A spaCy-based tool for extracting variability from NL requirements

Alessandro Fantechi
Dipartimento di Ingegneria dell'Informazione,
Università di Firenze
Firenze, Italy
alessandro.fantechi@unifi.it

Stefania Gnesi
Istituto di Scienza e Tecnologie dell'Informazione
"A.Faedo", Consiglio Nazionale delle Ricerche, ISTI-CNR
Pisa, Italy
stefania.gnesi@isti.cnr.it

Samuele Livi
Dipartimento di Informatica, Università di Pisa
Pisa, Italy
s.livi2@studenti.unipi.it

Laura Semini
Dipartimento di Informatica, Università di Pisa
Pisa, Italy
laura.semini@unipi.it

ABSTRACT

In previous work, we have shown that ambiguity detection in requirements can also be used as a way to capture latent aspects of variability. Natural Language Processing (NLP) tools have been used for a lexical analysis aimed at ambiguity indicators detection, and we have studied the necessary adaptations to those tools for pointing at potential variability, essentially by adding specific dictionaries for variability. We have identified also some syntactic rules able to detect potential variability, such as disjunction between nouns or pairs of indicators in a subordinate proposition. This paper describes a new prototype NLP tool, based on the spaCy library, specifically designed to detect variability. The prototype is shown to preserve the same recall exhibited by previously used lexical tools, with a higher precision.

KEYWORDS

Natural language requirements, NLP tools, variability

ACM Reference Format:
Alessandro Fantechi, Stefania Gnesi, Samuele Livi, and Laura Semini. 2021. A spaCy-based tool for extracting variability from NL requirements. In *Proceedings of 25th ACM International Systems and Software Product Lines Conference (SPLC'21).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Ambiguity defects in requirements may in some cases give an indication of possible variability, whether in design, implementation choices or configurability aspects. In fact, the ambiguity defects found in a requirements document may be due to references, intentional or unintentional, made in the requirements to problems that can be solved in different ways, perhaps by imagining a family of different products rather than a single product. Variability in a requirement represents the "ability" of the requirement to correspond to more than one possible configuration and to be modified or extended according to the specific context in which it is instantiated. Below we present a classification of those ambiguity defects that indicate potential points of variability.

- Vagueness: the requirement contains words that do not have an uniquely quantifiable meaning. Vague words are: *acceptable, bad, clear, clearly, difficult, easy, good, hard, important, inefficient, irrelevant, significant, simple, strong, unfair...*
- Weakness: the requirement contains a 'weak' verb, that makes the sentence non-imperative. Words that detect subjectivity are *can, could, may...* Weakness is directly classifiable as a variability when it introduces a possible choice in the configuration of the system or the optionality of a feature.
- Passive form: a sentence in passive form is considered ambiguous if there is no indication of the agent performing the action (the *by* is missing). If different product variants are envisaged for different agents (subjects of the active form), then there might be a variation point.
- Optionality: the requirement contains an optional part, a part that may or may not be considered. The words identifying optionality are: *eventually, if needed, optionally, possibly, probably...* Optionality is directly classifiable as a variability when it introduces a possible choice in the configuration of the system, i.e. a possible presence/choice of the characteristic in a given instance of the system.
- Multiplicity: the requirement does not refer to a single object, but addresses a list of objects, typically using disjunctions or conjunctions (*and, or, and/or,...*), and this is a well known source of ambiguity. In particular, conjunction or disjunction of noun phrases (NP multiplicity) likely highlights a range of features.
- Variability Escape Clauses: These are subordinate clauses containing both a conjunction *if, where, when, whether, ...* and a verb like *available, provided, implemented*, as for instance in the requirement "*If* run-time diagnostics are *implemented*, all failures should be available to be recorded in a train-borne recorder". Variability Escape Clauses clearly reveal a variation point.

SPLC'21, 06–11 September, 2021, Leicester, UK

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

This paper describes the tool support provided by a new prototype NLP tool, based on the spaCy library (https://spacy.io/), specifically designed to detect variability. The prototype demonstrated better performance than the results obtained with the tool QuaRS previously used to the same purpose.

2 SPACY

In order to provide a performing implementation to new syntactic rules able to detect potential variability, such as disjunction between nouns (instead of disjunction tout-court), agentless passive voices, or pairs of indicators in a subordinate clause we propose a new NLP tool, based on spaCy, specifically designed to detect variability.

spaCy is an open-source library designed to help build tools for natural language processing (https://spacy.io/). The library, written in Python and Cython, is licensed by MIT and currently implements statistical neural network models in a large number of natural languages, e.g, English, German, Spanish, Portuguese, French, Italian, Dutch, Greek... spaCy supports deep learning-based analysis, allowing the use of statistical models trained using machine learning libraries such as TensorFlow, Keras, Scikit-learn and PyTorch. In addition, spaCy's machine learning library, called Thinc, is available as an open source library for Python.

spaCy typically requires the loading of trained pipelines, to define linguistic annotations - determining, for example, whether a word is a verb or a noun. A trained pipeline typically consists of a few components that use a statistical model trained on labelled data. The trained pipelines for various languages can be installed as individual Python modules. We have used the `en_core_web_sm` pipeline package for English, which exhibits the best tradeoff between dimension/execution speed and accuracy.

The main features of spaCy that have been used are: Tokenization, Part-of-speech (POS) Tagging, Dependency Parsing, Lemmatization, Sentence Boundary Detection (SBD), Rule-based Matching. spaCy provides a library called Displacy which displays the syntactic derivation tree and the names assigned to each unit (see fig. 1

```
import spacy
from spacy import displacy

nlp = spacy.load("en_core_web_sm")
doc = nlp("This is a sentence.")
display.serve(doc, style="dep")
```

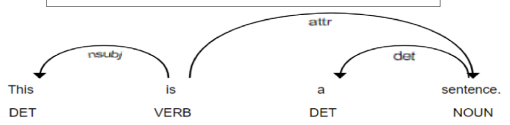


Figure 1: Syntactic derivation tree represented by DisplaCy

2.1 Usage

To use the spaCy library in Python, after downloading and installing a trained pipeline a model can be loaded using the instruction:

```
nlp=spacy.load('en_core_web_sm')
```

This assigns to `nlp` an object of type Language containing all the components and data needed to process the text string, called "text". Calling the `nlp` object on a text returns a document (doc):

```
ex_doc = nlp('this is the text to be processed')
```

When `nlp` is invoked on a text, spaCy first tokenizes the text to produce a Doc object. The document is then processed in several steps: this is also called a "processing pipeline". Trained pipelines are normally composed of a POS tagger, a parser, a Sentence Recognizer, an Attribute Ruler and a Lemmatizer. To implement our tool we have defined a pipeline composed, after the Tokenizer, of a Tagger, a Parser and a Lemmatizer, as shown in Figure 2.

The final result will be the document containing all the generated information, in a single object that is usually called *doc*.

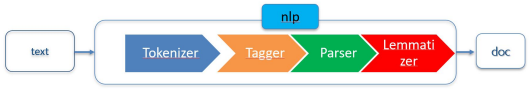


Figure 2: spaCy pipeline used in the tool

3 RUNNING EXAMPLE

To describe the components of the tool, we consider, as a running example, the following requirements document describing an e-shop.

R1 The system shall enable user to enter the search text on the screen.
R2 The system shall display all the matching products based on the search.
R3 The system possibly notifies with a pop-up the user when no matching product is found on the search.
R4 The system shall allow a user to create his profile and set his credentials.
R5 The system shall authenticate user credentials to enter the profile.
R6 The system shall display the list of active and/or the list of completed orders in the customer profile.
R7 The system shall maintain customer email information as a required part of customer profile.
R8 The system shall send an order confirmation to the user through email.
R9 The system shall allow an user to add and remove products in the shopping cart.
R10 The system shall display various shipping methods.
R11 The order shall be shipped to the client address or to an associated store, if the "shipping to store" service is available.
R12 The system shall enable the user to select the shipping method.
R13 The system may display the current tracking information about the order.
R14 The system shall display the available payment methods.
R15 The system shall allow the user to select the payment method for order.
R16 After delivery, the system may enable the users to enter their reviews and ratings.
R17 Shipping time should be as fast as possible.
R18 The system must report the available products, if the availability of these are less than 10% the system should show a pop-up.

4 DESIGN AND PROTOTYPE IMPLEMENTATION

We present here the architecture and prototype of the spaCy-based tool for NL analysis towards variability elicitation.

Once the language model (trained pipeline) has been applied to the requirements document, the process of identifying linguistic ambiguity indicators is divided into two steps: lexical analysis and syntactic analysis.

4.1 Lexical Analysis

The lexical analysis is carried out by means of a lexical parser and a Rule-Based Matcher (Python's Phrase Matcher). The variability introduced by the characteristics such as vagueness, optional choices and weakness described in the qualitative model are easy to identify since it is sufficient to detect whether or not the sentence under examination includes a term contained in the respective dictionary.

4.2 Syntactic Analysis

The syntactic analysis is based on the application of rules to the document processed by the spaCy pipelineThrough syntactic analysis we are able to localize: passive voices in which the agent is not indicated; variability escape clauses; NP multiplicity.

The analysis exploits the annotations of a dependency parser together with some spaCy methods.

To illustrate how the prototype works we examine 4 requirements inspired by the e-shop example: one in non-passive form, two in passive form without the presence of the word "by" and one in passive form with the presence of the word "by".

(1) The order shall be shipped to the client address.
(2) The order shall be delivered to the customer's address by courier company.
(3) Payment systems should be displayed.
(4) The system shall send an order confirmation to the user through email.

We expect the analysis to signal requirements 1) and 3).

The parser returns the number of matches found and then, for each match, the verb in passive form and the matching requirement.

```
Total matches found: 2
Match found: shipped . Requirement: R1 The order shall be shi
pped to the client address.
Match found: displayed . Requirement: R3 Payment systems shou
ld be displayed.
```

The result is as expected, namely the printing of the two requirements in passive form for which the agent is not specified. Here, to explain the rule, we have used four requirements inspired by e-shop, in Section 5 we provide the outcome of the application of the passive-form parser to the actual case study.

5 APPLICATION OF THE TOOL TO THE E-SHOP EXAMPLE: THE FULL PICTURE

We now show how to use the outcome of the tool for constructing a feature diagram (FD).

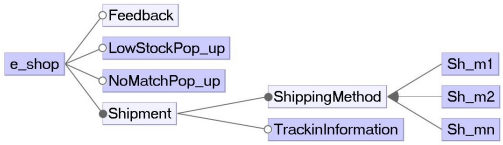
We have discussed in Section 1 which form of variability can be hidden behind each of the ambiguity defects and syntactical construct considered, providing a mapping from linguistic aspects to variability points. The representation of variability in a FD follows the standard [7]. The direct mapping from linguistic defect to relations in a FD can be found in previous work [5]. In the following of the section we summarise the output of each analysis, and build the corresponding partial trees, then we glue all them together.

Lexical Analysis. : The tool found 7 matches, out of which 5 highlight a variability and 2 are just ambiguities. Requirements to consider for variability are the following:

R3 The system possibly notifies with a pop-up the user when no matching product is found on the search.
R10 The system shall display various shipping methods.
R13 The system may display the current tracking information about the order.
R16 After delivery, the system may enable the users to enter their reviews and ratings.
R18 The system must report the available products, if the availability of these are are less than 10% the system should show a pop-up.

In these requirements there is a vagueness (various), an optionality (possibly), and some weak verbs (may, should).

Optionalities and weak verbs are all modelled with optional features (See Figure below). In the case of R16 a fictitious feature, *Feedback*, is needed to abstract reviews and ratings. The vagueness must be resolved asking the e-shop application commissioner which are the shipping methods to be offered, then it is modelled with an abstract feature *ShippingMethod* having the identified sub-features as children. For now, we use placeholders *Sh_m1, ... Sh_mn* for them and assume they are in *or* relationship. Then the commissioner may refine this relation and tell that some or all are mandatory or that they are mutually exclusive.



Passive form. : 1 match found for this indicator:

R11 The order shall be shipped to the client address or, if the "shipping to store" service is available, to an associated store.

The ambiguity here is related to not having specified who is responsible for sending, and the disambiguation may reveal variability: there may be some e-shops that use a central warehouse, others that use a number of distributed warehouses, as modelled in the following Figure

Variability escape clauses. : 2 matches found, one variability and the other a false positive. The requirement to consider for variability is:

R11 The order shall be shipped to the client address or to an associated store, if the "shipping to store" service is available.

Variability escape clauses are straightforwardly modelled with an optionality. In this case the *shipping to store service* is optional:

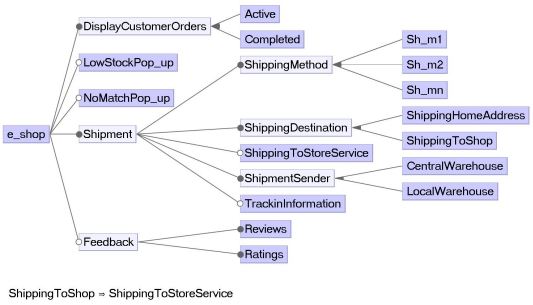
NP multiplicity. : The tool found 3 matches, out of which the first two are disjunctions and the third contains a conjunction.

R6 The system shall display the list of active and/or the list of completed orders in the customer profile.
R11 The order shall be shipped to the client address or to an associated store, if the "shipping to store" service is available.

R16 After delivery, the system may enable the users to enter their reviews and ratings.

Both *or* and *and/or* are modeled by default with a disjunction between features. Conjunction is not directly an indicator of variability, however a conjunction can aid in the process of family elicitation by typically finding sibling features that share the same optionality/mandatory indication. In this case we have detected two sibling mandatory features that are children of the optional abstract feature *feedback* introduced above. The resulting partial FD is below.

The complete e-shop feature diagram, obtained as a merge of the partial trees built so far, includes also the constraint defined in requirement R11: `ShippingToShop`→`ShoppingToStoreService`.



6 DISCUSSION AND RELATED WORK

We compared the performance of the new tool against that of the previously used tool, QuARS.

The comparison was based on two documents containing 256 and 65 requirements, respectively. Results are in Figure 3: the precision is highly improved, with a negligible loss in recall. Precision was indeed our goal: although other authors in the RE community recommend to maximise recall [10], increasing precision can make NLP for RE approaches more applicable in practice, in particular for variability detection. In fact, while it is of utmost importance to identify ambiguities in safety-critical system requirements, unidentified variabilities do not have such serious consequences, and losing a bit of recall in favor of accuracy would speed up the system analysis process.

In the requirement engineering of SPLs several researches have focused on exploiting Natural Language Processing (NLP) techniques and tools to extract information related to features and variability from Natural Language (NL) requirements documents [1–6, 9] that can be used as a source from which variability-relevant information can be elicited.

Systematic literature reviews on feature identification and variability extraction from NL documents have been published by Li *et al.* [8] and by Bakar *et al.* [2]. Our work differs from previous contributions in that they define techniques for feature identification, variability extraction being a consequence, while we use a technique that operates in the opposite way, looking directly for variation

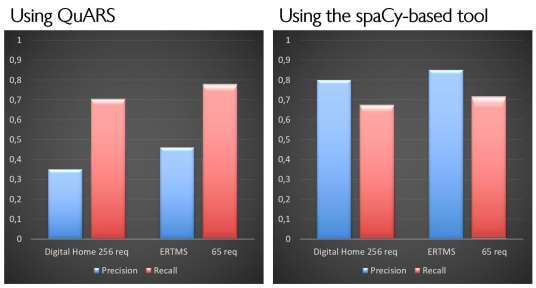


Figure 3: Precision and Recall vs QuARS

points. In addition, our approach integrates research on ambiguity detection with research on identifying points of variation.

7 CONCLUSION

In previous work we have shown that ambiguity detection in requirements can represent latent aspects of variability, and we have experimented with NLP tools able to detect ambiguities and hence to point at potential variation points. The latest works in this research stream have pointed at the limits of previously used tools for what concerns the variability indicated by particular syntactic constructions. In this paper we have presented a novel tool support to include detection of such syntactic indicators. The developed prototype has been shown to improve precision over the previous tool on a running example.

Future work includes the refinement of the represented tool to reduce false positives (that is, ambiguity detections that turn out not to be variation points) by means of a graphical interface that allows to accumulate in a dictionary specific terms that will be ignored in subsequent analyses.

REFERENCES

[1] M. Acher, A. Cleve, G. Perrouin, P. Heymans, C. Vanbeneden, P. Collet, and P. Lahire. On extracting feature models from product descriptions. In *Proc. of VaMoS '12*, pages 45–54, 2012.
[2] N.H. Bakar, Z.M. Kasirun, and N. Salleh. Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. *Journal of Systems and Software*, 106:132–149, 2015.
[3] A. Fantechi, A. Ferrari, S. Gnesi, and L. Semini. Hacking an ambiguity detection tool to extract variation points: an experience report. In *Proc. of VAMOS'18*, pages 43–50. ACM, 2018.
[4] A. Fantechi, A. Ferrari, S. Gnesi, and L. Semini. Requirement engineering of software product lines: Extracting variability using NLP. In *Proceedings of the 26th IEEE Int. Requirements Engineering Conf.*, pages 418–423. IEEE, 2018.
[5] A. Fantechi, S. Gnesi, and L. Semini. Ambiguity defects as variation points in requirements. In *Proc. of VAMOS '17*, pages 13–19. ACM, 2017.
[6] N. Itzik, I. Reinhartz-Berger, and Y. Wand. Variability analysis of requirements: Considering behavioral differences and reflecting stakeholders' perspectives. *IEEE Trans. on Software Engineering*, 42(7):687–706, 2016.
[7] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon Univ Pittsburgh, Software Engineering Inst, 1990.
[8] Y. Li, S. Schulze, and G. Saake. Reverse engineering variability from natural language documents: A systematic literature review. In *Proc. of the 21st SPLC*, pages 133–142. ACM, 2017.
[9] S. Ben Nasr, G. Bécan, M. Acher, J.B. Ferreira Filho, N. Sannier, B. Baudry, and J.M. Davril. Automated extraction of product comparison matrices from informal product descriptions. *Journal of Systems and Software*, 124:82–103, 2017.
[10] S.F. Tjong and D.M. Berry. The design of SREE - a prototype potential ambiguity finder for requirements specifications and lessons learned. In *Proc. of RESFQ*, volume 7830 of *LNCS*, pages 80–95. Springer, 2013.

APPENDIX - TOOL DEMONSTRATION

The tool demonstration will show the execution of the tool on example requirements documents containing potential variability indications. In particular, the demonstration will show how lexical analysis and syntactic analysis performed by the tool are able to point to potential variation points. Figures 4 and 5 show the tool working in the two cases.

An accompanying short video is available here: https://stlab.dinfo.unifi.it/fantechi/videolink.html

.....

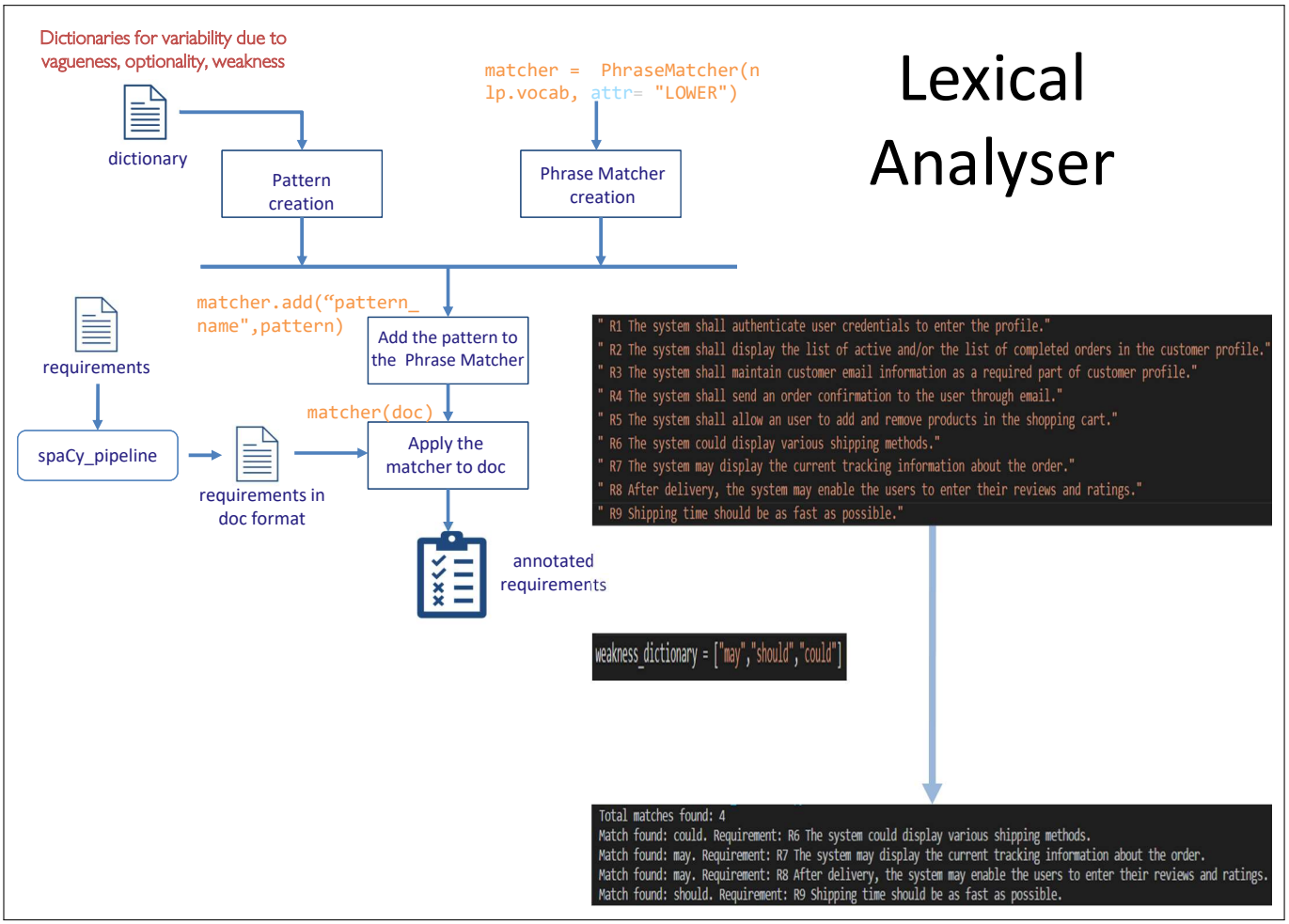


Figure 4: Lexical analyzer

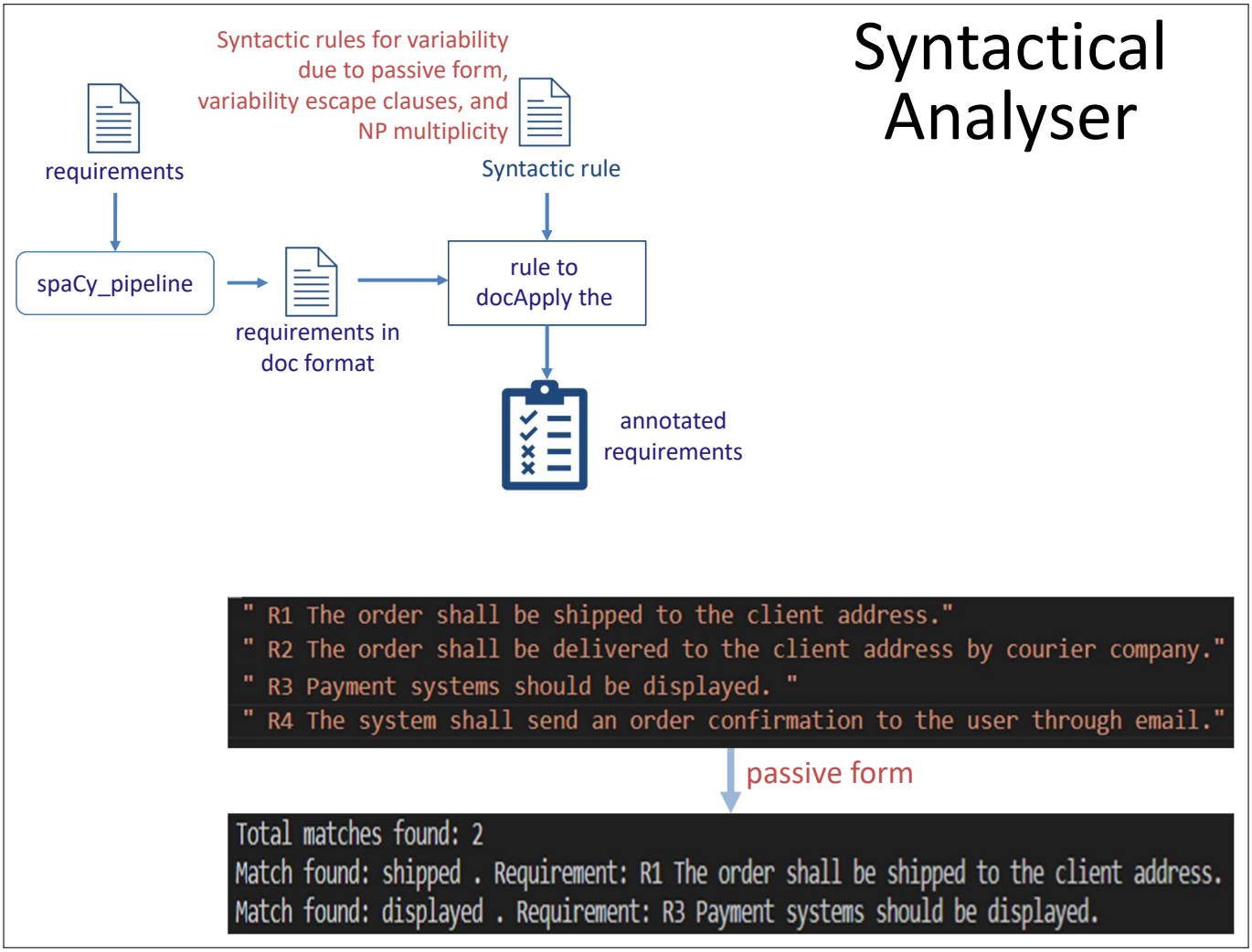


Figure 5: Syntactic analyzer