Natural Language Processing of Requirements for Model-Based Product Design with ENOVIA/CATIA V6

Romain Pinquié, Philippe Véron, Frédéric Segonds, Nicolas Croué

▶ **To cite this version:**

Romain Pinquié, Philippe Véron, Frédéric Segonds, Nicolas Croué. Natural Language Processing of Requirements for Model-Based Product Design with ENOVIA/CATIA V6. 12th IFIP International Conference on Product Lifecycle Management (PLM), Oct 2015, Doha, Qatar. pp.205-215, 10.1007/978-3-319-33111-9_19 . hal-01377444

HAL Id: hal-01377444

https://inria.hal.science/hal-01377444

Submitted on 7 Oct 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Natural Language Processing of Requirements for Model-Based Product Design with ENOVIA/CATIA V6

Romain Pinquié[1], Philippe Véron[1], Frédéric Segonds[2], Nicolas Croué[3]

[1]LSIS, UMR CNRS 7296, Arts et Métiers ParisTech, Aix-en-Provence, France
[2]LCPI, Arts et Métiers ParisTech, Paris, France
[3]KEONYS, Toulouse, France

{romain.pinquie[1], philippe.veron[1], frederic.segonds[2]}@ensam.eu ;
[3]nicolas.croue@keonys.com

Abstract. The enterprise level software application that supports the strategic product-centric, lifecycle-oriented and information-driven Product Lifecycle Management business approach should enable engineers to develop and manage requirements within a Functional Digital Mock-Up. The integrated, model-based product design ENOVIA/CATIA V6 RFLP environment makes it possible to use parametric modelling among requirements, functions, logical units and physical organs. Simulation can therefore be used to verify that the design artefacts comply with the requirements. Nevertheless, when dealing with document-based specifications, the definition of the knowledge parameters for each requirement is a labour-intensive task. Indeed, analysts have no other alternative than to go through the voluminous specifications to identify the values of the performance requirements and design constraints, and to translate them into knowledge parameters. We propose to use natural language processing techniques to automatically generate Parametric Property-Based Requirements from unstructured and semi-structured specifications. We illustrate our approach through the design of a mechanical ring.

Keywords: Functional Digital Mock-Up; ENOVIA V6, CATIA V6; Natural Language Processing; Requirements; Parametric Modelling.

1 Introduction

1.1 ENOVIA/CATIA V6 RFLP for integrated, model-based product design

In 1990, Gero [1] proposed the FBS ontology where F stands for the set of functions, B for the set of expected behaviours (Be) and the set of actual behaviours (Bs), and S for the structure. In [2], Christophe extends the FBS ontology to RFBS by including the R for requirements. Back in the nineties, in his theory of axiomatic design, Suh [3] defined four domains of activities: the customer domain, the functional domain, the physical domain and the process domain. Stepping back and looking at these product design methods, which could also be assimilated to the

systems engineering process [4], we notice that product design relies upon an iterative process among requirements, functions, behaviours and structures.

The Dassault Système's ENOVIA/CATIA V6 software solution proposes a similar integrated product design model named RFLP [5]. The R is for ENOVIA V6 Requirements, a requirements management workbench. The F, L and P layers are used to recursively break down the complexity of the design problem according to the Functional, Logical and Physical viewpoints of the product. This design approach follows from Descartes' reductionism method that consists in understanding a complicated problem by investigating simple parts and then reassembling each part to recreate the whole. In RFLP, the functional layer (F) relies upon a Functional Flow Block Diagram to design functional architectures in which functions transform material, energy or information input flows into output flows whose consistency is ensured by the matching of input and output typed-ports. The logical layer (L) is the behavioural viewpoint of the product and is materialised by a logical architecture within which each logical unit's behaviour is equation-based modelled with the Modelica[1] language. Modelica models are executable thanks to the Dynamic Behaviour Modelling workbench that is the integration of Dymola[2] within CATIA V6. Finally, the physical layer (P) is very similar to the CATIA V5[3] CAD modeller.

The integrated RFLP product design environment enables designers to define implementation links between a pair of requirements, functions, logical units or physical organs so as to trace implementation relationships thanks to a traceability matrix. In addition to the traceability capability, the tight integration of ENOVIA V6 Requirements and CATIA V6 offers parametric modelling functionalities that can be used to make sure that the design artefacts comply with the requirements.

1.2 Problematic

Among the product life cycle phases defined by Terzi [6], we focus on the requirement analysis phase without addressing the management of the requirements during the downstream detailed design and testing life cycle phases.

Nowadays, a set of requirements is usually very large. Indeed, with the ever-increasing complexity of products and their relentless customisation, the mushrooming accumulation of legal documents, let alone the geographically dispersed teams through whom products are developed, a supplier is faced with a staggering increase in the number of requirements. For instance, at Mercedes-Benz, the size of a system-of-interest (SOI) specification varies from 60 to 2000 pages and prescribes between 1000 and 50 000 requirements [7]. In addition to the massive volume of requirements, most specifications are unstructured documents – e.g. Word, PDF – and 79% of requirements are written in unrestricted natural language [8].

For all these reasons, in a "*buy* approach" of a "make *vs* buy" decision, OEMs struggle to deliver products that comply with the legal and contractor's requirements. Indeed, when an OEM collects the specifications and the applicable documents the specification refers to, he has no other alternative than to go through the documents to

[1] https://www.modelica.org/
[2] http://www.3ds.com/products-services/catia/products/dymola
[3] http://www.3ds.com/fr/produits-et-services/catia/

identify the applicable requirements so as to, *in fine*, provide a product that complies with the contractor's requirements. There are four standard verification methods: inspection, analysis/simulation, demonstration, and test [9]. In this paper, we benefit from the simulation method that the parametric modelling CATIA V6's capabilities offer in its integrated RFLP product design environment. Parametric modelling-based verification can be very time consuming since designers have to: (1) read the specifications, (2) identify the values of the performance requirements and the design constraints, (3) model the values of the performance requirements and the design constraints as requirements' knowledge parameters, (4) design the behavioural and structural artefacts using parametric modelling, and (5) define the knowledge verification rules that map requirements' knowledge parameters with design artefacts' knowledge parameters so as to verify their compliance.

In a "*make* approach" there is no exchange of document-based specification. Therefore, before designing, the company simultaneously prescribes the product's requirements and the requirements' knowledge parameters into ENOVIA V6 Requirements. However, in a "*buy* approach", the OEM has to move the requirements from the unstructured specification documents to the ENOVIA V6 Requirement database and manually build requirements' knowledge parameters.

1.3 Literature Review & Proposition

According to Lash [10], modal verbs such as *shall*, *must* and *should* are key lexical features for classifying sentences corresponding to requirement statements. In [11], Coatanéa et *al.* use the Stanford Parser to apply natural language processing (NLP) techniques such as sentence splitting, tokenization and POS-tagging so as to create a binary rules-based classifier based on the presence or absence of a modal verb in a sentence. Zeni et *al.* [12] propose GaiuST, a framework that extracts legal requirements for ensuring regulatory compliance.

Few research studies attempt to extract text-based requirements (TBRs) from unstructured specifications; however, none of them address the challenge of extracting the values of performance requirements and design constraints to ease simulation-based design verification.

To avoid the very time-consuming requirements' knowledge parameters definition process, we propose a NLP pipeline to extract TBRs from unstructured and semi-structured specifications and to model them as Property-Based Requirements (PBRs). PBRs are used to automatically generate Parametric PBRs (PPRBs) in ENOVIA V6 Requirements. Finally, while designing with CATIA V6 RFLP, designers define behavioural and structural design knowledge parameters that are manually mapped to PPBRs thanks to parametric knowledge verification rules.

2 From Unstructured Specifications to Design Synthesis

The model-based product design process that we present is twofold: (1) we extract TBRs from document-based specifications and transform them into PPBRs in ENOVIA V6 R2015X; (2) we exploit the PPBRs in an integrated, parametric, model-based product design synthesis with CATIA V6 R2015X.

2.1 From Unstructured Specifications to PPBRs

Before presenting the NLP pipeline that generates the PBRs, we must present the concepts of PBR and PPBR.

As Micouin introduces in [13], a PBR is an unambiguous formal definition of a requirement as a predicate and is defined as follows:

$$\text{PBR: When } C \rightarrow \text{val}(O.P) \in D \quad (\mathbf{1})$$

This formal statement means: "W*hen the condition C is true, the property P of object type O is actual and its value shall belong to the domain D*". A relevant characteristic of the concept of PBR is that it is grammar-free, i.e. a PBR does not have any particular syntactic structure and can therefore be implemented with various modelling language such as VHDL-AMS [14] and Modelica.

By combining the PBR theory with parametric CAD modelling, we coin the concept of Parametric PBR (PPBR). A PPBR is a PBR that is implemented with a parametric CAD modeller thanks to knowledge parameters and knowledge verification rules. In ENOVIA V6, a PPBR is analogous to the formal combination of an Object (O) – the subject in the requirement statement attribute – with one or several knowledge parameters that define the boundaries of the constrained domain (D) of a property (P), whereas the condition (C) is a CATIA V6 knowledge verification rule that is manually defined by designers while designing behavioural and structural artefacts.

The generation of PPBRs from TBRs requires: (1) an NLP[4] pipeline to generate an XML specification that stores a set of PBRs derived from TBRs, and (2) to interpret the XML specification of PBRs for creating PPBRs in ENOVIA V6 Requirements.

To derive PBRs from TBRs we implemented the following NLP pipeline:

Step 1 <Uploading>: The user uploads one or several specifications whose extension is .doc(x) (Word), .odf (OpenOffice), .pdf (Portable Document Format), .xls(x) (Excel), .xmi (SysML requirements diagram). While uploading, the file uploader item gets the input stream of each specification.

Step 2 <Parsing>: We trigger a specific parser according to the file extension of each specification. If it is a .doc or .odf, the parser uses the Apache Tika[5] API to extract each specification content and transform it into .html semi-structured data. We transform the content into HTML because it makes the analysis of tables, lists, headings, etc. a lot easier. The headings of .doc and .odf help us to identify the sections. Sections are used for multi-threading to run processing tasks in parallel. If the extension of the specification corresponds to a .pdf, we use the native capability of Word to convert from .pdf into .doc. Then, as for .doc, we use the Apache Tika API to convert from .doc into .html. Once we get the .html specification, we verify whether the .pdf was generated with Word or OpenOffice by looking for the header, footer or table HTML tags, which are not present in a .pdf that was created with another text editor such as LaTeX. If we find that the .pdf was generated with Word or OpenOffice, then we call the .doc parser; otherwise, we use the .pdf parser that relies upon the Apache Tika API and various regular expressions. The second scenario is

[4] One should refer to [15] for further details on statistical natural language processing.

[5] https://tika.apache.org/

less accurate as we lose the structures (tables, enumerations, footer, header, etc.). The .xls(x) parser uses the Apache POI[6] API to parse the textual content of each cell. We make the hypothesis that each cell is a sentence. Finally, the .xml parser extracts the requirement statements between the XML tags of a SysML requirement diagram.

Step 3 <Tokenization>: The Stanford CoreNLP [16] Tokenizer[7] API iteratively tokenizes each specification content, that is, it chops the textual content up into pieces of a sequence of characters that are grouped together as a useful semantic unit for processing, the *tokens*. We store the tokens in a term-sentence matrix whose rows are sentences and columns are tokens that make up each sentence.

Step 4 <Lemmatization>: The Stanford CoreNLP Lemmatizer API iteratively normalises each token by removing the inflectional ending and returns the dictionary form, the *lemma*. For instance, lemmatization reduces the tokens "requires", "required" and "require" to their canonical form "require". This enables us to increase the recall in step 8 <Classification>.

Step 5 <POS-tagging>: The Stanford CoreNLP POS-tagger[8] API iteratively POS tags each token, that is, it annotates each token with its grammatical category (noun, verb, adjective, adverb, etc.), the *Part Of Speech (POS).*

Step 6 <Sentence splitting>: The Stanford CoreNLP API iteratively splits each textual specification content into sentences.

Step 7 <Sentences cleaning>: We use various regular expressions and analyse HTML tags to clean the sentences. For instance, we rebuild sentences from enumerations, get rid of the headings, headers, footers and informative sections (introduction, scope, table of content, glossary, list of acronyms, etc.) that may generate false positives, and extract the content of .pdf tables.

Step 8 <Classification>: We use a knowledge engineering – a.k.a rules-based – text classification approach [17] to binary classify each sentence into a "requirement" *vs* "non-requirement" class. The algorithm iteratively traverses the matrix whose rows are sentences and columns are lemmas. For each iteration, if the condition "$token_i$ of $sentence_j$ = a prescriptive term $\in$ {shall, must, should, have to, require, need, want, expect, wish or desire}" is true, the current $sentence_i$ is classified as a requirement.

Step 9 <Dependencies analysis>: The Stanford CoreNLP Dependencies Analyzer[9] [18] API iteratively analyses each requirement to generate a semantic graph within which we identify the numeric dependencies and extract the source and target nodes of each dependency. The source of a numerical dependency is a numerical token annotated with the POS tag (CD), whereas the target is a word.

Step 10 <Classification>: While going through the dependencies list of each requirement, we check whether the word stored in the target node of each dependency is a physical unit such as N, °C, kg, Pa, etc. using a resource file that collects all existing physical units under its abbreviated and expanded form – e.g. *N* and *Newton*. Each time a given numerical dependency is classified as a physical numerical dependency, we add a third attribute from our resource file that is the dimension of the physical unit – e.g. *Force* for the unit *N* or *Newton*.

---

[6] https://poi.apache.org/
[7] http://nlp.stanford.edu/software/tokenizer.shtml
[8] http://nlp.stanford.edu/software/tagger.shtml
[9] http://nlp.stanford.edu/software/stanford-dependencies.shtml

Step 11 <PBR Pattern analysis>: A well-written TBR prescribing a functional level of performance or a design constraint usually follows three distinct syntactic patterns (Pattern 1, 2 and 3) [19]. Note that the condition is not always specified; consequently, there is one more syntactic pattern (Pattern 4).

Pattern 1: <Prescriptive> <Domain> <Condition> - PDC The Control_Subsystem shall open the Inlet_Valve in less than 3 seconds when the temperature of water in the Boiler is less than 85 °C.
Pattern 2: <Prescriptive> <Condition> <Domain> - PCD The Control_Subsystem shall, when the temperature of water in the Boiler is less than 85 °C, open the Inlet_Valve in less than 3 seconds.
Pattern 3: <Condition> <Prescriptive> <Domain> - CPD When the temperature of water in the Boiler is less than 85 °C the Control_Subsystem shall open the Inlet_Valve in less than 3 seconds.
Pattern 4: <Prescriptive> <Domain> - PD The Control_Subsystem shall open the Inlet_Valve in less than 3 seconds.

Given a list of conditional terms (when, if, while) and a list of prescriptive terms (shall, must, should, have to, require, need, want, expect, wish or desire), we know the index of the conditional term and the prescriptive term by iterating through the tokens of a given requirement. Thus, to identify the pattern associated to a given requirement we use a set of rules that compares the index of the physical numeric dependencies, the index of the prescriptive term, and the index of the conditional term. This set of rules enables us to infer the *Domain D* and the *Condition C*.

A physical numerical value is sometimes followed by a tolerance. Thus, the patterns 1, 2 and 3 give rise to four more patterns where the *domain D* and the *condition C* are split into a *nominal domain*, a *tolerance domain*, a *nominal condition* and a *tolerance condition* – e.g. pattern 5 follows from pattern 1.

Pattern 5: <Prescriptive> <Nominal Domain> <Tolerance Domain> <Nominal Condition> <Tolerance Condition> - PnDtDnCtC The Control_Subsystem shall open the Inlet_Valve in 3 seconds +/- 1 second, when the temperature of water in the Boiler is between 70 °C and 85 °C.

To make sure that two consecutive physical numerical dependencies form the so called <nominal, tolerance> pair of a *domain D* or a *condition C*, we check whether their units belong to the same physical dimension. For instance, in the requirement "*When the temperature is less than 40°C, the pressure shall be less than 30 Pa*", the consecutives physical numerical dependencies <40 °C> and <30 Pa> do not belong to the same physical dimension since the former is a temperature (°C), whereas the latter is a pressure (Pa). However, in the requirement "*The system shall control a pressure of 30 Mpa +/- 5 Pa*", the physical numerical dependencies <30 Mpa> and <5 Pa> belong to the same physical dimension – a pressure.

Finally, there are six more syntactic patterns according to whether there is a tolerance associated to the domain and/or the condition – e.g. pattern 7 follows from pattern 1 and 5.

Pattern 7: <Prescriptive> <Domain> <Nominal Condition> <Tolerance Condition> - PDnCtC The Control_Subsystem shall open the Inlet_Valve in less than 3 seconds, when the temperature of water in the Boiler is between **70 °C** and **85 °C**.

Step 12 < Tolerance calculation>: The calculation of the minimum, maximum and nominal values defining the tolerance of a *condition C* or a *domain D* relies upon four patterns: (1) "X +/- Y" with X > Y, (2) "X more or less Y" with X > Y, (3) "from X to Y" with X < Y and (4) "between X and Y" with X < Y. If there is no tolerance, e.g. "*the temperature shall be less than 50°C*", the maximum and minimum values of the domain are identical. At present, there is a limit when the unit is not the same, e.g. "1 daN +/- 10 N" or "from 10 Pa to 1 MPa", because we cannot compute the tolerance without using a unit convertor.

Step 13 <PBRs modelling>: The NLP pipeline ends up with an XML specification that lists the PBRs in a structure that complies with the PPBRs data model in ENOVIA V6 Requirements. Thus, each PBR element has a *statement*, a *nominal value*, a *minimal value* and a *maximum value* that specify a *domain D* that can be inferred from the *nominal domain* and *tolerance domain*, a *physical dimension* and a *unit* attribute. The XML specification can finally be imported into ENOVIA V6 Requirements so as to automatically generate the PPBRs. This pure software development part of our proposal has not been implemented yet.

Once the PPBRs have been generated in ENOVIA V6 Requirements, designers can start the design synthesis thanks to the F, L and P layers of CATIA V6 RFLP.

2.2 Integrated, Parametric, Model-Based Product Design Synthesis

Design synthesis is the translation of input requirements into possible solutions satisfying those inputs [20].

The design synthesis with the integrated CATIA V6 FLP product design environment consists in translating the input requirements (R) into functional, logical and physical solutions satisfying those inputs. In order to do so, designers recursively break down the functional requirements into functions (F) that transform flows. Functions are then implemented by dynamic logical units (L) that simulate the expected behaviour, whereas non-functional requirements that prescribe design constraints are implemented by inert structural organs (P). Once the design of a given hierarchical level is completed, we apportion the performance requirements of the current hierarchical level to the functions of the next lower level by either sticking with the same physical dimension (allocation) or by establishing new requirements resulting from specific implementation choices (derivation).

Parametric modelling enables designers to not only create flexible CAD model, but also to verify that the requirements comply with the design artefacts. When the PPBRs are directly specified in ENOVIA V6 Requirements, engineers have to manually create the requirements and the associated knowledge parameters. However, when requirements are imported from a document-based specification, the generation of PPBRs results from the NLP pipeline. The knowledge verification rules that link the PPBRs and the knowledge parameters of the design artefacts require domain-specific knowledge; consequently, they are manually defined by designers.

In the next section, we illustrate the transformation of TBRs into PBRs so as to generate PPBRs that drive the design synthesis of a mechanical ring. We use a mechanical ring as a case study because its design changes frequently and because it is a simple universally understood object.

3 Case study

3.1 From Unstructured Specifications to PPBRs

First, we put ourselves in the shoes of a contractor who wants to acquire a mechanical ring. We only write three requirements[10] to ease the illustration of our proposition. Each requirement is in a different specification (.doc, .pdf and SysML).

Then, we play the role of the OEM who receives the specifications. First, we upload the specifications and send them through the NLP processing pipeline. Once it has finished, the NLP processor generates the XML specification of PBRs.

In the XML specification, the data structure of the PBRs is defined in such a way that the PBRs are interpretable by ENOVIA V6 Requirements. Therefore, we can generate the PPBRs (Fig. 1) from the PBRs. Nevertheless, this functionality is not integrated into ENOVIA V6 because the NLP pipeline is a single capability of a broader on-going research project that requires Java EE as programming language.

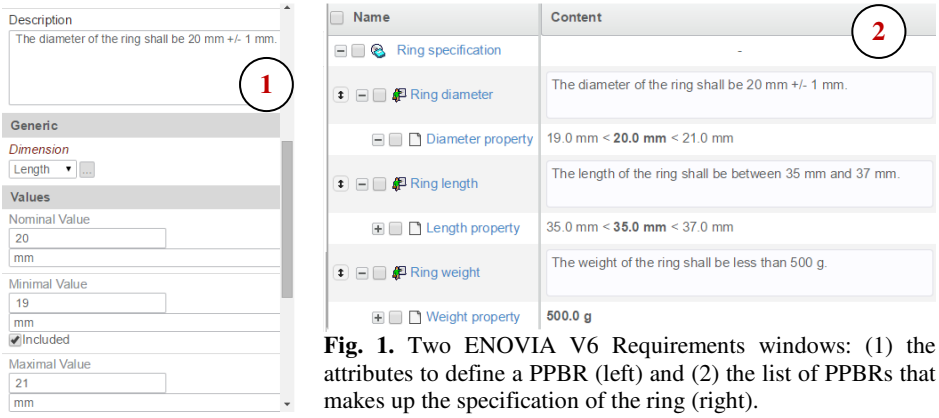

Fig. 1. Two ENOVIA V6 Requirements windows: (1) the attributes to define a PPBR (left) and (2) the list of PPBRs that makes up the specification of the ring (right).

3.2 Integrated, Parametric, Model-Based Product Design Synthesis

Now that the PPBRs are specified (Fig. 2 (1)), designers can start the design synthesis of the ring with CATIA V6 RFLP. We only have non-functional PPBRs that prescribe design constraints, therefore they will be implemented by a structural design artefact, the ring, and more precisely, the external diameter, length and weight properties of the ring. By using parametric modelling, we define a design parameter for each property (Fig. 2 (3)). A design parameter is a triple <Name, Physical dimension, Unit> – e.g. a parameter named <D> whose physical dimension is <length> and unit is <mm> that drives the external diameter property of the ring.

After having associated the design artefacts' knowledge parameters with the geometrical features, designers define knowledge verification rules between the PPBRs and the design artefacts' knowledge parameters. For instance, Fig. 2 (4) shows the knowledge verification rule verifying that the external diameter property of the

[10] (Req 1.) The diameter of the Ring shall be 20 mm +/- 1 mm. (Req. 2) The length of the Ring shall be between 35 mm and 37 mm. (Req. 3) The Ring shall weight less than 500 g.

ring belongs to the domain 20 mm +/- 1 mm. It consists in defining a conjunction between two partial order relations "<" that constrain the design knowledge parameter <D> with the maximum and minimum values of the external diameter property (Fig. 2 (4)) defined in the "Ring diameter" PPBR (Fig. 2 (2)). As shows Fig. 2 (5), a red light signals when the design does not comply with a requirement. In our case, the design artefact's knowledge parameter that stands for the external diameter property of the ring does not belong to the domain prescribed by the "Ring diameter" PPBR.

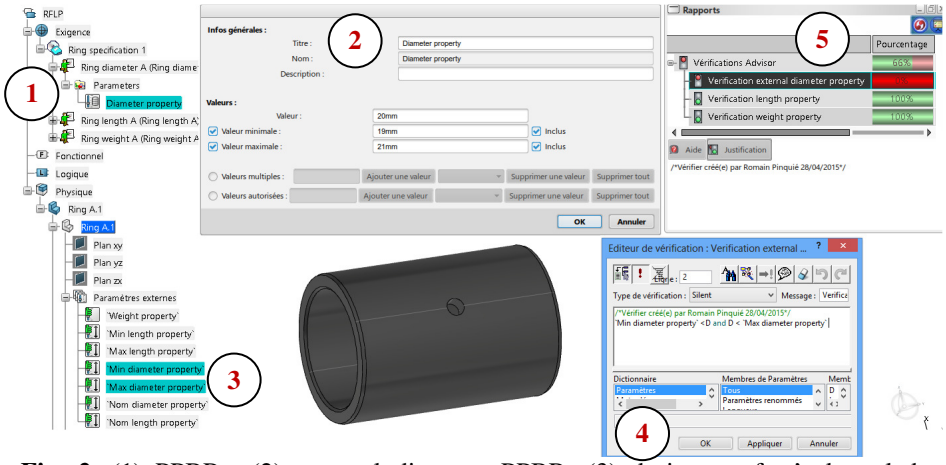

Fig. 2. (1) PPBRs, (2) external diameter PPBR, (3) design artefact's knowledge parameters, (4) knowledge verification rule, and (5) quantitative level of compliance.

3.3 Results & Limitations

We conducted experiments with both real industrial and handcrafted data sets. The initial results that we obtained after analysing handcrafted specifications are encouraging. One key factor is that we wrote the requirements by following the requirements writing best practices defined in [19]. Regarding the analysis of industrial specifications, the results are also promising, although we cannot fully validate the proposition without including a units converter.

Our solution presents a few limitations, such as the detection of sections when the headings functionality has not been used to edit .doc, .odf or .pdf. We were also challenged by original writing-style that came across while we were testing. The Stanford CoreNLP dependencies analyser relies upon statistics; consequently, it returns few false positive and false negative numerical dependencies. Finally, as previously explained, the translation of PBRs from TBRs is limited since the nominal and tolerance values must have the same unit.

4 Conclusion & future work

This paper presents a natural language processing pipeline to derive Parametric Property-Based Requirements from textual requirements.

In the future we plan to continue testing our algorithm on various specifications and integrate a unit converter to compute the minimum, maximum and nominal values defining the tolerance of a *condition C* or a *domain D* when the nominal and tolerance values do not have the same unit – e.g; 1 MPa +/- 10 Pa. We will also develop the plug-in to load the XML specification into ENOVIA V6 Requirements so as to automatically generate the PPBRs from the PBRs.

References

1 Gero, J.: Design prototypes: a knowledge representation schema for design. AI magazine, 11(4), 26-36 (1990)
2 Christophe, F., Bernard, A., Coatanéa, É.: RFBS: a model for knowledge representation of conceptual design. CIRP annals – manufacturing technology, 59(1), 155--158 (2010)
3 Suh, N.P.: Axiomatic design: advances and applications. Oxford University Press (2001)
4 ISO/IEC 15288.: Systems and software engineering – System life cycle processes (2008)
5 Kleiner, S., Kramer, C.: Model based design with systems engineering based on RFLP using V6. In: Smart product engineering, Springer, New York, Heidelberg, 93--102 (2013)
6 Terzi, S., Bouras, A., Dutta, D., Garetti, M., Kiritsis, D.: Product Lifecycle Management - from its history to its new role. Product Lifecycle Management, 4(4), 360--389 (2010)
7 Houdek, F.: Challenges in automotive requirements engineering. In: Industrial presentations by requirements engineering: foundation for software quality, Essen (2010)
8 Mich, L., Franch, M., Novi Inverardi, P.: Market research for requirements analysis using linguistic tools. Requirements Engineering,), 40--56 (2004)
9 ISO/IEC/IEEE 29148.: Systems and software engineering – Life cycle processes requirements engineering. 1--94 (2011)
10 Lash, A.: Computational representation of linguistics semantics for requirements analysis in engineering design. MSc thesis, Clemson University (2013)
11 Coatanéa, É., Mokammel, F., Christophe, F.: Requirements models for engineering, procurement and interoperability: a graph and power laws vision of requirements engineering. Technical report, Matine (2013)
12 Zeni, N., Kiyavitskaya, N., Mich, L., Cordy, J.R., Mylopoulos, J.: GaiusT: supporting the extraction of rights and obligations for regulatory compliance. Requirements engineering, 20(1), 1--22 (2015)
13 Micouin, P.: Toward a property based requirement theory: system requirements structured as a semilattice. Systems engineering, 11(3), 235--245 (2008)
14 Micouin, P.: Property-model methodology: a model-based systems engineering approach using VHDL-AMS. Systems engineering, 17(3), 249--263 (2014)
15 Manning, C., Schütze, H.: Foundations of statistical natural language processing. MIT Press, Cambridge (1999)
16 Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Berthard, S.K., McClsky, D.: The Stanford CoreNLP natural language processing Toolkit. In: 52nd annual meeting of the association for computational linguistics: system demonstrations, 55--60 (2014)
17 Feldman, R., Sanger, J.: The text mining handbook. Advanced approaches in analyzing unstructured data. Cambridge University Press (2007)
18 Cer, D., de Marneffe, M-C., Jurafsky, D. Manning, C.D.: Parsing to Stanford dependencies: trade-offs between speed and accuracy. In: LREC (2010)
19 INCOSE.: Guide for writing requirements. Requirements working group, International Council on Systems Engineering, San Diego, CA (2015)
20 INCOSE.: Systems engineering handbook. A guide for system life cycle processes and activities. Version 3.2. (2010)