# Dislog: a Logic-Based Language for Processing Discourse

Patrick Saint Dizier

# DISLOG: A logic-based language for processing discourse structures

**Patrick Saint-Dizier**

IRIT-CNRS, 118 route de Narbonne
31062 Toulouse cedex France, stdizier@irit.fr

**Abstract**

In this paper, we present the foundations and the properties of the Dislog language, a logic-based language designed to describe and implement discourse structure analysis. Dislog has the flexibility and the expressiveness of a rule-based system, it offers the possibility to include knowledge and reasoning capabilities and the expression a variety of well-formedness constraints proper to discourse. Dislog is embedded into the <TextCoop> platform that offers an engine with various processing capabilities and a programming environment.

**Keywords:** discourse, logic programming, linguistic modelling

## 1. The Challenges

Discourse analysis is a very challenging task because of the large diversity of discourse structures, the various forms they take in language and the potential knowledge needs for their identification. Rhetorical structure theory (RST) (Mann el al. 1988) is a major attempt to organize investigations in discourse analysis, with the definition of 22 basic structures. Since then, almost 200 relations have been introduced with various aims `http://www.sfu.ca/rst/`. Several approaches, based on corpus analysis with a strong linguistic basis, are of much interest for our aims. Relations are investigated together with their linguistic markers e.g. (Delin 1994), (Marcu 1997), (Miltasaki et ali. 2004), then (Kosseim et al. 2000) for language generation, and (Rossner et al. 1992), and (Saito et al. 2006) with an extensive study on how markers can be quite systematically acquired.

TextCoop is a logic-based platform designed to describe and implement discourse structures and related constraints via an authoring tool. Dislog (Discourse in Logic) is the language designed for writing rules and lexical data. Dislog extends the formalism of Definite Clause Grammars to discourse processing and allows the integration of knowledge and inferences. TextCoop and Dislog tackle the following foundational and engineering problems:

- taking into account of the diversity of discourse structures: generic (e.g. illustration, elaboration) as well as domain oriented (e.g. title-instructions in procedures),
- introduction, for easy tests and updates, of a declarative and modular language via rules. Our approach is based on (1) basic discourse structures, (2) selective binding rules to bind basic structures into larger units, (3) repair rules and (4) various classes of constraints on the way basic structures can be combined,
- introduction of accurate specifications of rule execution modes (e.g. order, concurrency, left-to-right or right-to-left, etc.), in order to optimally process structures,
- taking into account of the specification and binding of complex structures, e.g. multi-nucleus-satellite constructions as often found in domain dependent constructions (e.g. title-prerequisites-instructions in procedures), or cases where satellites are merged into their nucleus (dislocation),
- integration in rules of various forms of knowledge and inferences e.g. to compute attribute values or to resolve relation identification and scope, or ambiguities between various relations.
- development of an authoring tool to implement discourse relation rules and lexical resources. Note that in general discourse analysis rules are relatively re-usable over domains because markers are often domain-independent.
- finally, production of various forms of output representations (XML tags, dependencies).

TextCoop is currently used to process various kinds of procedural texts, industrial requirements and regulations, news texts and didactic texts. It is used in projects dedicated to health and ecology safety analysis in industrial procedures (the LELIE project) and in opinion analysis, in particular for argument extraction. TextCoop is in an early stage of development, it offers different functions than well-known platforms such as Gate or Linguastream.

## 2. The <TextCoop> platform and the Dislog language

### 2.1. The context

There are at the moment a few well-know and widely used language processing environments. They are essentially used for sentence processing, not for discourse analysis. The reasons are essentially that the sentence level and its substructures are the crucial level of analysis for a large number of applications such as information extraction, opinion analysis based on noun modifiers or machine translation. Discourse analysis turns out to be not so critical for these applications. However, applications such as summarization or question-answering do require an intensive discourse analysis level.

Dedicated to sentence processing, let us note the GATE platform (http://gate.ac.uk/) which is widely used and the Linguastream (http://www.linguastream.org) system which is based on a component architecture, making the system

really flexible. Except for some specific features for simple aspects of discourse processing, none of these platforms allow the specifications of rules for an extensive discourse analysis nor the introduction of reasoning aspects, which is essential to introduce pragmatic considerations into discourse processing. GATE is used e.g. for semantic annotation, corpus construction, knowledge acquisition and information extraction, summarization, and investigations around the semantic web. It also includes research on audio visual and language connections. Linguastream has components to mainly deal with part of speech and syntactic analysis. It also handles several types of semantic data with a modular approach. It is widely used for corpus analysis. The GETARUNS system (http://project.cgm.unive.it/getaruns.html), based on the LFG grammar approach, has some capabilities to process simple forms of discourse structures and argumentation analysis. Finally, (Marcu 2000) developed a discourse analyzer for the purpose of automatic summarization. This system is based on the RST assumptions which are not always met in texts, as developed in the section below.

### 2.2. Some linguistic considerations

Most works dedicated to discourse analysis have to deal with the triad: discourse function identification, delimitation of its textual structure (boundaries of the discourse unit) and structure binding. By function we mean a nucleus or a satellite of a rhetorical relation, e.g. an illustration, an illustrated expression, an elaboration, or the elaborated expression, a conditional expression, a goal expression, etc. Functions are realized by textual structures which need to be accurately delimited. Functions are not stand alone: they must be bound based on the nucleus-satellite or nucleus-nucleus principle.

### 2.3. Some foundational principles of <TextCoop>

The necessity of a modular approach, where each aspect of discourse analysis is dealt with accurately and independently in a module, while keeping open all the possibilities of interaction, if not concurrency, between modules has lead us to consider some simple elements of the model of Generative Syntax (a good synthesis is given in (Lasnik et al. 1988)). As shall be seen below, we introduce:

- productive principles, which have a high level of abstraction, which are linguistically sound, but which may be too powerful,
- restrictive principles, which limit the power of the first in particular on the basis of well-formedness constraints.

Another foundational feature is an integrated view of markers used to identify discourse functions, merging lexical objects with morphological functions, typography and punctuation, syntactic constructs, semantic features and inferential patterns that capture various forms of knowledge (domain, lexical, textual). <TextCoop> is the first platform that offers this view within a logic-based approach. If machine learning is a possible approach for sentence processing, where interesting results have emerged, it seems not to be so successful for discourse analysis (e.g. Carlson et ali. 2001), (Saaba et al 2008), the Annodis project). This is due to two main factors: (1) the difficulty to annotate discourse functions in texts and the high level of disagreement between annotators and (2) the large non-determinism of discourse structure recognition where markers are often immerged in long spans of text of no or little interest. For these reasons, we adopted a rule-based approach. Rules are hand coded, based on corpus analysis using bootstrapping tools.
Dislog rules basically implement the productive principles. They are composed of three main parts:

1. A discourse function identification structure, which basically has the form of a rule or of a pattern,
2. A set of calls to inferential forms using various types of knowledge, these forms are part of the identification structure, they may contribute to solving ambiguities, they may also be involved in the computation of the resulting representation or they may lead to restrictions.
3. A structure that represents the result of the analysis: it can be a simple XML structure, or any other structure a priori such as an element of a graph or a dependency structure. More complex representations, e.g. based on primitives, can be computed using a rich semantic lexicon. This is of much interest since our analysis is oriented towards a conceptual analysis of discourse.

### 2.4. The structure of Dislog rules

Let us now introduce in more depth the structure of Dislog rules. Dislog follows the principles of logic-based grammars as implemented three decades ago in a series of formalisms, among which, most notably: Definite Clause Grammars (Pereira and Warren 1981), Metamorphosis Grammars (Colmerauer 1978) and Extraposition Grammars (Pereira 1981). These formalisms were all designed for sentence parsing with an implementation in Prolog via a meta-interpreter or a direct translation into Prolog (Saint-Dizier 1994). The last two formalisms include a device to deal with long distance dependencies.
Dislog adapts and extends these grammar formalisms to discourse processing, it also extends the regular expression format which is often used as a basis in language processing tools. The rule system of Dislog is viewed as a set of productive principles.
A rule in Dislog has the following general form, which is globally quite close to Definite Clause Grammars in its spirit:
L(Representation) → R, {P}.
where:

1. L is a non-terminal symbol.
2. Representation is the representation resulting from the analysis, it is in general an XML structure with attributes that annotates the original text. It can also be a partial dependency structure or a more formal representation. The computation of the representation is typical of logic-based grammars and use the power of logic variables of logic programming.

3. R is a sequence of symbols as described below, and

4. P is a set of predicates and functions implemented in Prolog that realize the various computations and controls, and that allow the inclusion of inference and knowledge into rules.

R is a finite sequence of the following elements:

- terminal symbols that represent words, expressions, punctuations, various existing html or XML tags. They are included between square brackets,

- preterminal symbols: are symbols which are derived directly into terminal elements. These are used to capture various forms of generalizations, facilitating rule authoring and update. Symbols can be associated with a type feature structure that encodes a variety of aspects of those symbols, from morphology to semantics,

- non-terminal symbols, which can also be associated with type feature structures. These symbols refer to 'local grammars', i.e. grammars that encode specific syntactic constructions such as temporal expressions or domain specific constructs. Non-terminal symbols do not include discourse structure symbols: Dislog rules cannot call each other, this feature being dealt with by the selective binding principle, which includes additional controls. A rule in Dislog thus basically encodes the recognition of a discourse function taken in isolation, i.e. an elementary discourse unit.

- optionality and iterativity markers over non-terminal and preterminal symbols, as in regular expressions,

- gaps, which are symbols that stand for a finite sequence of words of no present interest for the rule which must be skipped. A gap can appear only between terminal, preterminal or non-terminal symbols. Dislog offers the possibility to specify in a gap a list of elements which must not be skipped: when such an element is found before the termination of the gap, then the gap fails.

- a few meta-predicates to facilitate rule authoring.

Symbols may have any number of arguments. However, in our current version, to facilitate the implementation of the meta-interpreter and improve its efficiency, the recommended form is:
identifier(Representation, Type feature structure).
where Representation is the symbol's representation. In Prolog format, a difference list (E,S) is added at the end of the symbol:

```
identifier(R, TFS, E,S)
```

A few examples in Dislog format are given at the end of this document. Rules in external format can be found at (Bourse and Saint-Dizier, LREC 2012).
Similarly to DCGs and to Prolog clause systems, it is possible and often necessary to have several rules to describe the different realizations of a given discourse function. These all have the same identifier $L$, as it would be the case e.g. for NPs or PPs. A set of rules with the same identifier is called a cluster of rules. Rule clusters are executed sequentially by the <TextCoop> engine following an order given in a **cascade**.

### 2.5. Dislog advanced features

#### 2.5.1. Selective binding rules

Selective binding rules allow to link two or more already identified discourse functions. The objective is e.g. to bind a nucleus with a satellite (e.g. an argument conclusion with its support) or with another nucleus (e.g. concessive or parallel structures). Selective binding rules can be used for other purposes than implementing rhetorical relations. These can be used more generally to bind structures whose rhetorical status is not so straightforward, in particular in some application domains. For example, in procedural discourse, they can be used to link a title with the set of instructions, prerequisites and warnings that realize the goal expressed by this title.
From a syntactic point of view, selective binding rules are expressed using the Dislog language formalism. Selective binding rules is the means offered by Dislog to construct hierarchical discourse structures from the elementary ones identified by the rule system. Different situations occur that make binding rules more complex than any system of rules used for sentence processing, in particular (examples are given in section 2.6):

- discourse structures may be embedded to a high degree, with partial overlaps,

- others may be chained (a satellite is a nucleus for another relation),

- nucleus and related satellites may be non-adjacent,

- nucleus may be linked to several satellites of different types,

- some satellites may be embedded into their nucleus.

Selective binding rules allow the binding of:

1. two adjacent structures, in general a nucleus and a satellite, or another nucleus.

2. two or more non-adjacent structures, which may be separated by various elements (e.g. causes and consequences, conclusion and supports may be separated by various considerations). However limits must be imposed on the 'textual distance' between units.

To limit the textual distance between argument units, we introduce the notion of **bounding node**, which is also a notion used in sentence formal syntax to restrict the way long-distance dependencies can be established (Lasnik et al. 1988). Bounding nodes are also defined in terms of barriers in generative syntax. In our case, the constraint is that a gap must not go over a bounding node. This allows to restrict the distance between the constituents which are bound. For example, we consider that an argument conclusion and support must be both in the same paragraph, therefore, the node 'paragraph' is a bounding node.

This declaration is taken into account by the <TextCoop> engine in a transparent way, and interpreted as an active constraint which must be valid throughout the whole parsing process. The situation is however more complex than in sentence syntax. Indeed, bounding nodes in discourse depend on the structure being processed. For example, in the case of procedural discourse, a warning can be bound to one or more instructions which are in the same subgoal structure. Therefore, the bounding node must be the subgoal node, which may be much larger than a paragraph. Bounding nodes are declared as follows in Dislog:

```
boundingNode(paragraph, argument).
```

#### 2.5.2. Repair rules

Although relatively unusual, annotation errors may occur. This is in particular the case when (1) a rule has a fuzzy or ambiguous ending condition w.r.t. the text being processed or (2) when rules of different discourse functions overlap, leading to closing tags that may not be correctly inserted. In argument recognition, we have indeed some forms of competition between a conclusion and its support which share common linguistic markers. For example, when there are several causal connectors in a sentence the beginning of a support is ambiguous since most supports are introduced by a connector. In addition to using concurrent processing strategies, repair rules can resolve errors efficiently.
The most frequent situation is the following:
<a>, ...<b> < /a>, ... < /b>
which must be rewritten into:
<a>, ... < /a>, ... <b>, ... < /b> .
This is realized by the following rule:

```
correction([<A> G1 </A> G2 <B> G3 </B>]) -->
     [<A>], gap(G1),[<B>],gap(G2),
        [</A>],  gap(G3), [</B>].
```

#### 2.5.3. Rule concurrency management

The current <TextCoop> engine is close to the Prolog execution schema. However, to properly manage rule execution but also the properties of discourse structures and the way they are usually organized, we introduce additional constraints, which are, for most of them, borrowed from sentence syntax.
Within a cluster of rules, the execution order is the rule reading order, from the first to the last one. Then, elementary discourse functions are first identified and then bound to others to form larger units, via selective binding rules. Following the principle that a text unit has one and only one discourse function (but may be bound to several other structures via several rhetorical relations) and because rules can be ambiguous from one cluster to another, the order in which rule clusters are executed is a crucial parameter. To handle this problem, Dislog requires that rule clusters are executed in a precise, predefined order, implemented in a **cascade of clusters of rules**.
For example if, in a procedure, we want first titles, then prerequisites and then instructions to be identified, the following constraint must be specified:

```
title < prerequisite < instruction.
```

Since titles have almost the same structure than instructions, but with additional features (bold font, html specific tags, etc.), this prevents titles from being erroneously identified as instructions.
In our engine, there is no backtracking between clusters. In relation with this notion of cascade, it is possible to declare **'closed zones'**, e.g.:

```
closed_zone([title]).
```

indicates that the textual span recognized as a title must not be considered again to recognize other functions within or over it (via a gap).

#### 2.5.4. Structural constraints

Let us now consider basic structural principles, which are very common in language syntax. This allows us to contrast the notion of consistuency with the notion of discourse relation. Consistuency is basically a part-of relation applied to language structures (nouns are part of NPs) while discourse is basically relational. Let us introduce here dominance and precedence constraints, these notions being valid as far as trees of discourse structures can be constructed, which is in fact the most frequent situation. Discourse abound in various types of constraints, which may be domain dependent. Dislog is open to the specification of a number of such structural constraints.
**Dominance constraints** can be stated as follows:

```
dom(instruction, condition).
```

This constraint states that a conditional expression is always dominated by an instruction, i.e. the condition XML tags are strictly within the boundaries of an instruction XML tags. This means that a condition must always be part of an instruction, not in a discourse relation with an instruction. In that case, there is no discourse link between a condition and an instruction, the implicit structure being consistuency: a condition is a constituent, or a part of, an instruction.
Similarly, **non-dominance constraints** can be stated to ensure that two discourse functions appear in different branches of the discourse representation, e.g.:

```
not_dom(instruction, warning).
```

which states that an instruction cannot dominate a warning.
Finally, **precedence constraints** may be stated. We only consider here the case of immediate linear precedence, for example:

```
prec(elaborated, elaboration).
```

indicates that an elaboration must follow what is elaborated. This is a useful constraint for the cases where the nucleus must necessarily precede its satellite: it contributes to the efficiency of the selective binding mechanism and resolves some recognition ambiguities.

### 2.6. Introducing reasoning aspects into discourse analysis

Discourse relation identification often require some forms of knowledge and reasoning. This is in particular the case

to resolve ambiguities in relation identification when there are several candidates or to clearly identify the text span at stake. While some situations are extremely difficult to resolve, others can be processed e.g. via lexical inference or reasoning over ontological knowledge. Dislog allows the introduction of reasoning, and the <TextCoop> platform allows the integration of knowledge and functions to access it and reason about it.

This problem is very vast and largely open, with exploratory studies e.g. reported in (Van Dijk 1980), (Kintsch 1988), and more recently some debates reported in (`http://www.discourses.org/UnpublishedArticles/SpecDis&Know.htm`) .

Let us give here a simple motivational example. The utterance (found in our corpus):

*... red fruit tart (strawberries, raspberries) are made ...*

contains a structure: (*strawberries, raspberries*) which is ambiguous in terms of discourse functions: it can be an elaboration or an illustration, furthermore the identification of its nucleus is ambiguous:

*red fruit tart, red fruit* ?

A straightforward access to an ontology of fruits tells us that those berries are red fruits, therefore:

- the unit *strawberries, raspberries* is interpreted as an illustration, since no new information is given (otherwise it would have been an elaboration)
- its nucleus is the '*red fruit*' unit only,
- and it should be noted that these two constituents, which must be bound, are not adjacent.

The relation between an argument conclusion and its support may not necessarily straightforward and may involve various types of domain and common-sense knowledge:

*do not park your car at night near this bar: it may cost you fortunes.*

*Women standards of living have progressed in Nepal: we now see long lines of young girls early morning with their school bags.* (Nepali Times).

In this latter example, *school bag* means going to school, then *school* means education, which in turn means better conditions, for women in this case.

### 2.7. Processing complex constructions: the case of Cleft constructions

As in any language situation, there are complex situations where discourse segments that contribute to form larger units, which are not clearly delimited, may overlap, be shared by several discourse relations, etc. Similarly to syntax, we identified in relatively 'free style' texts phenomena similar to quasi-scrambling situations, free-structure ordering and cleft constructions.

From a processing point of view, the <TextCoop> engine attempts to recognize the embedded structure first, then, if no unique text segment can be found for the embedding structure (standard case), it non-deterministically decomposes the rules describing the embedding structure one after the other, following the above constraints, and attempts to recognize it 'around' the embedded one.

As an example, we observed in our corpora quasi-scrambling situations, a simple case being the illustration relation. Consider again the example above, which can also be written as follows (in French):

*strawberries are red fruits similarly to raspberries, for example.*

where the enumeration itself is subject to dislocation.

## 3. The <TextCoop> engine

Let us now give some details about the way the <TextCoop> engine runs. The engine and its environment are implemented in SWI Prolog, using the standard Prolog syntax without using any libraries to guarantee readability, ease of update and portability. Since this is quite a complex implementation, we simply survey here the elements which are crucial for our current purpose. The principle is that the declarative character of constraints and structure processing and building is preserved in the system. The engine, implemented in Prolog, interprets them at the appropriate control points. The <TextCoop> engine code will be shortly available, under either the CECILL license (French GPL) or a low cost traditional license, together with a programming environment for rules and linguistic resources (for French and English).

The constraints advocated above remain as given in the examples below, these are directly consulted by the meta-interpreter to realize the relevant controls. The engine follows the cascade specification for the execution of rule clusters. For each cluster, rules are activated in their reading order, one after the other. Backtracking manages rule failures. If a rule in a rule cluster succeeds on a given text span, then the other possibilities for that cluster are not considered (but rules of other clusters may be considered in a later stage of the cascade).

A priori, the text is processed via a left to right strategy. In a cluster of rules, rules are executed sequentially, however, if a rule starts with an early symbol (e.g. a determiner), it is activated before another rule that starts on a later symbol (e.g. the noun it quantifies). <TextCoop> also offers a right to left strategy for rules where the most relevant markers are to the right of the rule, in order to limit backtracking. For the two types of readings, the system is tuned to recognize the smallest text span that satisfies the rule structure.

It processes raw text, html or XML texts. A priori, the initial XML structure is preserved.

### 3.1. System performances and discussion

Let us now analyze the performances of <TextCoop> with respect to relevant linguistic dimensions, and contrast these with performances of parsers dedicated to sentence processing. More details are given in (Saint-Dizier, 2012).

#### 3.1.1. General results

The <TextCoop> engine and related data are implemented in SWI Prolog which runs on a number on environments (Windows, Linux, Apple). Our implementation can support a multi-threaded approach, which has been tested with the <TextCoop> engine embedded into a Java environment. This is useful for example for 'parallel' processing on several machines or to distribute e.g. lexical data, grammars and domain knowledge on various machines.

The <TextCoop> engine has been relatively optimized and some recommendations for writing rules have been produced in order to allow for a reasonable efficiency.

#### 3.1.2. Lexical issues

An important feature of discourse structure recognition is that the lexical resources which are needed are quite often generic. This means that the system can be deployed on any application domain without any major lexical changes and update (Bourse and Saint-Dizier, LREC 2012). In total, the average size of the required lexical resources (number of rules being fixed) for discourse processing for an application such as procedural text parsing on a given domain is around 900 words, which is very small compared to what is necessary to process the structure of sentences for the same domain. Results below are given fro French. Results for English are not very different.

The following figures give the system performances depending on the lexicon size. Lexicon sizes correspond to comprehensive lexicons for a given domain (e.g. 400 corresponds to the cooking domain, the case with 180 lexical entries is a toy system).

| lexicon size (in no. of words) | Mb of text/hour |
|---|---|
| 180 | 39 |
| 400 | 27 |
| 900 | 20 |
| 1400 | 18 |
| 2000 | 17 |

Fig. 1 Impact of lexicon size

These results are somewhat difficult to precisely analyze, since they depend on the number of words by syntactic category, the way they are coded and the order in which they are listed in the lexicon (in relation with the Prolog strategy). In order to limit the complexity related to morphological analysis, a crucial aspect for Romance languages, a preliminary tagging process has been carried out to limit backtracking. The way lexical resources are used in rules is also a parameter which is difficult to precisely analyse.

Globally, reducing the size of the lexicon to those elements which are really needed for the application allows for a certain increase in the system performances.

#### 3.1.3. Issues related to the rule system size and complexity

Two parameters related to the rule system are investigated here: how much the number of rules and the rule size impact the efficiency.

The results obtained concerning the number of rules are the following:

| number of rules | Mb of text/hour |
|---|---|
| 20 | 29 |
| 40 | 23 |
| 70 | 19 |
| 90 | 18 |

Fig. 2 Impact of number of rules

As can be noted, increasing the number of rules has a moderate impact on performances, one of the reasons is that the most prototypical rules are executed first. Rules have here an average complexity: 4 symbols and a gap in average, and an average of 8 rules per cluster. Lexical size here is fixed (500 entries). 20 rules is a very small system while 80 to 120 rules is a standard size for an application. The results we obtain are difficult to accurately analyze: besides rule ordering considerations, results depend on the distribution of rules per cluster and the form of the rules. For example, the presence of non-ambiguous linguistic markers at the beginning of a rule enhances rule selection, and therefore improves efficiency. Constraints such as those presented above are also very costly since they are checked at each step of the parsing process for the structures at stake. Selective binding rules have little impact on efficiency: their first symbol being an XML tag backtracking occurs at an early stage of the rule traversal.

Let us now consider rule size, which is obviously an important feature:

| rule complexity (symbols per rule) | Mb of text/h |
|---|---|
| 3 | 30 |
| 4 | 23 |
| 5 | 20 |
| 7 | 18 |

Fig. 3 Impact of rule size

With the number of rules and the size of the lexicon being kept fixed, we note that the rule size has a moderate impact on performances, slightly higher than the number of rules. This may be explained by the fact that the symbols starting the rules are in a number of cases sufficiently well differentiated to provoke early backtracking if the rule is not the one that must be selected. However, the number of lexical entries of these symbols may have an important impact. If the symbol is a specific type of connector or if it is a noun or a verb, this may entail efficiency differences, difficult however to evaluate at our level. Finally, note that rules have in general between 4 and 6 symbols including gaps.

### 3.2. The <TextCoop> environment

The <TextCoop> environment is in a very early stage of development: many more experiments are needed before reaching a stable analysis of the needs. It includes tools for rules (syntax checking, but also e.g. controlling possible overlaps between rules, bootstrapping on corpora to induce rules) and for developing the necessary lexical resources. Accessing already defined and formatted resources is of much interest for authors. We have already designed the following sets of resources, for French and English (Bourse and Saint-Dizier, LREC 2012):

- lists of connectors, organized by general types: temporal, causal, concession, etc.
- list of specific terms which can appear in a number of discourse functions, e.g.: terms specific of illustration, summarization, reformulation, etc.
- lists of verbs organized by semantic classes, close to those found in WordNet, that we have adapted or refined for discourse analysis, e.g. with a focus e.g. on propositional attitude verbs, report verbs, (Wierzbicka 1987), etc.

- list of terms with positive or negative polarity, essentially adjectives, but also some nouns and verbs, this is useful in particular to evaluate the strength of arguments,
- local grammars for e.g.: temporal expressions, expression of quantity, etc.
- some already defined modules of discourse function rules to recognize general purpose discourse functions such as illustration, definition, reformulation, goal and condition.
- some predefined functions and predicates to access knowledge and control features (e.g. subsumption),
- morphosyntactic tagging functions,
- some basic utilities for integrating knowledge (e.g. ontologies) into the environment.

## 4. Perspectives

In this article, we have first presented the <TextCoop> platform and the Dislog language, designed for discourse analysis. <TextCoop> is based on a cooperation between grammar theory and knowledge and reasoning, which allows the introduction of knowledge and pragmatic factors to identify and properly bound discourse structures. From that point of view this platform offers several original features.

Quite a comprehensive set of rules is given in (Bourse and Saint-Dizier, LREC 2012) concerning discourse structures which aere related to explanation. Other examples, related to argumentation can be found in (Fontan et al. 2008) and (Saint-Dizier 2012).

From a software point of view, we plan to distribute the kernel of <TextCoop> (CECILL or free traditional licence) when sufficiently tested. A user manual will come with the Prolog code with examples in Dislog format and language resources. Probably, and more conveniently, the system will also be available as a web service with an adequate interface and input-output facilities, where users can upload texts and see the tagged texts. We also plan to make available a large corpus of arguments tagged by the system.

At the moment, we develop two main application frameworks: (1) argument extraction in opinion analysis, and (2) risk analysis and prevention as these can be detected from procedures. Argument extraction in opinion analysis, applied to customer opinions, aims at identifying the reasons why customers are happy or unhappy with a certain product or brand. Arguments may be explicit, introduced by a causal marker, or they may be incorporated into an evaluative expression such as an adjective or an adverbial construct.

We also initiated the LELIE project:
http://www.irit.fr/recherches/ILPL/lelie/accueil.html
aimed at analysing and preventing risks (e.g. health, ecology) from procedure analysis, in production and maintenance situations. This project requires an extensive discourse processing. The goal is to make sure that a set of procedures dedicated to a given task contain all the necessary safety advice and warnings as stipulated by norms, regulations, or business rules. Related to this project, we are investigating from a language point of view a number of aspects of requirement technology in order to improve their language structure for subsequent treatments.

## 5. Acknowledgements

This project is supported by the French ANR project LELIE and partly by an IFCPAR Indo-French project. We are also very grateful to a number of colleagues for discussions about this work, including David Roussel, Lionel Fontan, Estelle Delpech and Sarah Bourse and three anonymous reviewers.

## 6. References

Bal, B.K., Saint-Dizier, P., 2010. Towards Building Annotated Resources for Analyzing Opinions and Argumentation in News Editorial, LREC, Malta.

Carberry, S., 1990. Plan Recognition in natural language dialogue, Cambridge university Press, MIT Press.

Carlson, L., Marcu, D., Okurowski, M.E., 2001. Building a Discourse- Tagged Corpus in the Framework of Rhetorical Structure Theory. In Proceedings of the 2nd SIGdial Workshop on Discourse and Dialog, Aalborg.

Colmerauer, A., 1978. Metamorphosis Grammars, in Natural language understanding by computers, L. Bolc (ed.), LNCS no. 63, Springer verlag.

Delin, J., Hartley, A., Paris, C., Scott, D., Vander Linden, K., 1994. *Expressing Procedural Relationships in Multilingual Instructions*, Proceedings of the Seventh International Workshop on Natural Language Generation, pp. 61-70, Maine, USA.

Di Eugenio, B. and Webber, B.L., 1996. Pragmatic Overloading in Natural Language Instructions, *International Journal of Expert Systems*.

Fontan, L., Saint-Dizier, P., 2008. Analyzing the explanation structure of procedural texts: dealing with Advices and Warnings, International Symposium on Text Semantics (STEP 2008), Venise, , Johan Bos (Eds.).

Grosz, B., Sidner, C., 1986. Attention, intention and the structure of discourse, Computational Linguistics 12(3).

Kintsch, W., 1988. *The Role of Knowledge in Discourse Comprehension: A Construction-Integration Model*, Psychological Review, vol 95-2.

Kosseim, L., Lapalme, G., 2000. *Choosing Rhetorical Structures to Plan Instructional Texts*, Computational Intelligence, Blackwell, Boston.

Lasnik, H., Uriagereka, J., 1988. *A Course in GB syntax*, MIT Press.

Mann, W., Thompson, S., 1988. Rhetorical Structure Theory: Towards a Functional Theory of Text Organisation, *TEXT* 8 (3) pp. 243-281.

Mann, W., Thompson, S.A. (eds), 1992. Discourse Description: diverse linguistic analyses of a fund raising text, John Benjamins.

Marcu, D., 1997. The Rhetorical Parsing of Natural Language Texts, ACL'97.

Marcu, D., 2000. *The Theory and Practice of Discourse Parsing and Summarization*, MIT Press.

```
warning → it is (adv) important... to/that) gap verb gap Mfin(end)

forme(c-avt-eng, E, S, [
expr(EXP,itis,E,E2), gap([],[adj, ],E2,E3,Saute1),
adj(ADJ,imperatif,E3,E4), gap([], [verb, ], E4,E5,Saute2),
verb(V,[action,impinf],E5,E6),
gap([], [mfin, ], E6,E7,Saute3), mfin(MFIN, ,E7,S)], [],
[ <concl-avt> , EXP, Saute1, ADJ, Saute2, <verb type=actionimpinf>,
V, < /verb>, Saute3, < /concl-avt>, MFIN]).

warning → avoid gap Mfin(end)

forme(c-avt-eng, E, S, [
verb(V,[eviter,impinf],E,E3),
gap([], [mfin, ], E3,E4,Saute2), mfin(MFIN, ,E4,S)], [],
[ <concl-avt> , MDEB, Saute1, V, Saute2, < /concl-avt>, MFIN]).

warning → make sure not to /to (never) gap Mfin(end)

forme(c-avt-eng, E, S, [ expr(EXP2,ensure,E,E2), prep([to], ,E2,E4),
opt(neg(Neg, ,E4,E5)), verb(V,[action,impinf],E5,E6),
gap([], [mfin, ], E6,E7,Saute2), mfin(MFIN, ,E7,S)], [],
[ <concl-avt> , EXP2,[to],Neg, V, Saute2, < /concl-avt>, MFIN]).

warning → ensure not to gap Mfin(end)

forme(c-avt-eng, E, S, [ expr(EXP2,ensure,E,E2),
neg(Neg, ,E2,E3),prep([to], ,E3,E5),
verb(V,[ ,impinf],E5,E6),
gap([], [mfin, ], E6,E7,Saute2), mfin(MFIN, ,E7,S)], [],
[ <concl-avt> , EXP2,Neg,[to], V, Saute2, < /concl-avt>, MFIN]).
```

Figure 1: A few rules for warning conclusions in ready to run mode, see also more data in (Bourse and Saint-Dizier, LREC 2012)


Marcu, D., 2002. An unsupervised approach to recognizing Discourse relations, ACL'02.

Miltasaki, E., Prasad, R., Joshi, A., Webber, B., 2004. Annotating Discourse Connectives and Their Arguments, proceedings of new frontiers in NLP.

Pereira, F., 1981. Extraposition Grammars, *Computational Linguistics*, vol. 9-4.

Pereira, F., Warren, D., 1980. Definite Clause Grammars for Language Analysis, *Artificial Intelligence* vol. 13-3.

Rosner, D., Stede, M., 1992. *Customizing RST for the Automatic Production of Technical Manuals*, in R. Dale, E. Hovy, D. Rosner and O. Stock eds., Aspects of Automated Natural Language Generation, Lecture Notes in Artificial Intelligence, pp. 199-214, Springler-Verlag.

Saaba A., Sawamura, H., 2008. Argument Mining Using Highly Structured Argument Repertoire, *proceedings EDM08*, Niigata.

Saito, M., Yamamoto, K., Sekine, S., 2006. Using Phrasal Patterns to Identify Discourse Relations, ACL'06.

Saint-Dizier, P., 1994. *Advanced Logic programming for language processing*, Academic Press.

Saint-Dizier, P., 2012. Processing Natural Language Arguments with the <TextCoop> Platform, *Journal of Argumentation and Computation*.

Takechi, M., Tokunaga, T., Matsumoto, Y., Tanaka, H., 2003. *Feature Selection in Categorizing Procedural Expressions*, The Sixth International Workshop on Information Retrieval with Asian Languages (IRAL2003), pp.49-56.

Van Dijk, T.A., 1980. *Macrostructures*, Hillsdale, NJ: Lawrence Erlbaum Associates.

Webber, B., 2004. D-LTAG: extending lexicalized TAGs to Discourse, Cognitive Science 28, pp. 751-779, Elsevier.

Wierzbicka, A., 1987. English Speech Act Verbs, Academic Press.