainedAnnotation is itself constrained by the reciproque of this constraint. An example will be much more meaningful than this definition.

Going back to the last example, we've created Unit1, based on C1, with 16 utterances. Then, we created Schema1, whose constraint involves Unit1, with 4 utterances :

Annotation	Constraint	Matches
Unit1	C1-->TypeName = Verbe	16
Schema1	C2-->Contains(Unit1), Level=infinite min=1 max=No limits	4
Relation1	C3-->TargetContains(Schema1), Level=infinite min=1 ma...	1

Now, if we choose to activate the « two-ways constraints » option :

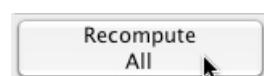


then, Unit1 will be also constrained by C2, which means that a unit will belong to Unit1 if it is of type « Verbe », but also if it exists (at least) un schema belonging to Schema1 which contains this Unit.

Moreover, since Schema1 appears in C3, it is also constrained by it, so that a schema will belong to Schema1 if it contains a element of Unit1, but also if it exists a relation belonging to Relation1 whose target is this schema.

Consequently, Schema1 is constrained by C3 (in addition to its natural constraint C2), and Unit1 is constrained by C2 and by C3 (in addition to its natural constraint C1), which is very restrictive compared to simple way constraints...

Let's see what happens with our corpus now we've activated this option. To do so, we have to recompute all, with the eponym button located in the bottom-left of the window :



And we see that with these constrained working in the two ways, the matches are of Schema1 go from 4 to 1, and the matches of Unit1 go from 16 to 1 :

Annotation	Constraint	Matches
Unit1	C1-->TypeName = Verbe	1
Schema1	C2-->Contains(Unit1), Level=infinite min=1 max=No limits	1
Relation1	C3-->TargetContains(Schema1), Level=infinite min=1 ma...	1

What we've discovered here is that there is only one unit of type « Verbe » which is contained in a schema (at any level) which is contained (at any level) in the target of a relation.

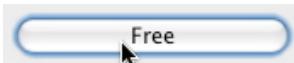
This way, the semantics of the constraints becomes much more powerful with no additional work for the user. It's up to you to see when you need to activate it or not, depending on what semantics you want to get. You can activate or deactivate it at any moment, but do not forget to **recompute all** each time you want to see the new results.

### 8.6.2 Unification mechanism

In GlozzQL, unification means considering ConstrainedAnnotations as variables, and the set of constraints as an equation system. Here again, an example will be more meaningful :

Assume we want to find all relations being oriented in the opposite way of reading, i.e. from bottom to top.

First, we create Any1, the set of all annotations. To do so, we use the « Free » constraint, which accept every annotations :

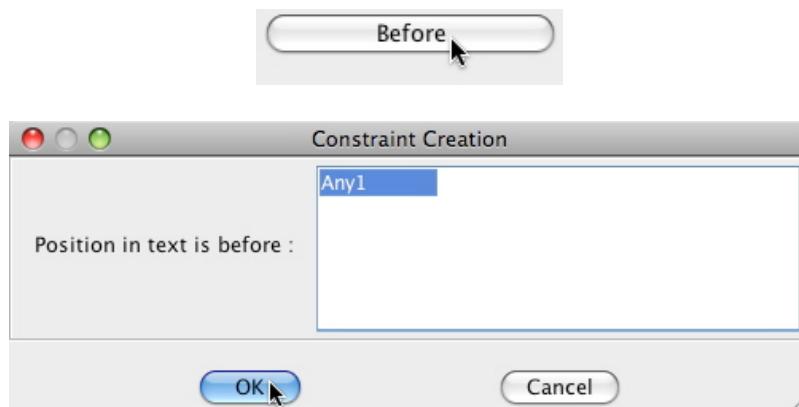


ConstraintID	Content	Scope
C1	Always true	Any

Then, by double-clicking it, we get Any1 with 128 utterances :

Annotation	Constraint	Matches
Any1	C1-->Always true	128

Now, we create Any2, with the constraint of being located before an Any2.



We get C2 :

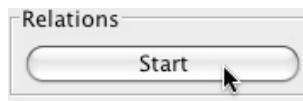
ConstraintID	Content	Scope
C1	Always true	Any
C2	Before(Any1)	Any

And double-clicking it, we get Any2 with 127 utterances :

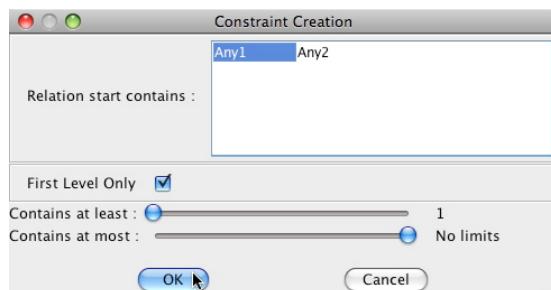
Annotation	Constraint	Matches
Any1	C1-->Always true	128
Any2	C2-->Before(Any1)	127

Now, we create Relation1, which relies on the combination of two constraints, Start with Any1 and Finish with Any2, as follows.

We create a Start constraint :



which concerns Any1 :



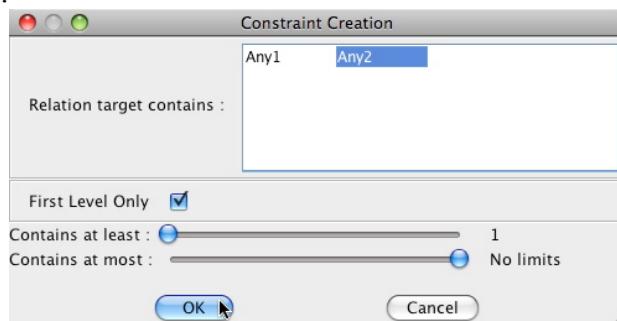
It's created as C3 :

ConstraintID	Content	Scope
C1	Always true	Any
C2	Before(Any1)	Any
C3	StartContains(Any1), Level=1 min... Relation	

We create a Target constraint:



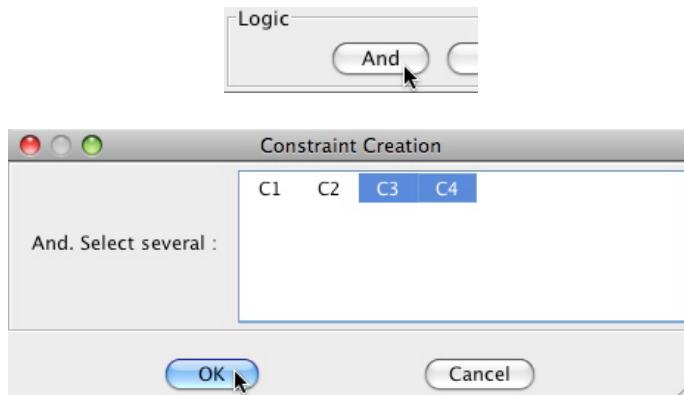
which concerns Any2 :



It's created as C4 :

ConstraintID	Content	Scope
C1	Always true	Any
C2	Before(Any1)	Any
C3	StartContains(Any1), Level=1 min...	Relation
C4	TargetContains(Any2), Level=1...	Relation

We combine C3 and C4 with a And constraint :



and get it as C5 :

ConstraintID	Content	Scope
C1	Always true	Any
C2	Before(Any1)	Any
C3	StartContains(Any1), Level=1 min...	Relation
C4	TargetContains(Any2), Level=1...	Relation
C5	And(C3,C4)	Relation

By double-clicking on C5, Relation1 is created as follows, with 12 utterances.

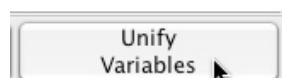
Unfortunately, as we can see in the next screenshot, some of the utterances are oriented from top to bottom, contrary to what we expect :

Annotation	Constraint	Matches
Any1	C1-->Always true	128
Any2	C2-->Before(Any1)	127
Relation1	C5-->And(C3,C4)	12

The reason is that in fact, Any1, Any2 and Relation1 are sets, not variables. It means for instance, that a relation belonging to Relation1 links any element of Any1 to any element of Any2. Of course, all the elements of Any2 **are not necessarily before all the elements of Any1**, but before at least one of them.

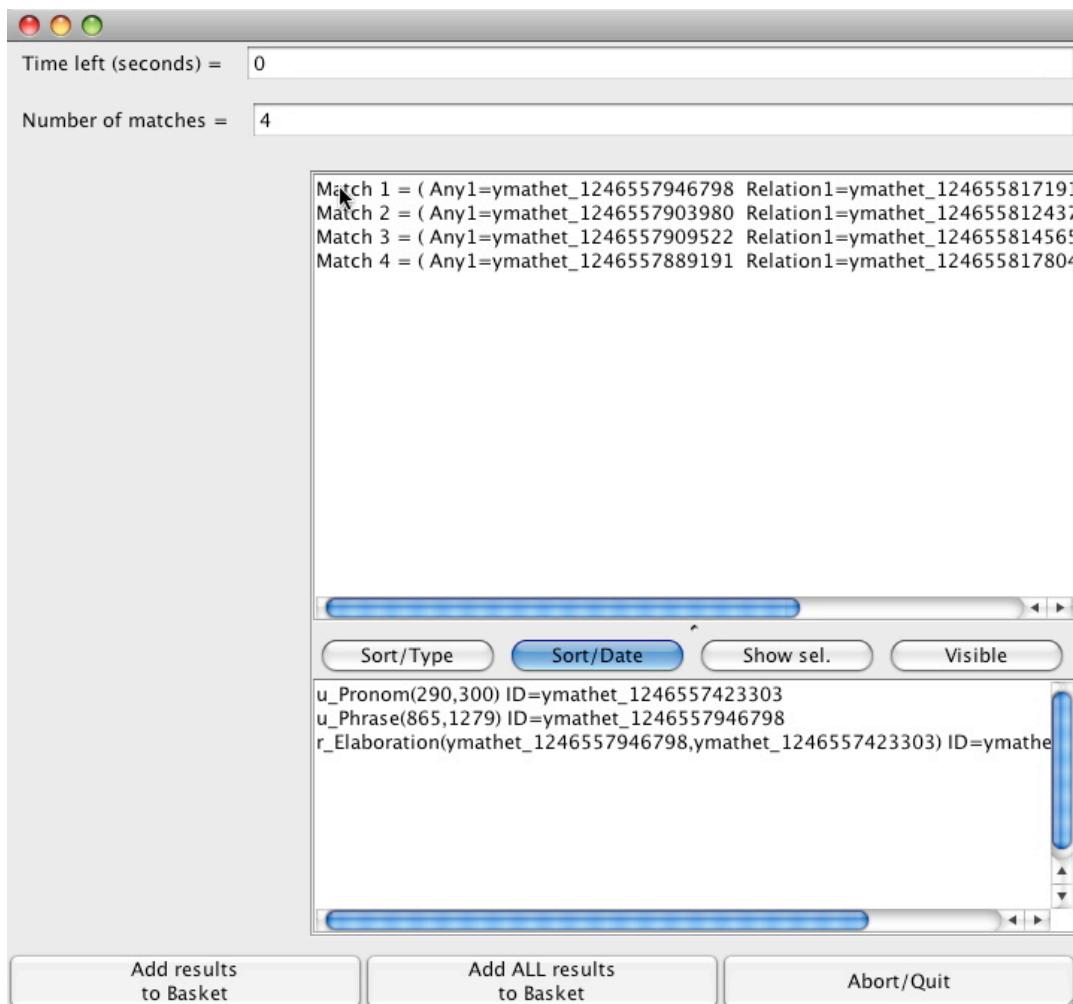
What we need for our purpose is that we talk about the same entities when we say that Any2 is before Any1 and that Relation1 goes from Any1 to Any2. This is what we call here unification mechanism.

We launch this process with the dedicated button, located in the bottom of the window :



A new window appears, dedicated to unification mechanism, and shows a countdown in seconds of the time left, because this computation happens to be long in some cases.

During the process, the results appear progressively, and we finally get the number of matches, and the corresponding list. We can click on any of the results, and then see its content in a frame below (for instance Match 1 in the screenshot) :



Now, only 4 matches appear (out of 16 without unification), such as the one with r\_Elaboration, and all of them are correct (oriented from bottom to top). These results can be added to the basket.

## **8.7. Use cases**

We report here some possible use cases of GlozzQL.

### **8.7.1 Splitting annotations by authors, types, etc.**

Using the relevant constraints and then the basket, it is possible to create new .aa files with for instance all (and only) the annotations of a given annotator.

### **8.7.2 Looking for mistakes**

In most annotation campaigns, some configurations are forbidden. However, the annotation model doesn't provide such capability, and an annotator may create wrong annotations according to the campaign. For instance, it should be the fact that a *Proposition* cannot contain more nor less than one *Verb*.

In these cases, it may be possible to express these forbidden configurations in GlozzQL, save it as, for instance, a forbidden.gql file, and provide the annotators with it. Hence, at any moment, the annotators can check their annotations by clicking on « recompute all » : if any utterance appears in the results, it means a mistake was done, and a click on the utterance(s) enable to select it immediately (in order to correct or remove it).

### **8.7.3 Basic statistics**

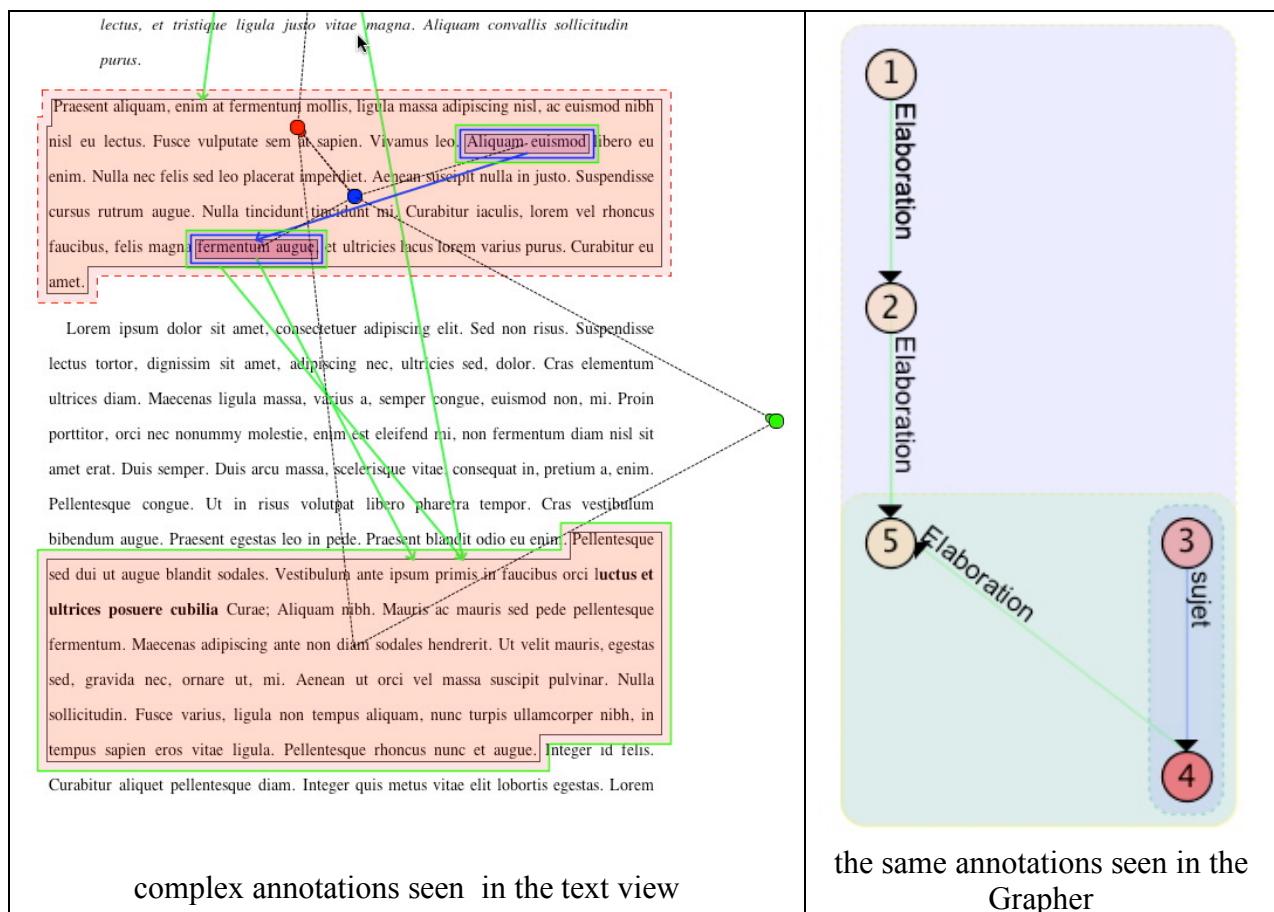
Getting for each ConstrainedAnnotation, the number of corresponding utterances, it is possible to use GlozzQL to check some assumptions, and do some statistics.

However, the use of SQL export may give faster results, and, moreover, enable to work with several annotated texts at the same time, contrary to GlozzQL (in current 1.0.0 version).

## 9. « Grapher » : annotations shown as a graph

### 9.1. Overview

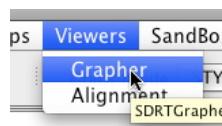
With some complex annotation structures (namely with imbrication of schemas and relations), it may become difficult to handle the main views (see the screenshot in the left below). Glozz provides a module which renders the annotations as a graph (see in the right below) :



### 9.2. Interface

#### 9.2.1 Launching Grapher

To launch the interface, use the menu as follows :



## 9.2.2 Auto-selection

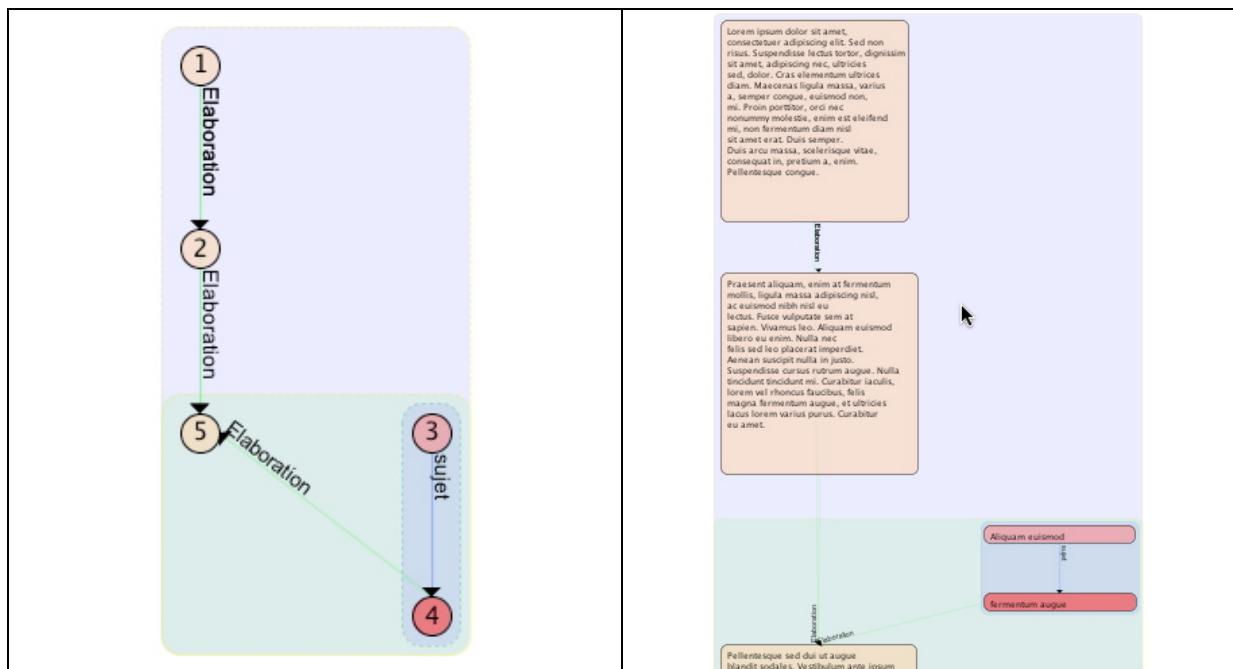
When moving the mouse over an element of the graph, this latter is automatically selected in all the other views.

## 9.2.3 Zoom-in, Zoom-out

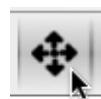
It is possible to zoom-in / zoom-out either using the box in the top of the window, or, more friendly, using the **wheelmouse**.

## 9.2.4 Showing embedded text

Two modes are available regarding the way the units are represented in the grapher. Default one shows each unit as a simple number (see the left screenshot below), but a second one shows the embedded text (see the right screenshot below) :

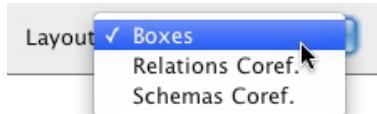


To switch from one mode to the other, use the button as follows :



## 9.3. SDRT layout

By default, the graph is shown in a SDRT-like mode, using boxes to show embedding structures. It's the best mode for annotations related to the SDRT theory. It can be reselected by choosing « boxes » in the layout option of the toolbar :



## 9.4. Co-reference chains layouts

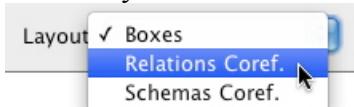
For annotations related to reference chains, two special layouts were created, depending the annotations use relations or schemas to create chains.

### 9.4.1 Co-reference using relations

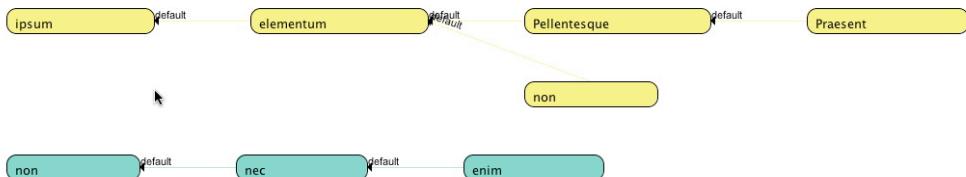
Assume a text is annotated with two co-reference chains, using units linked together by relations :

*Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non risus.  
Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed,  
dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a,  
semper congue euismod non, mi. Proin porttitor, orci nec nonummy  
molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis  
semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim  
Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras  
vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit*

Then, choosing the « Relations Coref. » layout as follows :



Will result in a graph showing each individual chain horizontally :



### 9.4.2 Co-reference using schemas

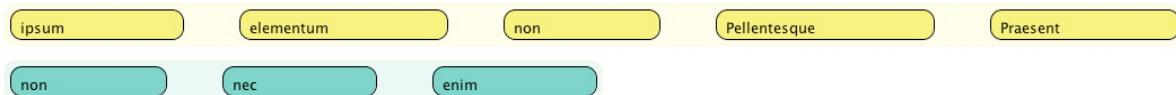
Assume a text is annotated with schemas, one schema embedding all the units a given reference chain :

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non risus.  
 Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed,  
 dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a,  
 semper congue, euismod non, mi. Proin porttitor, orci nec nonummy  
 molestie, enim est eleifend mi. fermentum diam nisl sit amet erat. Duis  
 semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a enim.  
 Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras  
 vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit  
 odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum

Then, choosing the « Schemas Coref. » layout as follows :



Will result in a graph showing each schema horizontally :



## **10. « Glozz Aligner » : Alignment and agreement tool for multi-annotated texts**

### **10.1. Principles**

A new approach of alignment and inter-annotator agreement measurement has been developed in (Mathet&Widlöcher 2011), which provides a unified method to do both (aligning and measuring). Please, refer to this article to be introduced to this method.

To sum up the agreement measure principle, the set of multi-annotations is considered as generating some disorder, called entropy, compared to a set of annotations with full agreement. Hence, each set of multi-annotations is given an « entropy » value.

Besides, a « random entropy » is computed by automatic analysis of a reference corpus of a given campaign. This can be done one for all (for a given campaign).

Then, what is called agreement is the value :

$$\text{Agreement} = (\text{randomEntropy} - \text{entropy}) / \text{randomEntropy}$$

For a full agreement, entropy = 0 and agreement=1.

For annotators being not better than random, the agreement=0.

In some case, it is possible to get a negative agreement value when annotators do worse than random.

So, what we need to do when we want to get the agreement of a multi-annotated text is :

- to compute the random entropy of the corpus
- to compute the entropy of this text

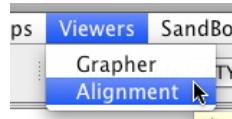
These principles are integrated in Glozz in a dedicated tool called Aligner. They will be shown in section 10.3, after introducing the views provided by the Aligner tool in section 10.2.

However, please note that this tool is still under development (current version of Glozz is 1.0.0 when writing this version of the manual), and should integrate in the future the possibility to choose the « dissimilarity » function to work with, and to adjust the inter-categorial matrix.

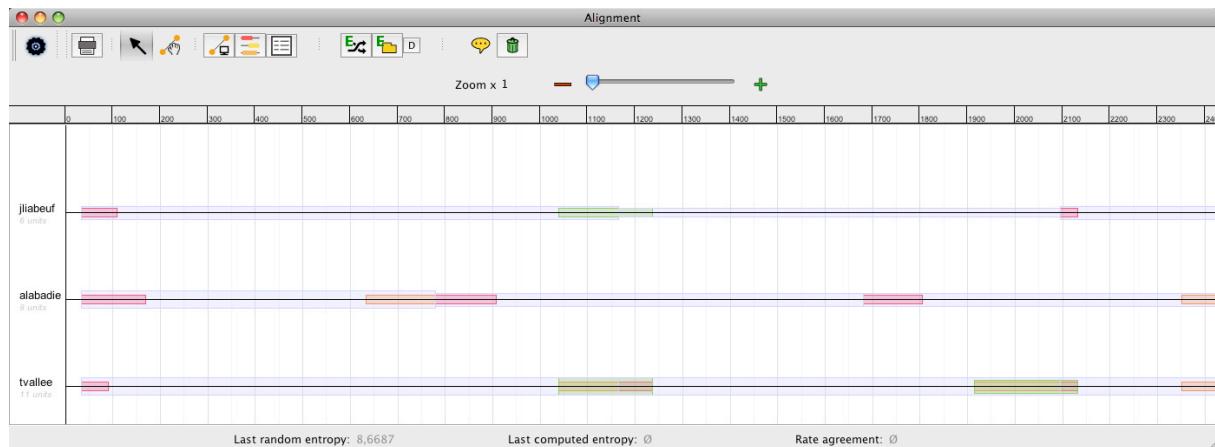
### **10.2. Aligner special view**

When a text is annotated by several annotators (human or software), Glozz provides a special view which consists in separating annotations from each annotator, and showing them on an horizontal line (one line by annotator). At the moment (Glozz version 1.0.0), **only Units** are considered by this tool.

To launch this module, use the Viewers menu and click on Alignment :



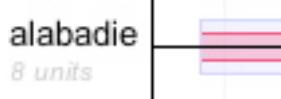
## 10.2.1 overview



In this example, annotations come from three different annotators, and so are separated into three horizontal lines.

One line represents the annotations from one annotator on the whole text, from the first character at the left to the last character to the right.

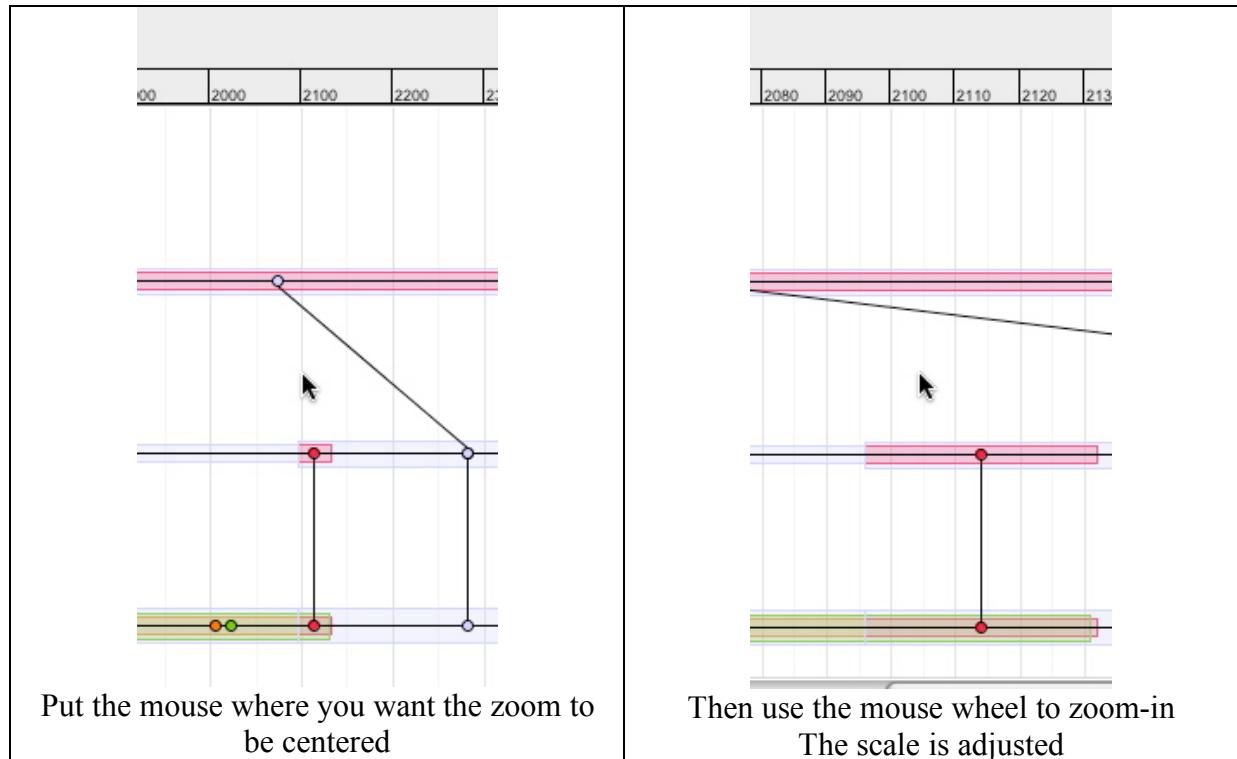
On the left of each line is shown the annotator's login, and the number of annotations, for example there are 8 units created by annotator « alabadie » :



A scale is shown above the views, going from character 0 to character 2400 in our example.

## 10.2.2 Zooming-in, zooming-out

It is possible to zoom-in or zoom-out with the mouse wheel or with the Zoom cursor.

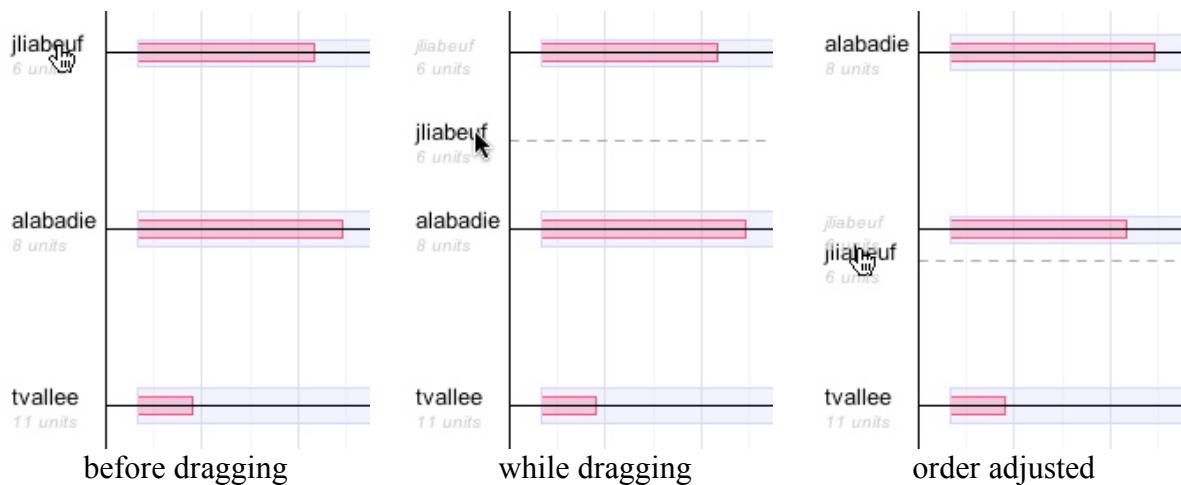


### 10.2.3 Moving to the right or to the left

When the view is zoomed, you see on the screen only a part of the whole text, but you can move to the left or to the right with a drag&drop (press the mouse button, move the mouse, then release the button).

### 10.2.4 Changing the order the annotators appear

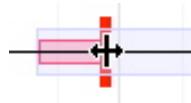
The annotators are ordered as the first annotations come in the annotation file, but you may want to re-order them in order to better compare them, for some reasons. This is possible by dragging the name of an annotator (in the left side) to the desired position. All the views are automatically adjusted while dragging.



### 10.2.5 Adjusting units positions

Each unit appearing in this alignment tool can be adjusted by a drag&drop action on either one of its bounds, or in its body.

To move one of its bounds, put the mouse over it (it appears in red color), press, move while pressing (a green vertical line appears in order to help alignment with other annotations), then release :

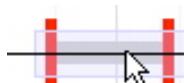


before dragging a bound



while dragging a bound

To move the whole unit, put the mouse over its body (the two bounds appear in red color), and do the same.

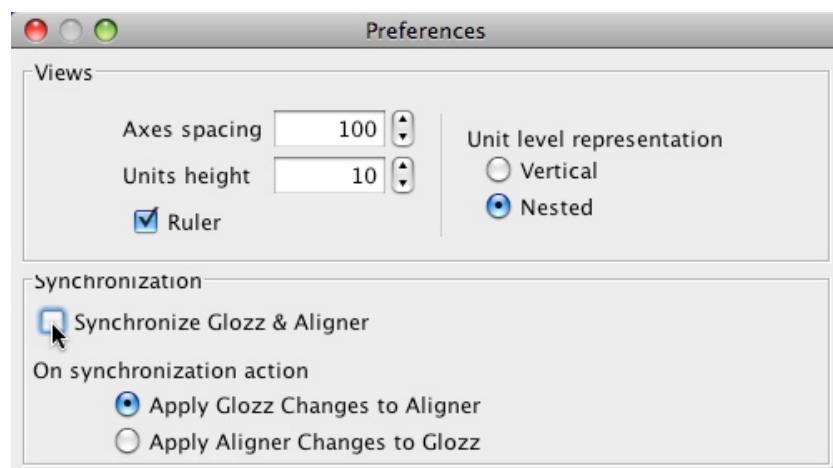


Selecting a whole unit in order to move it

However, by default, these modifications in this tool won't have any consequence on the real annotations.

If you want to get that behavior, i.e. that any editing action in this tool result in an editing action on the annotations this view comes from, you have to select it in the preferences.

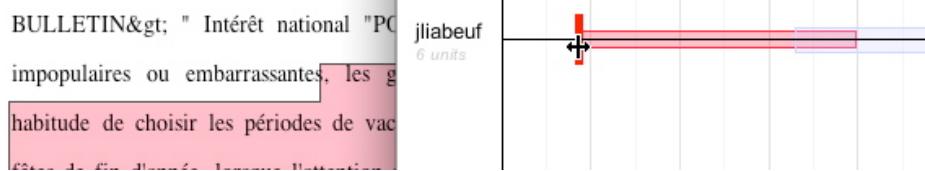
To active the preferences panel, click on this button in the top-left of the window :



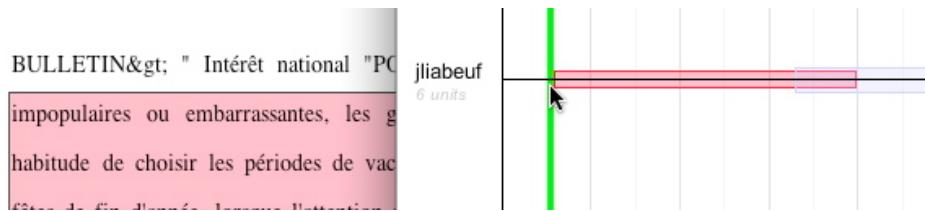
Then click on the check box « Synchronize Glozz & Aligner ». However, at the time you do that, the real annotations and the ones you're working with in the aligner tool may be different (in the case you've moved some units in the tool without having activated this option). It's the reason before selecting this option, you have to choose whether « applying glozz changes to aligner », which means that the aligner tool will go back to the initial position of each unit, or « applying aligner changes to glozz », which means that all previous change in the aligner tool will result in a change in the annotations in Glozz.

Once this option is activated, you can observe the real time adjustment in Glozz of any change in the aligner tool :

Before dragging (see the red unit in Glozz starting at the comma) :



While dragging (see the red unit in Glozz starting now at « impopulaires » word) :

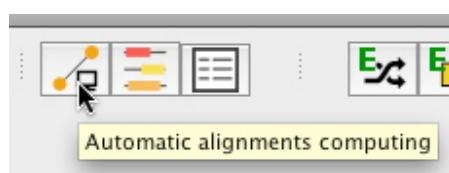


### 10.3. Alignment tool

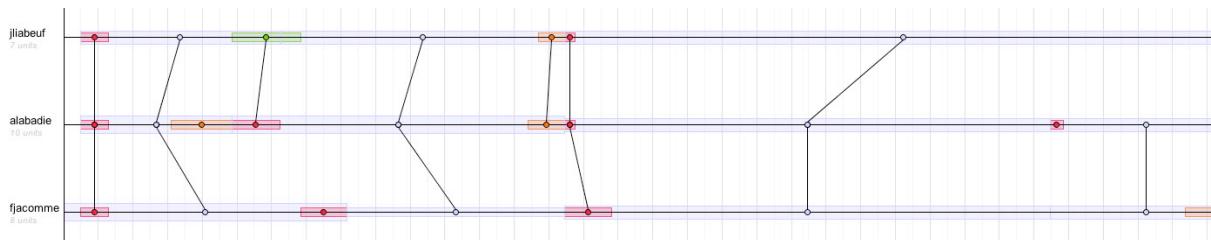
An alignment consists in considering that several annotators have annotated a same phenomenon. This is obvious when they have all annotated the same unit, with the same exact bounds and the same category. However, this is far too restrictive in numerous annotation campaigns, and it is the reasons why a method enabling some tolerance has been developed.

#### 10.3.1 Auto alignment

Once the annotations are loaded and the Aligner is launched, click on the button as follows :



Then you get some semi vertical lines which consider units from different annotators as aligned :



As you can see, some alignements involve all the (three) annotators, wheras others involve two only, or one only (units leaved alone).

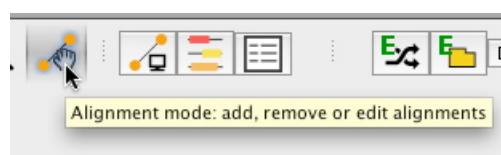
Remark: in this example, you can see that most alignment concern a same category (rose or orange), but one concern a green unit and a rose one. The reason is that we've set the categorial matrix so that green and rose category may be aligned. In the next versions of this tool, it will be possible to set this matrix (manually or automatically), but at present time, in the version 1.0.0 (and contrary to the example above), only units of the same category may be aligned.

## 10.3.2 Manual alignment

You may need to make some alignments manually, for instance in order to create some real examples of what you consider as being aligned or not (and then use the print function, for instance).

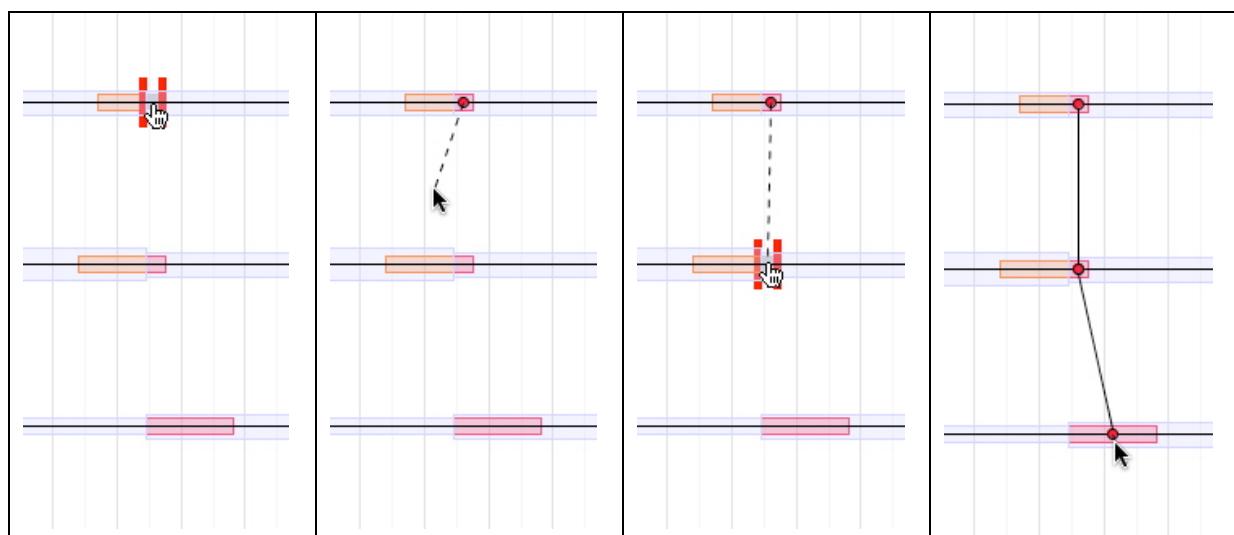
### 10.3.2.1 Choosing special mode

First of all, use the special mode by clicking on the button as follows :



### 10.3.2.2 Creating a new alignment

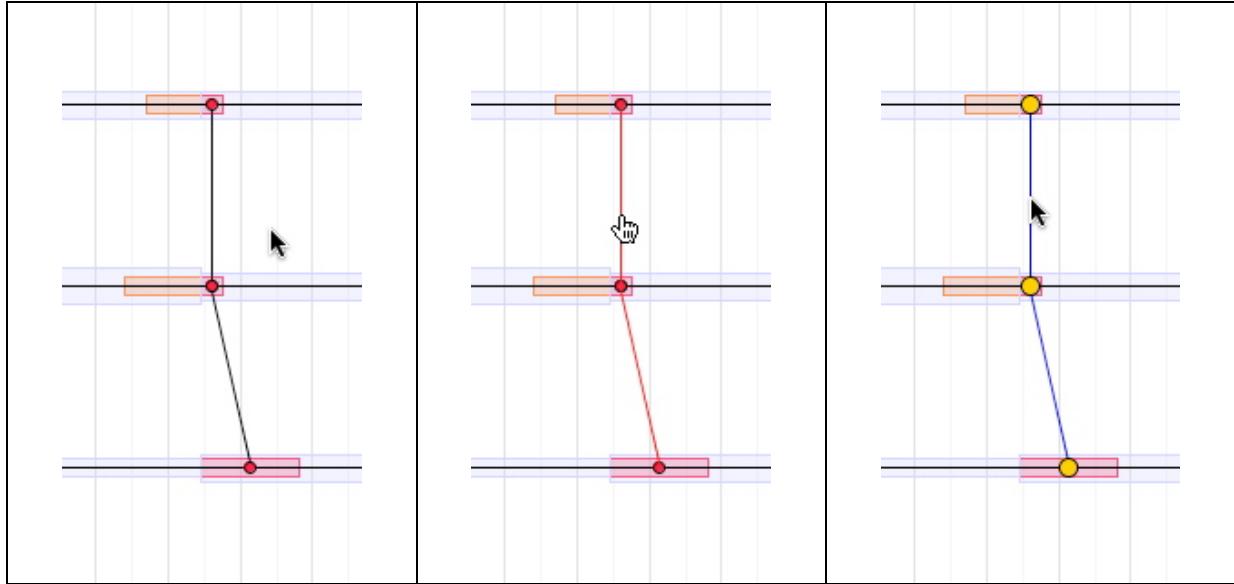
Then, you can create a new alignment by clicking on one unit of each annotator, in whatever order (in our example, we go from top to down).



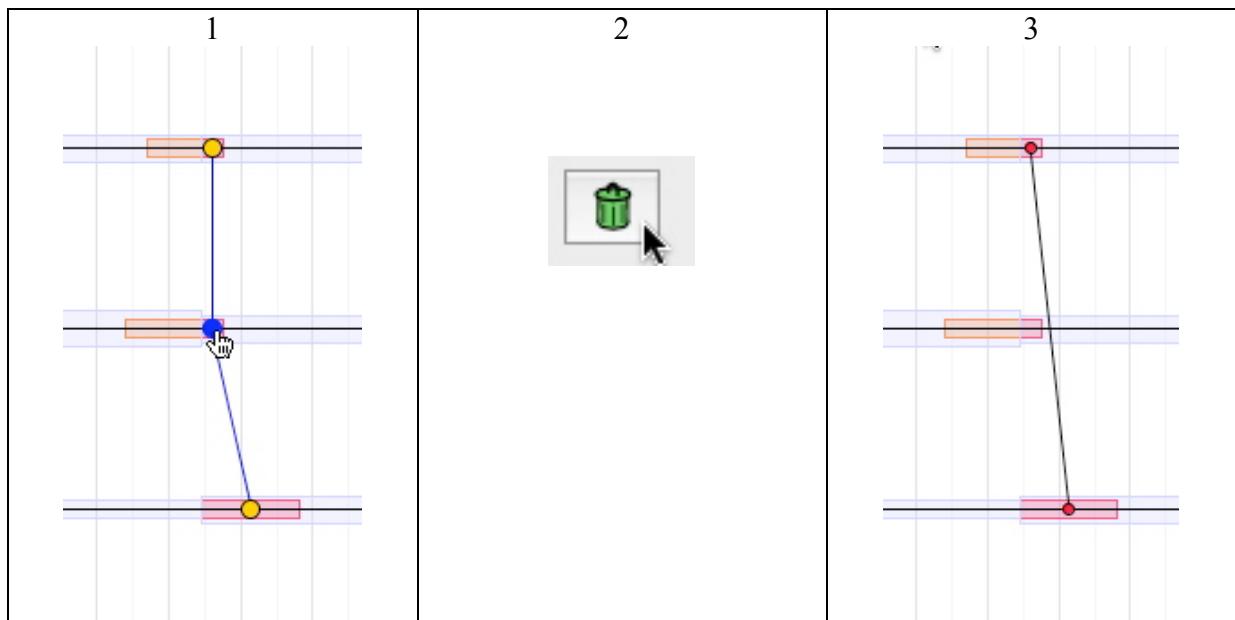
Once you've finished, double click, the alignment is done. It may involve as many annotators as you want (all the three in our example).

### 10.3.2.3 Editing an alignment

To select an alignment, put the mouse over one of its lines, which becomes red, then click, and the lines become blue and the unit disc yellow:



Then, you can remove one unit from the selected alignment. Put the mouse over its yellow disc, click, the disc becomes blue, then click on the trash button. The unit is removed from the alignment, and the alignment is no longer selected.



### 10.4. Agreement measurement

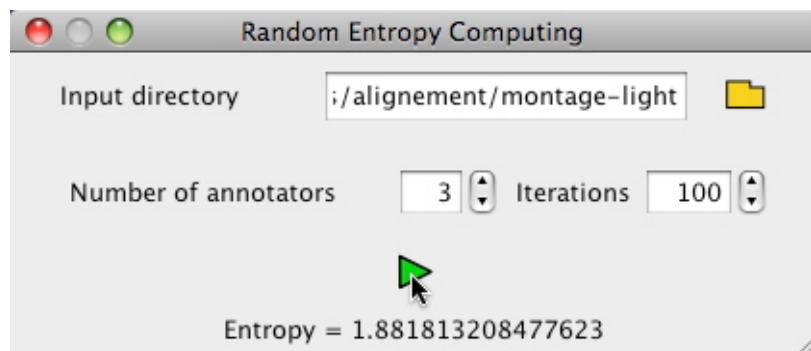
To get an agreement measurement, we need first to compute the « random entropy » of our corpus. To do so, we have to put in a folder all or part of the annotated texts of our corpus.

Then, we can submit this folder to the tool so that it computes the best possible entropy randomly.

The button is as follows :



Then, browse to your folder (« montage-light » in our example below) by clicking on the yellow button, choose the number of annotators to consider (in our example corpus, most of the texts are annotated by 3 annotators, hence the choice of 3 below), and a number of iterations (100 or more is recommended to have the very good random value, but sometimes it takes too much time with certain corpus). Then you can click on the green arrow to launch the process. In our example, some seconds later, we get Entropy=1.88181... This is the random entropy, and it is stored in the system as long as you do not launch again this random entropy computing. Even if you leave and restart glozz, this value will reappear. Hence, it permits you to work on the same corpus without re-computing the random value each time. You will have to do it again only when working with texts from a different corpus.



Now, for any text of our corpus, we can compute its entropy, and, having also the random entropy of the corpus, get the agreement value.

Once the annotated text is loaded, launch the automatic alignment tool as we've seen in section 10.3.1 :



In the same time it makes the alignments, it also computes the entropy value.

Then, in the bottom of the window, we get the three values :

- last random entropy : the random entropy we've computed from a folder
- last computed entropy : the one we've just computed for the current annotated text
- the rate agreement for the current annotated text

Last random entropy: 1,8780	Last computed entropy: 1,0646	Rate agreement: 0,4331
-----------------------------	-------------------------------	------------------------

In our example, with a computed entropy of 1.0646 for the text and a random entropy of the corpus of 1.8780, the rate agreement is 0.4331.

Please refer to (Mathet&Widlöcher 2011) to see how to consider this value.

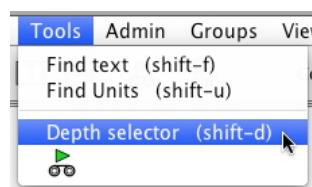
## 11. Additionnal tools

### 11.1. Depth selector

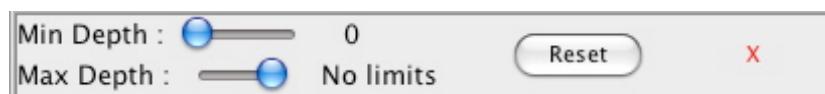
When units are embedding others recursively, it is possible to show only those whose depth belongs to a given range.

**Depth:** when a unit A is covered by another unit B which is coverder by another unit C (covered by no any unit), we say that the depth of A is 2, the depth of B is 1, and the depth of C is 0, and so on...

To launch this tool, use the menu as follows :



You get this additionnal tool on the right side :



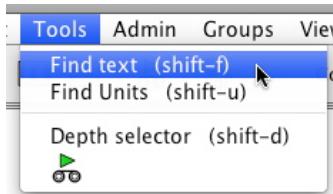
Two slider enable to set respectively the min and the max depth of the units to be shown. A reset button reset min to 0 and max to « no limit ». A red cross enables to close this tool. Let's what happens when playing with the two cursors :

<p>Min Depth : 0 Max Depth : No limits</p>	<p><i>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus.</i> <i>Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed,</i> <i>dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a,</i> <i>semper congue, euismod non, mi. Proin porttitor, orci nec nonummy</i></p>
<p>Min Depth : 1 Max Depth : No limits</p>	<p><i>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus.</i> <i>Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed,</i> <i>dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a,</i> <i>semper congue, euismod non, mi. Proin porttitor, orci nec nonummy</i></p>
<p>Min Depth : 1 Max Depth : 1</p>	<p><i>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus.</i> <i>Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed,</i> <i>dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a,</i> <i>semper congue, euismod non, mi. Proin porttitor, orci nec nonummy</i></p>
<p>Min Depth : 2 Max Depth : No limits</p>	<p><i>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed non risus.</i> <i>Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed,</i> <i>dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a,</i> <i>semper congue, euismod non, mi. Proin porttitor, orci nec nonummy</i></p>

## 11.2. Simple search tools

### 11.2.1 « Find Text » Tool

Besides GlozzQL, it is possible to use a quite simple tool to do some full-text search. Select it in the Tools menu :



This additional tool then appears in the right side of the interface, and comes with two text fields, and two buttons. Enter the word you're looking for in the « search text » field. As long as what you write has at least one occurrence in the text, it appears in GREEN color in this field. It is the case for instance for « diam » with the lorem ipsum text being loaded :

Search text :	diam	Next	Annot...	X
Unit type :				

In the main view, each occurrence appears with a blue background color :

...es diam. M

Then it is possible to jump from one occurrence to the next one clicking on the « Next » button. As soon as the sequence you're entering does not have any occurrence in the text, it appears in RED color (it's no use to go further, use the backspace key to modify the field) :

Search text :	diamu'	Next	Annot...	X
Unit type :				

You can use a second field to specify a unit type the results must belong to. For instance, here, we're looking for the « diam » sequence belonging to a « Verbe » type only :

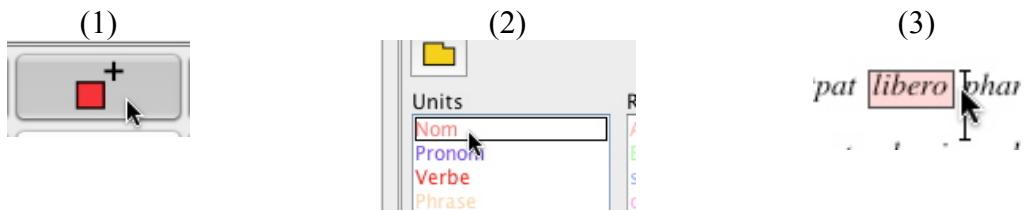
Search text :	diam	Next	Annot...	X
Unit type :	Verbe'			

And we get only one result now :

...ras elementum ultrices diam. Maecenas ligu

Once again, if the current combination of the searched text and the specified unit type does not have any occurrence, it appears in red color. This may happen while you're typing the unit type category, till the category name is fully entered.

This tool comes with an additional feature which enables to automatically create units around each occurrence of a given character sequence. Let's take an example : we want to annotate each « libero » word of this text as a unit of type « Nom ». To do so, we create a first unit of this type :



Then, we select this unit :



Now, back in the search tool, we enter « libero » as searched text. Then, we can of course jump from result to result using the « Next » button, but, at each time, we can choose to click on « Annotate » to automatically create a « Nom » unit around the current occurrence :

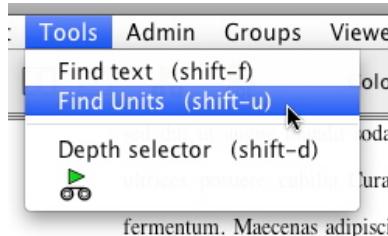


If you want to annotate faster several occurrences, you can even click several times on « Annotate » without clicking on « Next », since this will result in going to the next result and then annotate it.

### 11.2.2 « Find units » tool

This tool works almost the same way as the « Find text » one, just seen above, but is dedicated to looking for units, not text. It is typically something GlozzQL can do, but it was created before, and is a little faster to use when you just want to do so simple requests.

It is launched from the Tools menu :



and it appears as follows in the right side of the interface :



In the « Search Unit » field, enter the type name of searched units. You can additionnaly specify a feature name and its expected value in the two next fields to restrain the search scope. In the example below, where looking for all utterances of units whose type is « Nom », and whose « genre » feature value is « masculin ». There is at least one, since the « Search Unit » field appears in green color :

Search Unit :	Nom
Attribute constr. :	genre
Value constr. :	masculin
<input type="button" value="Next"/> <input type="button" value="First"/> <input type="button" value="X"/>	

We can jump from one result to the other using « Next » button, and go back to the first one using « First » button.

The same request with the « feminin » value for « genre » feature provides no result, as we can see with the « Search Unit » field appearing in red color :

Search Unit :	Nom
Attribute constr. :	genre
Value constr. :	feminin
<input type="button" value="Next"/> <input type="button" value="First"/> <input type="button" value="X"/>	

### 11.3. « TimePlayer »

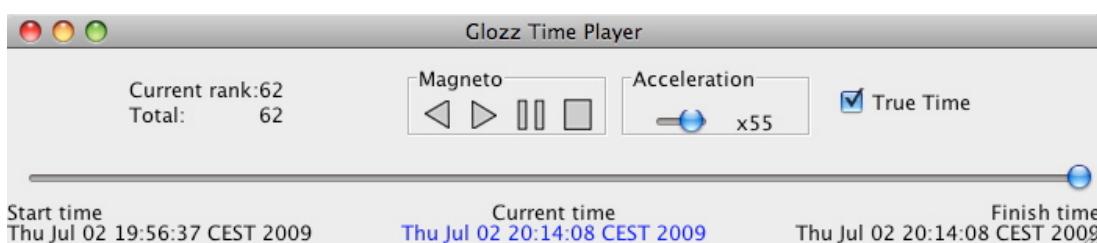
#### 11.3.1 Main principle

Each annotation in Glozz is time-stamped. Hence, it is possible to get an history of the annotation process of a given text. It's the object of the « timePlayer » tool.

To launch it, you can use the button as follows, either in the toolbar or in the « Tool » menu :



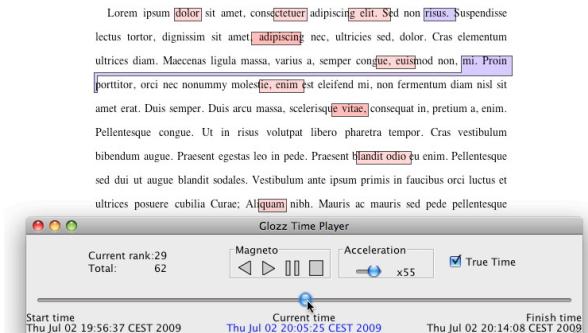
The « TimePlayer » appears as follows :



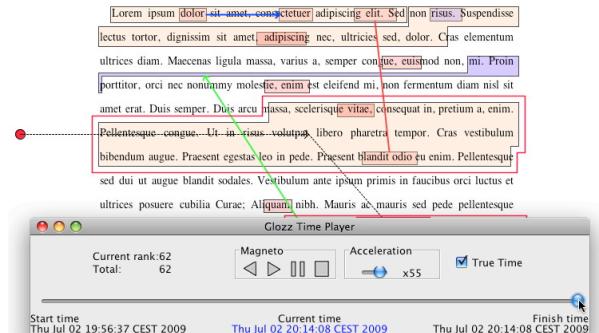
Its main feature is the time line, with a cursor which can be set from first time position in the left (July 2'th at 19:56 in our example), that is to say the time the first annotation of this document was created, to a final position (same day, 20:18 in our example). Just below this cursor line, are shown resp. the first time value, the current time value (the one of the current position of the cursor) and the last time value.

When playing with the cursor, you'll see more or less annotations shown in the main views, from no annotation at all if the cursor is set completely to the left, to all annotations if the

cursor is set completely to the right. Below are two examples for the same annotated text, with two different cursor positions :



Cursor positionned in the center



Cursor positionned to the last position

### 11.3.2 True time option

Two time modes are provided in the time player : « true time » versus « virtual time ». « True time » means that the cursor ratio corresponds to time ratio, or in other words that if the cursor is set to, say, the middle of the line, then the time will be set to the middle of the annotation time, which means the center between the start time and the finish time. This way, you can see exactly the time reality of the annotation process : for instance, some periods of time will have many annotations created, whereas other periods will have almost no new annotations created.

« virtual time » (corresponds to « True time » unchecked in the interface) means that the cursor ratio corresponds to a ratio in terms of number of annotations, chronologically ranked. For instance, if a text is annotated with 100 annotations, the middle position of the live will result in showing the 50 first annotations, and so on... This way, the annotations appear progressively, whatever the cadency they were created.

### 11.3.3 Auto-replay

To play the time automatically, you can use the « magneto » buttons, to start, stop or accelerate the replay.

### 11.3.4 Caution

Before you resume annotation work, make sure the time is set back to the latest position (on the right), in order not to have any new created annotation being strangely hidden...

## Appendices

### Appendice 1 : hashCode algorithm in Java

```
String hash = "";
FileInputStream s = null;
try
{
    s = new FileInputStream(f);
    int length = s.available();
    hash = "" + length+"-";
    long code=1;
    for (int i = 0; i < length; i++)
    {
        int n = s.read();
        if (n!=0)
        {
            code*=n;
            code=code%999999999;
        }
        //System.out.println("code="+code);
    }
    hash+=code;
}
}
```

## **References**

Mathet Y., Widlöcher A., 2011, « Une approche holiste et unifiée de l'alignement et de la mesure d'accord inter-annotateurs », Montpellier, actes de TALN 2011

Mathet Y., Widlöcher A., 2011, « Stratégie d'exploration de corpus multi-annotés avec GlozzQL », Montpellier, short-paper, actes de TALN 2011

Widlöcher A., Mathet Y., 2009, « La plateforme Glozz : environnement d'annotation et d'exploration de corpus », Senlis, actes de TALN 2009