The <TextCoop> Platform: Analyzing arguments in procedural texts

Thomas De Filippo, Laureline Marsal, Patrick Saint-Dizier [1]

1 Introduction and Aims

<TextCoop> is an environment for text semantics analysis. In a first experimental stage, it is dedicated to procedure processing for testing and evaluation. Processing procedures has obviously major applications. Besides instructions, procedures also contain a number of forms of explanation, in particular warnings and advice. Therefore, our platform can also be used as a tool for argument mining.

The goal of this document is to show the current possibilities of the software, version V2, in Java, after a proof of concept realized two years ago in Perl, based on scripts (Delpech et al. 2008).

Procedural texts are organized sets of instructions, they may also be sets of advice, as in social behavior texts. In our perspective, procedural texts range from apparently simple cooking recipes to large maintenance manuals. They also include documents as diverse as teaching texts, medical notices, social behavior recommendations, directions for use, assembly notices, do-it-yourself notices, itinerary guides, advice texts, savoir-faire guides etc. Even if procedural texts adhere more or less to a number of structural criteria, which may depend on the author's writing abilities and on traditions associated with a given domain, we observed a very large variety of realisations, which makes parsing such texts quite challenging.

Procedural texts explain how to realize a certain goal by means of actions which may be temporally organized. Procedural texts can indeed be a simple, ordered list of instructions to reach a goal, but they can also be less linear, outlining different ways to realize something, with warnings, advice, conditions, hypothesis, preferences. They also often contain a number of recommendations and comments of various sorts. The organization of a procedural text is in general made visible by means of linguistic and typographic marks.

Research on procedural texts was initiated by works in psychology, cognitive ergonomics, and didactics, (Mortara et ali. 1988) (Adam 1987), (Kosseim 2000) to cite just a few. Several facets, such as temporal and argumentative structures have then been subject to general purpose investigations in linguistics, but they need to be customized to this type of text. There is however very little work done in Computational Linguistics circles. The present work is based on a preliminary experiment we carried out (Delpech et ali., 07), (Aouladomar, 05) where a preliminary structure was proposed, from corpus analysis.

Let us now give an illustrative example (translated from French), extracted from the 'Do-It-Yourself Home' domain, to show how texts are processed: *In the bedroom, it is necessary to clean curtains. These are cleaned first with a vacuum-cleaner to remove dust, then, if they are in cotton, they can be washed in the washing machine at 60 degrees; if they are white, it is even recommended to add some bleech so that they look whiter. With some starch, they can be easily ironed.*

In this text, the sequence: *In the bedroom, it is necessary to clean curtains* is analyzed as a justification of the actions to undertake. The next portion: *These are cleaned first with a vacuum-cleaner to remove dust, then, if they are in cotton, they can be washed in the washing machine at 60 degrees.* is the instruction kernel, where the last instruction is associated with a condition. Finally, *If they are white, it is even recommended to add some bleech so that they look whiter. With some starch, they can be easily ironed.* are two subordinated clauses, analyzed as being in the rhetorical relation advice to the kernel instructions.

Another example in parenthetical format (French gloss) is the following:

[$_{instruction}$ The first step consists in opening the computer box, then to place it on a large, clean surface,
[$_{advice}$ where you have ample space to work comfortably,]
and then to withdraw all the cashes at the PC front.]

A more complex case is:

[[$_{Goal}$ To clean leathers,]
[$_{instruction}$ choose specialized products dedicated to furniture,
[$_{advice}$ [$_{instruction}$ and prefer them colorless],
[$_{support}$ they will play a protection role, add beauty, and repair some small damages.]]]]

In the framework of argumentation, <TextCoop> can be used as a simple but efficient and flexible argument mining system. This is realized via the description of text patterns that describe the linguistic structure of arguments.

2 The software and the demo

<TextCoop> analyses procedures (dedicated to installation, maintenance, production, etc. over various technical and large public domains) in French and in English. Texts come either from the Web or from textual databases. <TextCoop> identifies structures and tags them in XML, in particular:

- Titles (goals of the text) and their structure, it also indexes titles (similarly to a search engine),
- Instructions and instructional compounds, temporal structure and conditions,
- Prerequisites (materials, tools, prior knowledge, etc.),
- Explanation structures: advice, warnings, illustrations, hints, evaluations, etc.

<TextCoop> is based on linguistic analysis, knowledge representation and language processing. The environment is modular and flexible so that it can adapt to users' needs and specificities via tailored interfaces. The linguistic interface allows for the specification of grammar rules or patterns (supporting gaps representing finite sequences

[1] IRIT-CNRS Toulouse France, email: sstdizier@irit.fr

of symbols), and lexical and ontological data. It also includes morphological analysis, and utilities such as web access, web page cleaning, and querying based on XML tags on the output structures. It is being made compliant with current data representation norms, via its planned embedding into the UIMA framework.

The rules and patterns include terminal (triggers as well as grammatical words like connectors) and non-terminal symbols (refering in general to 'local grammars', e.g. temporal expressions, instrument expressions, etc.). They also include gaps, standing for finite strings of words of no present interest, XML tags (treated as terminal elements), functions (for handling controls and for computing attribute values for tags), and insertion point specifications, designed to insert annotations at precise places. Symbols may also be associated with attribute value structures to encode morphology, syntax as well as semantic elements. An extraction rule for an advice is, for example:

```
begin-mark, Pronoun, to be/modal, adv,
verb/advice-exp, gap, end-mark.
```

as in: *It's better to install a grounding electrode.*
We advise you to use a waterproof box.

Similarly, for warnings, we have:

```
begin-mark, it, is, adverb,
important/necessary/compulsory/essential/
vital/fundamental, to, verb, gap, end-mark.
```

as in: *It is very important to be aware of electrical grounding safety.*
It is important to wash your hands before entering the room.

<TextCoop> recently got a much more stable implementation using Java, and in particular we developed an engine based on JFLEX and JCUP. This engine is based on an LALR(1) automaton that we have adapted so that it can handle non-determinism and gaps. Interfaces (user and administrator) have been defined so that it is quite easy, via some training, to describe patterns and lexical data for various semantic recognition treatments. At the functional level, <TextCoop> V.2 allows to (via customization):

- Index large volumes of written procedures and query your database,
- Enrich procedures, and make some controls on their contents (e.g. via business requirements),
- Add useful information : list of tools, materials, etc.,
- Develop advice and precautions to take for a given task, via document synthesis,
- Contribute to various types of applications, in particular risk analysis and prevention as detected from instructions.

A typical XML output of <TextCoop> is given below in Fig. 1. <TextCoop> can be viewed as the kernel of the system, to which dedicated add-ons or plug-ins can be added to realize specific, application-oriented tasks. These confer a stronger value to this system.

In terms of performances, we have at the moment the following results. On a standard PC, windows XP, an average of 300Mo of input texts is processed per hour, with a lexicon and ontology of a few thousand terms and about 25 tagging patterns or rules. We give below the performances for the major structures found in procedures.

As reported in (Fontan et al. 2008), we carried out an indicative evaluation on a corpus of 66 texts over various domains, containing 262 arguments. Those texts where manually annotated by a trained linguist, and the results were then compared with the system output. We get the following results for warnings:

conclusion recognition	support recognition	(3)	(4)
88%	91%	95%	95%

(3) conclusions well delimited (4) supports well delimited, with respect to warnings correctly identified.

We also carried out an indicative evaluation on the same corpus of 66 texts containing 240 manually identified advice. We get the following results for advice:

conclusion recognition	support recognition	(3)	(4)	(5)
79%	84%	92%	91%	91%

The software demonstration will include:

- Introduction to the software and its aims and functions,
- Automatic extraction of arguments (essentially warnings and advice) from procedural documents (technical and social) in French and English, attendees can propose texts, discussion of results (and errors),
- Introduction of new rules or patterns in the system to recognize new semantic structures,
- Demonstration of the adequacy of interfaces, of the introduction of domain knowledge (via an ontology), feedback from the audience,
- Automatic construction of a list of advice/warnings for a given topic.

An example of output is given below in Fig. 2.

REFERENCES

[1] Adam, J.M., *Types de Textes ou genres de Discours? Comment Classer les Textes qui Disent De et Comment Faire*, Langages, 141, pp. 10-27, 2001.

[2] Aouladomar, F., Saint-Dizier, P., *An Exploration of the Diversity of Natural Argumentation in Instructional Texts*, 5th International Workshop on Computational Models of Natural Argument, IJCAI, Edinburgh, 2005.

[3] Delpech, E., Murguia, E., Saint-Dizier, P., *A Two-Level Strategy for Parsing Procedural Texts*, VSST07, Marrakech, October 2007.

[4] Fontan, L., Saint-Dizier, P., Analyzing the explanation structure of procedural texts: dealing with advice and Warnings, STEP conference, Venice, August 2008.

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

[6] Mortara Garavelli, B., *Tipologia dei Testi*, in G. Hodus et al.: lexicon der romanistischen Linguistik, vol. IV, Tubingen, Niemeyer, 1988.

[7] Rosner, D., Stede, M., *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, 1992.

<instruction> <verb sem=”action”> Energize </verb> the Aircraft Electrical Circuits <tool> from the APU </tool> </instruction> <warning strength=”normal”> Make sure that circuit breakers are closed </warning>. <instruction> <temp-exp type=”durative”> During the test </temp-exp>, <verb type= ”epistemic”> obey </verb> the instructions shown <loc> on the terminal. </loc> </instruction> <instruction> <verb sem=”action”> Start </verb> the module test.</instruction> </explanation type=”evaluation”> The test is in progress as <verb type=”communication”> shown </verb> on the screen. </explanation> <instruction type=”check”> Indication that test is OK comes into view. </instruction> <instruction> <verb sem=”action”> Close </verb> the OMS session. </instruction>

Figure 1. Extract of an annotated procedure

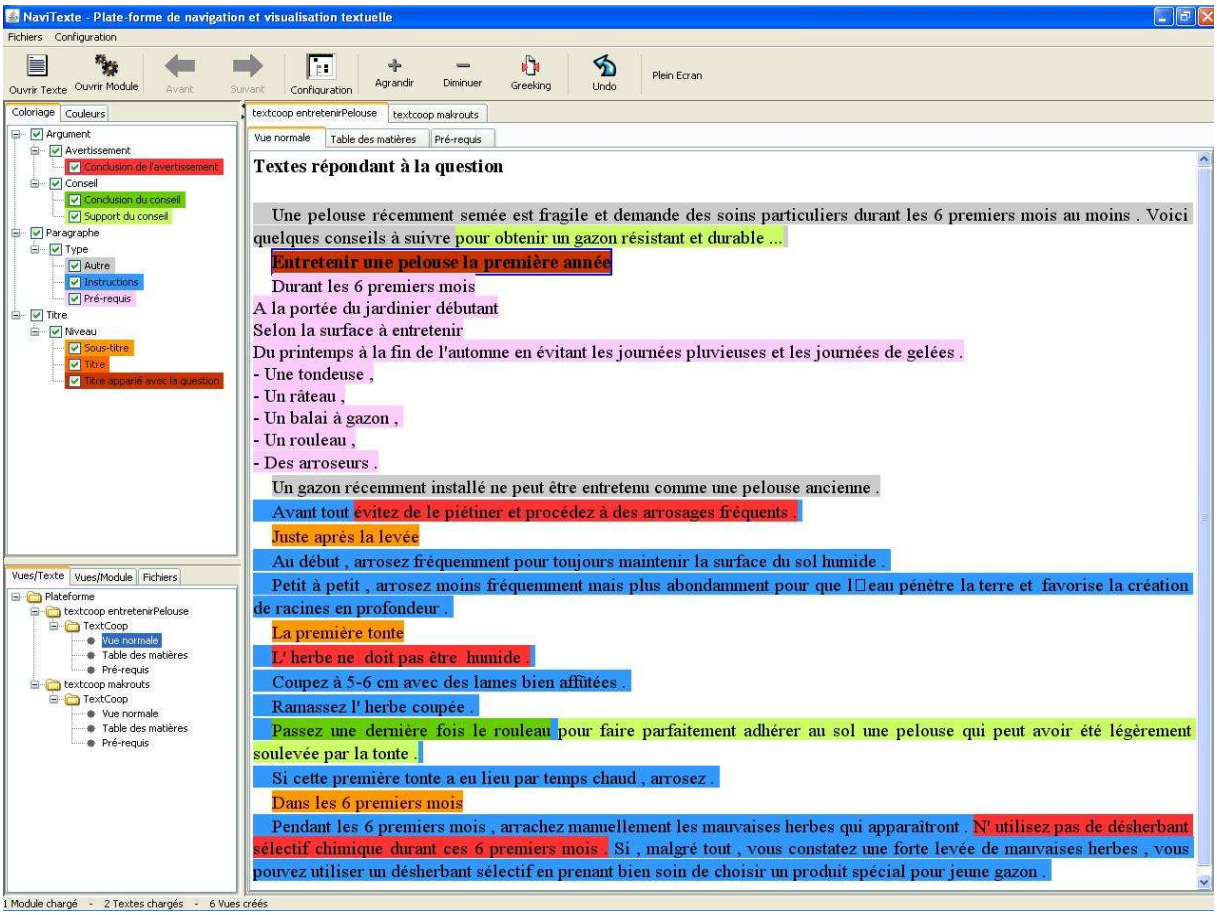

Fig. 2 Navitexte Output