Discourse Structure Analysis for Requirement Mining

Juyeon Kang*, Patrick Saint-dizier**

ARTICLE INFO

Article history:
Received 24 October 2013
Revised 26 December 2013
Accepted 28 December 2013

Keywords:
Natural Language Processing,
Discourse Structure Analysis,
Requirement Authoring,
Requirement Mining

ABSTRACT

In this work, we first introduce two main approaches to writing requirements and then propose a method based on Natural Language Processing to improve requirement authoring and the overall coherence, cohesion and organization of requirement documents. We investigate the structure of requirement kernels, and then the discourse structure associated with those kernels. This will then enable the system to accurately extract requirements and their related contexts from texts (called requirement mining). Finally, we relate a first experimentation on requirement mining based on texts from seven companies. An evaluation that compares those results with manually annotated corpora of documents is given to conclude.

1. Introduction

Technical documents form a linguistic genre with very specific linguistic constraints in terms of lexical realizations, including business or domain dependent aspects, and grammatical and style constructions. Typography and overall document organization are also specific to this genre. Technical documents cover a large variety of types of documents: procedures (also called instructional texts), which are probably the most frequently encountered in this genre, equipment and product manuals, various notices such as security notices, regulations of various types (security, management), requirements (e.g. concerning the design properties a certain product) and product or process specifications. These documents are designed to be easy to read and as efficient and unambiguous as possible for their users and readers in general. For that purpose, they tend to follow relatively strict authoring principles concerning both their form and contents. However, depending in particular on the industrial domain, the required security level, and the target user, major differences in the writing and overall organization quality can be observed.

Regulations and requirements form a specific subgenre in technical documents: they do not describe how to realize a task but the constraints that hold on certain types of tasks or products (e.g. safety regulations, management and financial regulations). Similarly, process or product specifications describe the properties and the expectations related to a product or a process. These may be used

* IRIT-CNRS, Toulouse cedex France (j.kang@prometil.com)
** IRIT-CNRS, Toulouse cedex France (stdizier@irit.fr)
International Journal of Knowledge Content Development & Technology, 3(2): 43-65, 2013.
http://dx.doi.org/10.5865/IJKCT.2013.3.2.043

as a contractual basis in the realization of the product at stake. Specifications often have the form of arguments.

Technical documents are seldom written from scratch: they are often the revision, the adaptation, or the compilation of previously written documents. Technical documents of a certain size are not produced by a single author: they result from the collaboration of several technical writers, technicians and validators. Their production may take several months, with several cycles of revisions and validation. Text revision and update is a major and complex activity in industrial documentation, which requires very accurate authoring principles and text analysis to avoid any form of ‘textual chaos’.

Design and software requirements must be unambiguous, short, relevant and easy to understand by professionals. They must form a homogeneous and coherent set of specifications. Safety requirements must be carefully checked since they may be as complex as procedures. Requirements must leave as little space as possible for personal interpretations, similarly to procedures (Van der Linden, 1993). The functions and the types of requirements are presented in depth in e.g. (Hull *et al*., 2011; Sage *et al*., 2009; Grady, 2006; Pohl, 2010; Nuseibeh *et al*., 2000). A lot of work is devoted to requirement engineering (traceability, organization), via systems such as e.g. Doors and its extensions (e.g. Reqtify, QuaARS). However, very little work has been carried out in the area of natural language processing (Gnesi *et al*., 2005).

Requirement documents are in general not a mere list of organized requirements. They often start with general considerations which are not requirements such as purpose, scope, or context. Then follow definitions, examples, scenarios or schemas. Then come a series of sections that address, via sets of requirements, the different facets of the problem at stake. Each section may include for its own purpose general elements followed by the relevant requirements. Each requirement can be associated with e.g. conditions or warnings and forms of explanation such as justifications, reformulations or illustrations. These documents are often produced over a long time span by various authors with different profiles and interests. It is therefore clear that authoring controls on the form and on the contents are necessary within a single document (which may be very large) and over several documents. A short extract of a requirement document is the example (5) given in section 2.

In this paper, we first introduce two main approaches to writing requirements and propose a method based on the Natural Language Processing to improve requirement authoring and the overall coherence, cohesion and organization of requirement documents. We investigate the structure of requirement kernels, and then the discourse structure associated with requirement kernels. Then, we briefly introduce the TextCoop platform and the Dislog language that we use to implement discourse analysis and requirements in particular. This will then allow an accurate extraction of requirements from texts (requirement mining). Finally, we relate a first experimentation on the requirement mining of documents coming from seven companies.

2. A Typology for Requirements

In 1998, the IEEE-STD association defined requirements in a very general way as follows:

> *A requirement is a statement that identifies a product or process operational, functional, or design characteristic or constraint which is unambiguous, testable or measurable, and necessary for product or process acceptability (by consumers or internal quality assurance guidelines).*

This general definition is then somewhat elaborated by the IEEE with the three following points:

- a condition or capability needed by a user to solve a problem or achieve an objective
- a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formal document
- a documented representation of a condition or capability of the two items above.

These definitions give a general framework for requirements, the second item being probably the most crucial. The third item describes the realization aspects. It is important to note the large difference in terms of size that we have between product requirements which are often expressed by means of very short sentences and safety requirements, which are complex from a language point of view. These latter often emerge from public regulations.

Several types of requirements have been identified in the literature. They often have very different language forms. Let us describe here the main types of requirements identified in the literature (Hull et al., 2011; Sage et al., 2009; Pohl, 2010). ***Project requirements*** define how the work at stake will be managed. This includes the budget, the communication management, the resource management, the quality assurance, the risk management, and the scope management. Project requirements focus on the who, when, where, and how something gets done. They are generally documented in the Project Management Plan. ***Product requirements*** include high level features, specifications or capabilities that the business team has committed to delivering to a customer. Product requirements do not specify how the features or the capabilities will be designed or realized. This is specified at a lower level. Product requirements include specification documents where the properties of a product are described prior to its realization. Specification documents often have a contractual dimension.

Functional requirements describe what the system does, what it must resolve or avoid. They include any requirement that outlines a specific way a product function or component must perform. These are more specific in general than product requirements. However, they do not explain how they can be realized. ***Non-functional requirements*** develop elements such as technical solutions, the number of people who could use the product, where the product will be located, the types of transactions processed, and the types of technology interactions. Non-functional requirements develop measurable constraints for functional requirements. A non-functional requirement may for example specify the amount of time it takes for a user with a given skill to realize a certain action.

For a given product or activity, requirements are produced by different participants, often with different profiles. For example, stakeholders are at the initiative of requirements that describe the product they expect. Then, engineers, safety staff, regulation experts and technicians may produce more specialized requirements, for example: quality, tracing, modeling or testing requirements (Hull et al., 2011). These documents may then undergo several levels of validation and update. The production and life cycles of requirements is often complex. This explains why it is necessary to control the writing quality of requirements, their contents and their homogeneity.

Requirements address a system as a whole and then its components and subsystems. Each component and subsystem may have its own requirements, possibly different if not contradictory with the main system ones. Even for a small, limited product, it is frequent to have several thousand requirements, hierarchically organized. The overall organization of requirements can be structures as given in table 1:

Table 1. Requirement Categories

A	Project requirement
	Product requirement (e.g. stakeholder requirement, system requirement, subsystem requirement, component requirement)
B	Functional requirement
	Non-functional requirement
C	Facets of requirements: performance requirement, management requirement,
D	Risk prevention requirement (e.g. Security regulations, business rules for security)

Requirements are now becoming a major activity in the industry. The goal is to specify a number of activities in a more rigorous, modular, structured and manageable way. Requirements may be proper to a company, an activity or a country, in this latter case, it may be a regulation. We now observe requirements in areas as diverse as product or software design, finance, communication, staff management and security and safety. These requirements have very different language forms. Software design is in general very concise and goes to the point. At the other extreme, safety requirements can be very long, describing all the precautions to take in a given circumstance. They often resemble procedures: they start with a context (e.g. manipulating acid) and then develop the actions to undertake: *wear gloves, glasses*, etc.

To conclude this introduction, let us give a few short illustrations:

- *(1) Login: Each login and each role change MUST be protected via authentication process. Motivation: Special functions or administrative tasks are only possible based on a role change. Misuse of a user account with too many authorizations is sharply reduced.*

- *(2) A password MUST be at least 8 characters long. PINs can be shorter for special requirements and MUST be at least four characters in length. Motivation: A minimum length of 8 characters plus the use of a large range of characters offers sufficient protection against brute force attacks.*

- *(3) The equipment must be tested and compliant with the following requirements of the main environmental European Standards:*
  climatic: ETSI EN 300 019 (the corresponding temperature class must be clearly given) and ETSI EN 300 119-5
  mechanic: ETSI EN 300 119-1 to -4
  acoustic: ETS 300-753 (table 1)
  chemical: ETSI EN 300 019-2-3, specification T3.1

transportation: ETSI EN 300 019-2-2, specification T 2.3

-*(4) Where an ECP exists on a road which is to be improved or will be subjected to major maintenance, the Design Organisation, in conjunction with the Overseeing Organisation, must discuss with the relevant Emergency Services, the need for the ECP to be retained.*

- *(5) BOMB THREAT. This is a major problem that has to be faced more and more frequently. It is important to remain calm and to contact authorities as early as possible. These will decide on evacuation conditions if appropriate. In this notice different scenarios are given which must be adapted to the context. The following cases are not independent, so it is crucial to read all of them.*

In case of a bomb threat:

a) Any person receiving a telephone threat should try to keep the caller on the line and if possible transfer the Area Manager. Remain calm, notify your supervisor and await instructions.

a) If you are unable to transfer the call, obtain as much relevant information as possible and notify the Area Manager by messenger. Try to question the caller so you can fill out the 'Bomb Threat Report Form'. After the caller hangs up, notify the Division Manager, then contact 911 and ask for instructions. Then contact Maintenance and the Governor's office.

c) The Division Manager will call the police (911) for instructions.

c1. If the Division Manager is not available call 911 and wait for instructions.

c2. Pull EMERGENCY EVACUATION ALARM. Follow evacuation procedures.

d) The person who received the bomb threat will report directly to the police station.

The first two examples are relatively short statements typical of software design. They include a kind of explanation: the motivations for such a constraint, so that it is clearly understood and accepted by the designer. The third example is related to the application of a regulation with the list of the different norms to consider. The fourth one is related to management and work organization. The last one is a typical (short) safety requirement or emergency notice implemented in a precise company (general safety requirements must often be customized to local environments). The requirement starts by a title and the context (*bomb threat*) and then it is followed by a structured list of instructions which are designed to be short and easy to understand by all the staff of that company.

3. Requirement Writing and Control

Requirement production is often a complex process that involves various actors and hierarchy levels (Alred, 2012; Ament, 2012). The different types of actors have been advocated above. It is frequent, in particular for safety requirements or generic requirements, that they undergo several steps of specification and customization to various precise environments. For example, a government can produce general regulations about chemical products. Then, public or private institutions may develop a more specialized account of these regulations for a given type or class of chemical product, going into more details and possibly interpreting these regulations. Then, at the end of this process, safety engineers adapt theses texts to their precise environment, product by product, activity by activity, possibly also building by building (e.g. fire alarm and related actions may differ from one building to another). Through all these steps, writing and coherence problems may easily arise.

Requirements must follow various authoring principles and recommendations. They are in general quite close to procedural texts writing (Barcellini et al., 2012; Saint-Dizier, 2014). A few norms have been produced and international standards have emerged. Let us note in particular: (1) IEEE Standard 830- Requirement Specification: 'Content and qualities of a good software requirement specification' and (2) IEEE Standard 1233, 'Guide for developing System Requirements Specifications'. Let us now review two of the main approaches to requirement authoring: boilerplates and *a posteriori* control.

3.1. Boilerplates: a Template-Based Approach

The boilerplate approach is essentially used for writing requirements in software and product design. These requirements are often very short sentences which follow strict and very regular formats. The RAT-RQA approach developed by the Reuse-company is a typical and simple example. Boilerplates are a technique that uses predefined simple language templates based on concepts and relations to guide requirements elicitation. These templates may be combined to form larger structures. Requirement authors must follow these templates. Boilerplates thus define a priori the language of requirements. Defining such a language is not so straightforward for complex requirements such as safety requirements, financial and management requirements.

The language introduced by boilerplates allow to write general structures such as propositions as well as more specialized components which are treated as adjuncts, such as capability of a system, *capacity (maximize, exceed), rapidity, mode, sustainability, timelines, operational constraints and exceptions*. Repositories of boilerplates have been defined by companies or research groups, these may be generic or activity dependent. An average size repository has about 60 boilerplates. Larger repositories are more difficult to manage. These are not in general publicly available.

The examples below illustrate the constructions proposed by boilerplates. Terms between < > are concepts which must receive a language expression possibly subject to certain restrictions:

General purpose boilerplates:

- The <**system**> shall be able to <**capability**>.
  The <u>washing machine</u> shall be able to <u>wash the dirty clothes</u>.
- The < **system** > shall be able to <**action**> <**entity**>.
  The <u>ACC system</u> shall be able to <u>determine</u> <u>the speed of the eco-vehicle</u>.

In these examples, the concept *action* is realized as a verb, the concept *entity* as a direct object and the concept *capability* as a verb phrase. The syntactic structures associated with each of these concepts are relatively well identified; their syntax may also be partly controlled. The different terms which are used such as e.g. *entities* can be checked by reference to the domain ontology and terminology.

Boilerplates can then be defined in terms of a combination of specialized constructions inserted into general ones. The following templates have been customized to a particular activity. They show various levels of granularity. In these examples, modals, prepositions and quantifiers are imposed:

- The <**system**> shall <**function**> <**object**> every <**performance**> <**units**>.
  The coffee machine shall produce a hot drink every 10 seconds.
- The < **system** > shall <**function**> not less than <**quantity**> <**object**> while <**operational conditions**>.
  The communications system shall sustain telephone contact with not less than 10 callers while in the absence of external power.
- The < **system** > shall display status messages in <**circumstance**>.
  The Background Task Manager shall display status messages in a designated area of the user interface at intervals of 60 plus or minus 10 seconds.
- The < **system** > shall be able to <**function**> <**object**> composed of not less than <**performance**> <**units**> with <**external entity**>.
- The < **system** > shall be able to <**function**> <**object**> of type <**qualification**> within <**performance**> <**units**>.
- While <**operational condition**> the <**stakeholder**> shall <**capability**>.
  While activated the driver shall be able to override engine power control of the ACC-system.
- The <**stakeholder**> shall not be placed in breach of <**applicable law**>.
  The ambulance driver shall not be placed in breach of national road regulations.

As the reader may note it, the user of such a concept-based system needs some familiarity with them to be able to use boilerplates correctly. Some types are relatively vague while others are very constrained. These latter require some training and documentation for the novice technical writer. Additional tools can be defined to help writing each.

The boilerplates has some advantages when producing large sets of short requirements in the sense that predefined templates guide the author, limiting the revisions needed at the form level. Revisions are still required at the contents level because a number of types remain general (*performance, operational conditions, capability*, etc.). It is however an attempt to standardize the requirement sublanguage, at least from a lexical and syntactic point of view. Using predefined attributes makes it to easier to write requirements for authors, in particular those with limited experience. Authors can select the most appropriate templates and instantiate concepts such as *capability, condition*. However, this language may not evolve so much, otherwise large revisions of already produced requirements will need to be updated. New templates can be added, but it is not possible to remove existing ones or to transform their structure.

Limitations are in relation with the low level of freedom that is offered to writers. Only relatively simple requirements can be written with this approach. Furthermore, authors which do not have a good command of the concepts and of what they cover will have a lot of difficulties to use boilerplates; this is obviously the case of stakeholders. Therefore, boilerplates may be used essentially for technical and low level requirements. In addition, some training may be necessary for writers in order for them to have a clear idea of the linguistic and conceptual role of the different templates and what they allow to write. Our experience is that in a number of companies where boilerplates are used, technical writers try to follow to some degree the boilerplate patterns, but also allow themselves quite a lot of freedom because boilerplates seem to be too rigid for what they must express.

3.2. An a posteriori Control of the Language Quality of Requirements

A different approach consists in letting requirement writers produce their documents rather freely and to offer them a tool to control their production a posteriori or upon demand. This is obviously a much less constraining view that leaves more freedom and flexibility to the author. This is well-adapted for domains or areas where requirements are complex, for example security requirements or regulations which may be several pages long. In that case, boilerplates are not appropriate.

This approach allows for high-level (writing methodology and guidelines) as well as low-level (lexical terms, business terms, grammar) controls. At the high level here are a few advice which are quite well known in the requirement community (e.g. Buddenberg, 2011):

Table 2. Recommendations for writing requirements

Accuracy	Systems of requirements must precisely define the system capabilities in a real-world environment, as well as how it interfaces and interacts with it. This aspect of requirements is a significant problem area for many systems.
Completeness and organization	No necessary information should be missing. The requirements should be hierarchically organized in the requirement document to help reviewers understand the structure of the functionality described, so that it is easier for them to tell if something is missing.
Consistency	A requirement must not conflict or overlap with other requirements (Wolf et al., 2005).
Feasibility	It must be possible to implement or to realize each requirement within the capabilities and limitations of the system and its environment.
Relevance	Each requirement should account for something that is a real need.
Objectivity	It is important to avoid value judgments via adjectives such as *easy, clear, effective, acceptable, suitable, good, bad, sufficient, useful*; also avoid subjective modals such as: *if possible, if necessary.*
Prioritized	An implementation or usage priority must be assigned to each requirement, feature, or use case (in terms of cost, risk, etc.).
Traceability	Each software requirement should be linked to its source, which could be a higher-level system requirement, a use case, or a voice-of-the-customer statement. Dates and authors should also be mentioned.
Validity	Each requirement should be understood, analyzed, accepted, and approved by all parties and project participants. This is one of the main reasons SRSs are written using natural language.
Verifiability	Tests or other verification approaches, such as inspection or demonstration, should be used to determine whether each requirement is properly implemented in the product.

4. An introduction to TextCoop and Dislog

Before going into the details of the structure of requirement rules, lets us first introduce the main features of the TextCoop platform that supports the Dislog language (Saint-Dizier, 2012). Dislog (for discourse in logic) has been primarily designed for discourse analysis. This platform has been used to implement the requirement analysis system we present here.

> Dislog follows the principles of logic-based grammars as implemented three decades ago in a series of formalisms, among which, most notably: Metamorphosis Grammars and Definite Clause Grammars. These formalisms were all designed for sentence parsing with an implementation in Prolog via a meta-interpreter or a direct translation into Prolog. Various processing strategies have been investigated in particular bottom-up parsing, parallel parsing, constraint-based parsing and an implementation of the Earley algorithm that merges bottom-up analysis with top-down predictions. These systems have been used in applications, with reasonable efficiency, robustness and a real flexibility to updates.

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 representing the different language forms taken by discourse structures.

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:

- L is a non-terminal symbol.
- 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.
- R is a sequence of symbols as described below, and
- 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. These are included between curly brackets as in logic grammars to differentiate them from grammar symbols.

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 in the rules,
- ***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. They should be preferred to terminal elements when generalizations over terminal elements are possible. Symbols can be associated with a type feature structure that encodes a variety of aspects 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 is 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.

- ***optionality and iteration marks*** 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. The length of the skipped string can also be controlled.
- ***a few meta-predicates*** to facilitate rule authoring.

Symbols in a rule may have any number of arguments. However, in our current version, to facilitate the implementation of the meta-interpreter and to improve its efficiency, the recommended form is:

```
identifier(Representation, Typed 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)
```

The typed feature structure (TFS) can be an ordered list of features, in Prolog style, or a list of attribute-value pairs. Examples are developed in the next chapter.

Similarly to DCGs and to Prolog clauses, it is possible and often necessary to have several rules to fully describe the different realizations of a given discourse function. They all have the same identifier Ident, as it is the case e.g. for the rules that describe NPs or PPs. A set of rules with the same identifier is called a ***cluster of rules***. Rule in a cluster are executed sequentially, in their reading order, from the first to the last one, by the <TextCoop> engine. Then, clusters are called in the order they are given in a ***cascade***. This is explained below.

As an illustration, let us consider a very generic and simple rule that describes conditional expressions:

```
Condition(R) --> conn(cond,_), gap(G), ponct(comma).
```

with:

```
conn(cond,_) --> if / when.
```

For example, in the following sentences, the underlined structures are identified as conditions: *If all of the sources seem to be written by the same person or group of people, you must again seriously consider the validity of the topic.*

In this rule, the gap(G) allows to skip the words which are of no present interest for the current analysis (they may be nouns, determiners, etc.). These words are however kept and reproduced in their original form in the result. The gap G covers the entire conditional statement between the mark *if* and the comma. The argument R in the above rule contains a representation of the discourse structure, for example, in XML:

> *<condition> .... </condition>....*
> *<condition> If all of the sources seem to be written by the same person or group of people, </condition>*
> *you must again seriously consider the validity of the topic.*

The rules given below and the associated lexical data have all been implemented in this framework.

5. The Discourse Structure of Requirement Documents

5.1. The analysis methodology

In this section we address in detail the structure of requirement documents. Requirements may come in isolation, as structured lists, or they can be embedded into larger documents. We first address here the organization of requirements in large documents, and then focus on the internal structure of requirements, which may have a complex discourse structure. By large document, we mean that requirements often do not come as isolated lists, but they are embedded into larger documents where they are associated with a large variety of information (definitions, contexts, etc.). One of our goals is to identify requirements in those documents and to investigate their relationships with the other types of information.

Our analysis of requirement structures is based on a corpus of requirements coming from 7 companies, kept anonymous at their request. Documents are in French or English. Our corpus contains about 500 pages extracted from 27 documents. These 27 documents are composed of 15 French and 12 English documents. The main features considered to validate our corpus are the following:

- requirements corresponding to various professional activities: product design, management, finance, and safety,
- requirements corresponding to different conceptual levels: functional, realization, management, etc.
- requirements following various kinds of business style and format guidelines imposed by companies,
- requirements coming from various industrial areas: finance, telecommunications, transportation, energy, computer science, and chemistry.

Diversity of forms and contents in this corpus allows us to capture the main linguistic features of requirements. We focus here on the linguistic and discursive structures of requirements, which parallel their logical structure. The logical structure has been widely addressed in the literature, as in e.g. (Hull *et al.*, 2011; Sage *et al.*, 2009; Grady, 2006; Pohl, 2010).

Considering the complexity of the discourse structures, we decided, in this our analysis, to realize a manual analysis and rule elaboration. We proceeded by generalizing over closely related language cues identified from sets of examples, as introduced in (Marcu, 1997; Takechi et al., 2003; Saito, 2006; Bourse et al., 2011) for procedural texts. Using learning methods would have required the annotation of very large sets of documents, which are difficult to obtain. Contrary to procedures, in requirements typography and punctuation is rather poor.

Once this analysis is realized, the rules and lexical data that follow are implemented on the TextCoop platform.

In general, in most types of specifications, requirements are organized by purpose or theme. The organization follows principles given in e.g. (Rossner et al., 1992). The structure of these specifications is highly hierarchical and very modular, often following authoring and organization principles proper to a company or an organization. Requirements may often be associated with diagrams or pictures, which will not be investigated here. They may even be organized as logigrams.

The higher level of these documents often starts with general considerations which are not requirements such as purpose, scope, or context. Then follow definitions, examples, scenarios or schemas. Then we have a series of sections that address, via sets of requirements, the different facets of the problem at stake. Each section may include for its own purpose general elements followed by the relevant requirements. Requirements can just be listed in an appropriate order or be preceded by text. Each requirement can be associated with e.g. conditions or warnings and forms of explanation such as justifications, reformulations or illustrations.

One of the challenges of our analysis is to be able to capture all the 'adjuncts' that characterize a requirement and give its scope, purpose, limitations, priority and motivations. This is an important difficulty in requirement mining, which is addressed below. Due to the complexity of the underlying semantics of discourse structures, this is a long-term research topic.

5.2. The structure of Requirement Kernels

A requirement is often composed of a main part, called its kernel, and additional elements such as conditions, goals or purposes, illustrations and various constraints. Let us first address the structure of requirement kernels. By kernel, we mean the main clause as can be seen in the real-life examples which are given in this article. As introduced in the section dedicated to boilerplates, requirements have quite a strict structure. We review below the main ones found in our corpus.

While most documents share a large number of similarities in requirement expression, there is quite a large diversity of structures which are only found in a subset of them. We categorized 20 prototypical structures for English which have different linguistic structures, which can be summarized, modulo lexical parameters, by the following eight rules.

The rules given below are written in Dislog, which is a format close to Definite Clause Grammars or BNF in grammar theory. They are designed to be easy to read and to write by linguists. In these rules, the symbols `bos` (beginning of structure) and `eos` (end of structure) are characterized by a punctuation, a connector, or simply the beginning or the end of a sentence or of an enumeration. Html marks can also be considered as starters or ending marks. These marks are crucial: they define the boundaries of requirements, whereas lexical marks allow the identification of requirements, in contrast e.g. with instructions. In the rules, gaps allow to skip finite sets of words of no present interest.

(1) ***Lexically induced requirements***: in this case, the main verb of the clause is *to require* or a closely related verb. The verb has a requiretype constraint, which is a lexical constraint:

```
requirement → bos, gap, verb(requiretype), gap, eos.
```

Company X requires equipment that complies with relevant environment standards (EMC, safety, resistibility, powering, air conditioning, sustainable growth, acoustic, etc.).

The notation 'requiretype' is a type we created (any verb in the require family is a candidate for that rule) for this rule that allows to structure verbs in the lexicon. Verbs such as 'need' and its morphological variants are also of this type.

(2) ***Requirements composed of a modal applied to an action verb***: a large number of requirements are characterized by the use of a modal (*must, have to* or *shall*) applied to an action verb in the infinitive. Other modals, which are less persuasive (*should, could*) must be avoided. This construction has a strong injunctive orientation. The lexical type action denotes verbs which describe concrete actions. This excludes a priori (but this depends on the context) state verbs, psychological verbs and some epistemic verbs (*know, deduce,* etc.). Adverb phrases are optional, they often introduce aspectual or manner considerations:

```
requirement → bos, gap, modal, {advP}, verb(action, infinitive), gap, eos.
```

The solution, software or equipment shall support clocks synchronization with an agreed accurate time source.

(3) ***Requirement composed of a modal, the auxiliary be and an action verb*** used as a past participle:

```
requirement → bos, gap, modal, aux(be), {advP}, verb(action, past-participle), gap, eos.
```

Where new safety barriers are required and gaps of 50 m or less arise between two separate safety barrier installations, where practicable, the gap must be closed and the safety barrier made continuous.

(4) A special case, with the same structure as in (3) are requirements which use an ***expression of type 'conformity'*** instead of an action verb, as, e.g. *to be compliant with, comply with.* The auxiliary *be* is included into this expression. The modal must be present to characterize the injunctive character of the expression:

```
requirement → bos, gap, modal, {advP}, expr(conform), gap, eos.
```

All safety barriers must be compliant with the Test Acceptance Criteria.

(5) A number of requirements include ***comparative or superlative forms*** or the expression of a minimal or maximal quantity or amount of an entity or a resource. The modal is present and the main verb is the auxiliary *have*, which plays here the role of a light verb. In the rule below, expr(quantity) calls a local grammar that includes various forms of expressions of quantity that include a constraint (maximal, minimal, superlative, comparative). These constraints are often realized by means of specific lexical items such as *at least one, a maximum of, not more than*, etc.

```
requirement → bos, gap, modal, have, expr(quantity), gap, eos
```

Physical entities must have at least one Ethernet interface per zone it is connected to (front, back, *administrative)*

The rule with the comparative form with the auxiliary *be* writes as follows. The symbol comparative_form is a call to a local grammar that describes the structure of comparative forms (*equal to, greater than*, etc.)

```
requirement → bos, gap, modal, aux(be), comparative_form, gap, eos.
```

For all other roads, the Containment Level at the ECP/MCP shall be equal to or greater than that of the adjacent safety barrier e.g. if the safety barrier is N2, then the ECP/MCP must also have a minimum N2 Containment Level.

(6) ***Requirements expressing a property to be satisfied***. This class of requirements is more difficult to characterize since properties may be written in a number of various way. They can be adjectives, or more complex business terms. In the rule below, the property is skipped, while the construction '*must be*' is kept since it is typical of requirements. However, this rule is not totally satisfactory as it stands since it may over generate incorrect cases.

```
requirement → bos, gap, modal, aux(be), gap, eos.
```

For terminals that face oncoming traffic, e.g. those at both ends of a VRS on a two-way single Carriageway road, the minimum Performance Class must be P4.

(7) When a document is a ***list of requirements*** without any other form of text, it is frequent that the modal *must* is omitted because there is no need to have specific marks for requirements. The modal is simply underlying. In that case, the rule is similar to an instruction with an action verb in the infinitive:

```
requirement → bos, verb(action, infinitive), gap, eos.
```

Leave a 1 / 8 - inch gap between the panels.

These rules require lexical resources, in particular: modal, auxiliaries, action verbs, adjectives referring to performance, completion, necessity or conformity. The open lexical categories can be constructed from corpus inspection and the addition of closely related terms. In general the vocabulary which is used in technical texts is not very large, except for business terms. Those terms are often defined in a terminology.

5.3. Implementation

These rules have been implemented in Dislog and run on the <TextCoop> platform. The system architecture follows the architecture of TextCoop. We have in particular:

- A general purpose lexicon that contains closed classes of words (pronouns, auxiliaries, modals, determiners, etc.) and symbols corresponding to punctuation and closely related marks (end of sentence or a few html tags such as those used for enumerations). This lexicon is not very big, it counts about 250 entries,
- One or more specialized lexicons, that correspond to the resources needed for the rules described above. We have in particular a lexicon for action verbs (about 5000 entries, this is large, but

in real-life applications, about 150 such verbs are used in a given domain), those verbs may receive semantic types such as the ‘requiretype’ or ‘action’ (action is opposed to state or psychological verbs for example) mentioned above. Morphological types are also introduced (e.g. ‘inf’ for infinitive). Technical nouns and deverbals are also included in case verbs are replaced by their deverbals.

- A morphological system for verbs and nouns,
- Local grammars that deal with various structures such as the syntax of comparative forms requested in rule 5. About 6 local grammars have been defined with about 12 rules in DCG format each.
- List of dedicated expressions, more or less generic or domain dependent, which have been collected from our corpora and made more generic, e.g. expression of conformity and quantity. In general, these expressions are relatively fixed and limited in diversity (about 140 such expressions are considered for our experiments). They may be partly grammatical, for example the expression of quantity includes a figure followed by a unit (e.g. 3 cms), which is expressed by a small grammar.
- A set of rules (as those above), 9 rules for requirement analysis and about 120 rules for the discourse structures presented in section 6 below.

These resources are directly handled by the TextCoop engine that has a dedicated processing strategy. These resources are stored in different modules allowing easy updates.

As a result, the system produces annotations of the following form (verbs are also tagged with their morphology: pp = past participle, inf = infinitive). These annotations are produced in a fully automatic way by the system:

<requirement> running services must be <verb type = "pp"> restricted </verb> to the strict minimum </requirement>

<requirement> equipments must < verb type = " inf " > provide < /verb> acl filtering </requirement>

<requirement> physical entities must have at least one Ethernet interface per zone it is connected to (front, back, administrative) </requirement>

<requirement> every administrative access must be initiated from a rebound server using a set of predetermined tools </requirement>

<requirement> for shared services, a shared information system interface enabler is <verb type = “pp"> required </verb> to route real time charging messages to local systems </requirement>

The above rules can be used in a number of applications such as requirement mining (e.g. Sampaio et al., 2005).

5.4. Evaluation

We have evaluated the accuracy of the above rules in the TextCoop framework. The task was, given a set of texts, to evaluate how accurate these rules are to identify requirements in any kind of technical document (other types of documents being irrelevant from a linguistic and conceptual point of view).

To have a correct and relevant evaluation, we considered a variety of technical texts: procedures, equipment description and requirement documents. Requirements could naturally be searched in these documents. We feel it is not very relevant to consider other types of documents such as novels, publications or emails because their genre is quite far from technical documents and trying to annotate requirements out of them would not lead to any useful conclusions. In the industry requirements are searched in a variety of technical documents, included working and meeting notes. We feel this is a sufficiently large data set to evaluate the relevance of our rules.

The technical documents considered are very diverse in terms of contents and style. Some of them do not contain any requirements. The overall volume of sentences which are requirements in these documents is below 10%, the remainder are instructions, titles, warnings, advice, lists of prerequisites, definitions and various other considerations.

The corpus considered for evaluation has the following characteristics:

Table 3. Evaluation data

File	French	English
Doc2	Texts randomly selected from the corpora concerning biomedical engineering and telecommunication	Texts randomly selected from the corpora concerning e-finance, telecommunication and mechanical handling
	33 pages, 9,511 words	*41 pages, 14,759 words*

The files doc2 allow us to evaluate the accuracy of the system on a different set of documents. This constitutes a direct evaluation of the performances of the rules.

These files have been manually annotated and the results reported below are based on a comparison between the manually annotated texts and the results automatically produced by the rules, implemented on the <TextCoop> platform. Results are the following:

Table 4. Evaluation results

Results	French	English
Manually Annotated requirements	35	45
Structures identified as requirements by the system	30	40
Erroneously tagged	0	6
Requirements not recognized	5	11
Accuracy (total errors/ total manually tagged)	86%	76%

These results are really good. This is for a large part due to the fact that requirements kernels follow very strict authoring guidelines independently of the documents in which they occur. Their recognition is therefore a rather easy task carried out with a modest number of rules.

This grammar can therefore be used for requirement mining. However, we then need to consider that requirements may contain additional elements such as conditions, contexts, illustrations, etc. This makes requirements more complex to process. This is developed in the next section.

At the moment, there are, surprisingly, very few systems that use natural language processing techniques to analyze the structure of requirements with the goal to contribute to their improvement (traceability, consistency checking, clustering, etc.). Let us note the system QuaARS (Gnesi et al., 2005; Barcellini et al., 2012).

6. Complex Requirements Analysis

Requirement kernels are often associated with various discourse structures. It is of particular interest to analyze the discourse structures (Mann et al., 1988/1992; Marcu, 1997/2000/2002; Rossner et al., 1992) which act as adjuncts to requirement kernels: conditions, goals, purpose, circumstance, illustration, restatement, etc. These structures are within the scope of a requirement. Samples are given below with examples for each relation. The entire system was initially developed for explanation analysis, it is developed and evaluated in (Saint-Dizier, 2014), it has been adapted here to requirements.

The examples below are also implemented and tested in the Dislog language. The formalism is quite straightforward; a `gap(G)` stands for a finite string of words to skip because they are of no present interest, `bos` and `eos` respectively stand for beginning and end of sentence. For each structure we give a rule sample, an example and the lexical resources which are needed. As the reader may note it, lexical resources are very limited, which makes the system generic and easy to use in different contexts, with no need a priori to update the resources.

A. Advice: An advice is an optional requirement meant to obtain better results; it is often paired with a support that gives its motivation.

Specific rules designed: 6 for advice, 3 for their supports.

```
Advice → verb(pref,infinitive), gap(G), eos./
      [it,is], adv(prob), gap(G1), exp(advice1), gap(G2), eos./ exp(advice2), gap(G), eos.
```

Resources:

verb(pref): *choose, prefer, ...*

exp(advice1): *a good idea, better, recommended, preferable*

exp(advice2): *a X tip, a X advice, best option, alternative*

adv(prob): *probably, possibly, etc.*

Example: *It is better to mention the capacity of high bandwidth probes, because these can be used for advanced tests.*

B. Warning: A warning draws the attention on a requirement or a situation which is crucial and possibly difficult, a support (reason) may give a motivation for taking this requirement into account or the risks which may arise if it is not taken into account (Fontan et al. 2008). Warnings have the highest priority level.

Specific rules designed: 9 for warnings, 9 for supports.

```
Warning → exp(ensure), gap(G), eos./
      [it,is], adv(int), adj(imp), gap(G), verb(action,infinitive), gap(G), eos.
```

Resources:
exp(ensure): *ensure, make sure, be sure*
adv(int): *very, absolutely, really adj(imp): essential, vital, crucial, fundamental*
Example: *It is essential that real time charging functions are implemented in a single enabler otherwise no coherent testing could be carried out.*

C. Conditions: are the factors, which may be diverse, under which a requirement is relevant. Specific rules designed: 8.

```
Condition → conn(cond), gap(G), ponct(comma)./ conn(cond), gap(G), eos.
```

Resources:
conn(cond): *if, when, ...*
Example: *When the local real-time charging system does not natively support diameter, protocol adaptation shall be performed by ...*

D. Concession: is a way to introduce exceptions in a requirement, for specific cases. This structure is quite rich and has many facets.
Specific rules designed: 9.

```
Concession → conn(opposition alth), gap(G1), ponct(comma), gap(G2), eos./
      conn(opposition alth), gap(G), eos./
      conn(opposition how), gap(G), eos.
```

Resources:
conn(opposition alth): *although, though, even though, even if, notwithstanding, despite, in spite of*
conn(opposition how): *however*
Example: *The manufacturer shall detail if protocols are used for health-checking mechanisms, however the by default protocol just needs to be mentioned.*

E. Contrast: is used in a requirement to relate actions used in opposite situations. Specific rules designed: 5.

```
Contrast → conn(opposition whe), gap(G), ponct(comma)./
      conn(opposition whe), gap(G), eos. /
      conn(opposition how), gap(G), eos.
```

Resources:
conn(opposition whe): *whereas, but whereas, while*
Example: *The periodic H34 controls must be carried out when the process is running correctly, whereas the cumulative H36 controls must be used in case of emergency or important delay.*

F. Circumstance: indicates factors under which the requirement is viewed or considered. This notion is quite large, it does not introduces any conditions as in (D), but simply sets a context.
Specific rules designed: 12.

```
Circumstance → conn(circ), gap(G), ponct(comma)./ conn(circ), gap(G), eos.
```

Resources: conn(circ): *when, once, as soon as, after, before*

Example: *Class A documents must be approved before IT managers confirm they fully understand all the elements in these documents in order to promote them within their countries.*

G. Purpose: gives the main aim(s) of the requirement: the reasons why it has been introduced.

Specific rules designed: 14.

```
Purpose → conn(purpose), verb(action,infinitive), gap(G), ponct(comma).
```

Resources: conn(purpose): *to, in order to, so as to*

Example: *To write a good report, you must do five things ...*

H. Illustration: may be associated with any of the above discourse structures. The goal is to make sure the structure is well understood.

Specific rules designed: 20.

```
Illustration → exp(illus eg), gap(G), eos./
      [here], auxiliary(be), gap(G1), exp(illus exa), gap(G2), eos./
      [let,us,take], gap(G), exp(illus bwe), eos.
```

Resources:

exp(illus eg): *e.g., including, such as*

exp(illus exa): *example, an example, examples*

exp(illus bwe): *by way of example, by way of illustration*

Example: *A dedicated airline task force (including AF, KLM, OS, IB) must be set up in order to further qualify the business opportunity.*

J. Restatement: rephrases a requirement or any of the above structures without adding any new information.

Specific rules designed: 9.

```
Restatement → ponct(opening parenthesis), exp(restate), gap(G), ponct(closing paren-
      thesis)./
      exp(restate), gap(G), eos.
```

Resources: exp(restate): *in other words, to put it another way, that is to say, i.e., put differently*

Example: *The interface must detail how transactions are processed in case of traffic overload, in other words it must indicate network load rates at each point.*

K. Justification: explains the motivations of the requirement and possibly details about the way it is termed.

Specific rules designed: 6.

```
Cause → conn(cause), gap(G), ponct(comma)./ conn(cause), gap(G), eos.
```

Resources: conn(cause): *because, because of, on account of*

ponct(comma): , ; :

Example: *Because specification documents are so thorough and long, you must learn how to gradually master them in a top-down way.*

In the implementation we carried out in TextCoop on discourse structures of this type (general results are given (Saint-Dizier, 2014)), the principle is to first analyze these discourse structures, before recognizing requirement structures. These are processed one after the other, via a cascade of automata. Then the requirement structure is processed and integrates these structures into its scope. Our implementation allows the recognition of a variety of complex requirements such as (below are real system outputs, with some indentations to be more readable):

```
<requirement>
          <condition> if required by the distributor< /condition> ,
we must be able to bill all usages on a single wallet and supervised some postpaid account </requirement> .

 <requirement>
the tenderer shall <verb type = "inf"> detail </verb> the average duration for the different phase
of the development of a new release of its product release definition, design, implementation, unit
test, end
          <purpose> in order to end tests. </purpose>  </requirement>.

<requirement>
for each of the network elements included in its solution, the tenderer shall <verb type = "inf ">
describe </verb> which hardware components
          <illustration> ie racks, chassis, servers, blades, processing units, i/o cards, mother cards,
daughter cards etc </illustration>
          of its overall platform are redundant < /requirement>.

<requirement>
          <condition> if geo redundancy is feasible </condition> ,
the tenderer will indicate which requirements ,
          <illustration> such as minimum link throughput , maximum latency , and maximum distance
between sites </illustration>,
have to be met in order to achieve geographical stateful redundancy </requirement>.

<requirement>
the performance figures provided here should be measured
          <circumstance> when the full functional capabilities of the platform are activated
</circumstance> </requirement>.

 <requirement>
the bidder must <verb type = "inf "> identify </verb> the information being held allowing the operator
          <purpose> to determine the sensitivity of the information </purpose>
and mandate the level of protection required </requirement>.
```

<***requirement***>
in case the solution, software or equipment is connected to an ip network
<***illustration***> *(being public, private, administration or uncontrolled)* </***illustration***>,
connectivity,
<***circumstance***> *when needed* </***circumstance***> ,
shall < verb type = "inf "> support < /verb> ciphered transport protocols
<***illustration***> *such as, but not limited to, ipsec, tls/ssl, ssh, etc* </***illustration***> </***requirement***>.

<***requirement***>
solution, software or equipment must be able to be adequately managed and controlled,
<***goal***> *in order to be protected from threats* </***goal***> , *and*
<***purpose***> *to maintain security for the systems and applications using the network* <***illustration***>
including information in transit </***illustration***>
</***purpose***>, </***requirement***>.

Besides the use of the modals *shall* or *must*, the reader can observe the use of other modals: e.g. *should* and *will* which have a lower injunctive or persuasion force. This is of interest to implicitly give priorities to requirements. However, it is not recommended to use these modals in requirements which then become weak.

7. Conclusion

In this paper, we have presented several features of requirements: (1) their purpose and the different forms they can have, and (2) authoring recommendations so that requirements can be understood straightforwardly. Two main trends govern the way requirements can be produced: the boilerplate approach, based on predefined templates, and an *a posteriori* control of the quality of the language. We have outlined the advantages and limitations of each of these approaches.

Then, we developed the structure of requirement kernels and the discourse structures which are often used in complex requirements. We also presented some simple elements and an evaluation for requirement mining based on the requirement structures we have developed. Requirements have relatively rigid forms, which makes their recognition a relatively feasible task with a good accuracy.

This kind of approach based on *a posteriori* control of the quality of the language can be also applied to the requirement mining of non European languages such as Korean and Japanese. For example, we find in requirement documents in Korean[1] some regularities which can be summarized in the two following categories: 1) requirement kernel marked by using the grammatical morpheme *-ya* often connected to an auxiliary verb like *hae-ya*; this is equivalent to the use of modals in requirements in English or in French, 2) requirement kernel marked by the composition of a noun and an auxiliary

[1] For this observation, we analyzed the examples of different types of requirements given in the “Guideline for writing requirements” proposed by Ministry of Security and Public Administration (MOSPA.go.kr).

verb like *upgrade+han-da, interface+han-da*.

Acknowledgements

This work was supported by the French ANR, LELIE project. We thank the students of the group who participated: Mathilde Janier, Camille Albert and Sarah Bourse. We also thank the companies (EDF, Orange, Thomson-Reuters, Liebherr, Airbus, EADS) that gave us extracts from their technical documents so that we could work on real-life texts. We thank the company Prometil for providing some technical support for this project. Finally, we thank anonymous reviewers whose comments helped improve this work.

References

Alred, G. J., Brusaw, C. T., & Oliu, W.E. (2012). *Handbook of Technical Writing*. St Martin's Press: New York.

Ament, K. (2002). *Single Sourcing. Building modular documentation*. W. Andrew Pub: New York.

Barcellini, F., Grosse, C., Albert, C., & Saint-Dizier, P. (2012). Risk Analysis and Prevention: LELIE, a Tool dedicated to Procedure and Requirement Authoring, proceedings of LREC'12, Istanbul.

Bourse, S., & Saint-Dizier, P. (2011). The language of explanation dedicated to technical documents. *Syntagma*, 27(1), 67-89.

Buddenberg, A. (2011). (Private communication). Guidelines for writing requirements.

Di Eugenio, B., & Webber, B.L. (1996). Pragmatic Overloading in Natural Language Instructions. *International Journal of Expert Systems*, 9(2), 53-84.

Donin, J., Bracewell, R. J., Frederiksen, C. H., & Dillinger, M. (1992). Students' Strategies for Writing Instructions: Organizing Conceptual Information in text. *Written Communication journal*, 9(3), 209-236.

Fontan, L., & Saint-Dizier, P. (2008). Analyzing the explanation structure of procedural texts: dealing with Advices and Warnings. In: J. Bos (Eds.). International Symposium on Text Semantics (pp. 221-242), Association for Computational Linguistics (ACL).

Gnesi, S., Lami, G. (2005). An Automatic Tool for the Analysis of Natural Language Requirements. *International Journal of Computer Systems Science & Engineering*, 20(1), 233-245.

Grady. J. O. (2006). *System Requirements Analysis*. Academic Press: USA.

Hull, E., Jackson, K., & Dick, J. (2011). *Requirements Engineering*. Springer Verlag.

Keil, F.C., & Wilson, R.A. (2000). Explanation and Cognition, Bradford Book.

Kintsch, W. (1988). The Role of Knowledge in Discourse Comprehension: A Construction Integration Model. *Psychological Review*, 95(2), 163-182.

Mann, W., & Thompson, S. (1988). Rhetorical Structure Theory: Towards a Functional Theory of Text Organisation, TEXT 8 (3), 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, proceedings of ACL'97.

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

Marcu, D. (2002). Au unsupervised approach to recognizing Discourse relations, ACL.

Miltasaki, E., Prasad, R., Joshi, A., & Webber, B. (2004). Annotating Discourse Connectives and Their Arguments, proceedings of the HLT/NAACL Workshop on Frontiers in Corpus Annotation.

Nuseibeh, B., & Easterbrook, S. (2000). Requirements Engineering: A Roadmap, ICSE'00 Proceedings of the 22nd international conference on Software engineering, 37-46.

Pohl, K. (2010). *Requirements Engineering: Fundamentals, Principles, and Techniques*. Springer Verlag.

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

Sage, P.A., & Rouse, W. B., (2009). *Handbook of Systems Engineering and Management*, 2nd Edition. Wiley: USA.

Saint-Dizier, P., (2012). Processing Natural Language Arguments with the <TextCoop> Platform. *Journal of Argumentation and Computation*, 3(1), 86-112.

Saint-Dizier, P., (2014). *Challenges of Discourse Processing: the case of technical documents.* Cambridge Scholars: UK.

Saito, M., Yamamoto, K., & Sekine, S. (2006). Using Phrasal Patterns to Identify Discourse Relations, proceedings of the Human Language Technology Conference of the NAACL'06, 133-136.

Sampaio, A., Loughran, N., Rashid, A., & Rayson, P. (2005). Mining Aspects in Requirements. Aspect-Oriented Requirements Engineering and Architecture Design Workshop, Chicago, Illinois, USA.

Stede, M. (2012). Discourse Processing, Morgan and Claypool Publishers.

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

Taboada, M., & Mann, W.C. (2006). Rhetorical Structure Theory: Looking back and moving ahead. *Discourse Studies*, 8(3), 423-459.

Taboada, M. (2006). Discourse markers as signals (or not) of rhetorical relations. *Journal of Pragmatics*, 38(4), 567–592.

Van der Linden, K. (1993). Speaking of Actions: choosing Rhetorical Status and Grammatical Form in Instructional Text Generation, Doctoral Dissertation, University of Colorado, USA.

Wolf, F., & Gibson, E. (2005). Representing Discourse Coherence: A Corpus-Based Study. *Computational Linguistics*, 31(2), 249-288.