Deriving Service-Oriented Dynamic Product Lines Knowledge from Informal User-Requirements: AI Based Approach

Najla Maalaoui
National School of Computer Sciences
RIADI Lab
Manouba University, Tunisia
email: najla.maalaoui@ensi-uma.tn

Raoudha Beltaifa
National School of Computer Sciences
RIADI Lab
Manouba University, Tunisia
email: raoudha.beltaifa@ensi.rnu.tn

Lamia Labed Jilani
National School of Computer Sciences
RIADI Lab
Manouba University, Tunisia
email: lamia.Labed@isg.rnu.tn

***Abstract*—A Service-Oriented Dynamic Software Product Line (SO-DSPL) is a family of service-oriented systems sharing a set of common features. Hence, they are automatically activated and deactivated depending on the running situation. Such product lines are designed to support their self-adaptation to new contexts and requirements. Particularly, user requirements can be analyzed and enriched thanks to the existing of the SO-DSPL ontology that we previously built. This will facilitate the configuration of a derived service from the family of services, corresponding both to the desired requirement and a specific context. As we know, a user requirement can be ambiguous, vague and incomplete, which motivate the need for the extraction of the hidden knowledge. Our challenge is to use artificial intelligence techniques to automatically extract new SO-DSPL knowledge from textual user requirements and derive appropriate services of the service line for the user. In this paper, our approach is based on Natural Language Processing (NLP) learning techniques, a rule engine and a reasoner. This process permits to better understand the user requirements, to predict other information about the requirements and to derive an appropriate service (software application as a combination of several services) in the SO-DSPL application engineering phase. We use the Smart Home product line and a dataset of textual user requirements to evaluate our proposal. Notes that when we say product, we mean an application based on service compositions.**

***Index Terms*—Service-Oriented Dynamic Software Product Lines; Ontology; User requirements; Natural language processing.**

I. INTRODUCTION

The demand of customized service-based systems is constantly increasing from day to day. This requires tailoring and adapting a software system according to the specific customer needs. Faced with this challenge, Service Oriented Architecture (SOA) provides a promising mean for supporting continuously changing customers' needs, context and expectations, as more sophisticated software systems are connected to the Internet. However, developing reusable and dynamically reconfigurable service-based systems tailored to meet different customers' needs and contexts becomes a main challenge. To address this issue, the reuse approach has been suggested as one of the pioneer solutions. The Service Oriented Dynamic Software Product Line (SO-DSPL) technique has already been adopted as one of the most promising techniques for reuse. This framework is addressed by combining SOA with Dynamic Software Product Lines Engineering (DSPLE);

A SO-DSPL [15] is known as a family of service-oriented systems sharing a set of common features. Other than the common ones, as in Product Line Engineering, there is also a set of variable features. These features are managed according to the needs of a specific market segment and/or environment and are automatically activated and deactivated according to the running situation. SO-DSPLs support their self-adaptation to new context and requirements.

In order to get customized software, users can use their own language and their own knowledge to express freely their request. In that case, providers may not understand the client's requirements. But, such an understanding is essential to provide better products and services to consumers. Overall, facilitating the understanding between providers and customers requires a knowledge representation of customer's requirements. In a previous work [14], we are interested in SO-DSPL knowledge. In this work, we were studied the semantic relationship between the SO-DSPL knwolegde and their usability by DSPL activities. To unify the studies knowledge, we have proposed an ontology named "OntoSO-DSPL" that is developed in a modular way. OntoSO-DSPL has four modules (sub-ontologies) defined as: user context sub-ontology, service sub-ontology, DSPL sub-ontology and adaptation sub-ontology where each sub ontology is interested in covering a particular dimension. Based on this work, we have concluded that the user requirements knowledge play an important role in SPL activities, as well they influence derived SPL product. Thus, in SPLE, it is necessary to identify, document and maintain user requirements to be used by SPL activities. Therefore, there is a great need for an intelligent system able to represent, understand and interpret the user requirements from the textual request targeting a specific product in the SO-DSPL framework.

To tackle these challenges and facilitate the understanding of user requirements, a knowledge representation of the requirements is needed. For this purpose, we propose in this paper an approach that enables the interpretation of the user requirements through his/her textual request and transforms it to a formal representation (requirements structure) that will be later used by SPL activities such as, product derivation, user satisfaction analysis, product adaptation, etc.

The remainder of this paper is structured as follows. Section 2 introduces SO-DSPL engineering with a brief overview. In

 ISBN: 978-1-61208-997-3

Section 3, we motivate our contribution with the help of an example. In Section 4, we present the related works. Section 5 exposes our proposed approach and Section 6 presents an illustrative case. In Section 7, we evaluate our proposed approach. Finally, the last section concludes the paper and deals with future works.

II. Service Oriented Dynamic Software Product-line Engineering

Software Product-line engineering is a paradigm within software engineering, used to define and derive sets of similar products from reusable assets [15].The development life cycle of a product line encompasses two main processes: domain engineering and application engineering [15].While domain engineering focuses on establishing a reuse platform, application engineering is concerned with the effective reuse of assets across multiple products. Feature modeling is the main activity to represent and manage product line requirements as reusable assets by allowing users to derive customized product configurations [15]. Product configuration refers to the decision-making process of selecting an optimal set of features from the product line that comply with the feature model constraints and fulfill the product's requirements. A common visual representation for a feature model is a feature diagram. The feature diagram defines common features found in all products of the product line, known as mandatory features, and variable features that introduce variability within the product line, referred to as optional and alternative features. In addition, feature diagrams often contain cross-tree constraints: a feature can include or exclude other features by using requires or excludes constraints, respectively.

Dynamic Software Product Lines (DSPLs) provide configuration options to adjust a software system at runtime to deal with changes in the users' context [14] and Service-oriented dynamic software product lines (SO-DSPL) represent a class of DSPLs that are built on services and Service-Oriented Architectures (SOAs) [13]. Figure 1 shows a part of the smart home feature model.

III. Running Example

Performing service derivation by manual configuration and adaptation can generate inconsistency. In addition, it is difficult in the context of SO-DSPL due to the important number of features, constraints, contexts, adaptation rules and web services. Thus, describing requirements and their changes in a textual format facilitates configuration, however, it requires an automated knowledge extraction process that understand and manage user requirements, their changes and their impact on the current configuration in order to generate a corresponding configuration or adapting an existing one. To illustrate the challenges faces when configuring/ adapting based on textual requirements, we introduce a smart home system for home automation as an example of an SO-DSPL. The smart home consists of smart devices equipped with sensors and web service based actuators interconnected through a software system, which aims to automate a connected home. The smart home SO-DSPL aims to develop customized smart home products for its customers. Its customers can be an external client or a developer that needs to derive or adapt product. Based on the description of the smart home options, customers express their requests in natural language. As an example, the following request is given by a customer to express the important functionalities that must be covered by the derived product in order to satisfy his/her needs.

Req1: *In the morning, I am very lazy so I think that my smart home shall have voice command feature to operate activities from bathrooms which accelerates my morning routines. Because my children Lora and Alex are always doing something stupid, my home shall have a vacuum cleaner to clean my home floors when anything spills. Sometimes, I am very tired since I work all the day, so I prefer that the smart delivery of my home may be able to order delivery food by simple voice command.*

Requirement "Req1" is very vague and its description does not directly include the smart homes features, which makes matching more difficult. Thus, it must be processed to : 1) better present the needs, 2) predict features relative to the described requirements in order to derive the desired customer product, 3) to reuse fragments of requirements and/or to adapt previous product requirements to new ones. As a result, "Core requirements" are extracted.
A core requirements have the following structure:
<feature>+ <obligationdegree>? <goal>+ <item >* <condition>*
Where :
– * : *denotes a zero repetition to an infinite number of times repetitions.*
– + : *denotes repetition once or more number of times*.
– ? :*denotes a repetition zero or one time.*
– — :*denotes a disjunction (it signifies OR).*
– ! : *denotes a negation*

Feature is the system or the feature denoted by the requirement, obligation degree, which denotes the importance of the action that must be performed by the system or the feature, goal, which denotes the objective that must be attended by the feature/system, item, which denotes the entities affected by the goal that can be also a feature or all the system, and condition, which denotes self-adaptation triggering constraint in most cases. The condition is presented by a subject, action and additional parameters denoted by entities.

From the request "Req1", we can extract three core requirements presented in Table I: From the extracted core requirements, we derive two abstract partial products based on the DSPL associated to the feature model of Figure 1. We note that the feature "Smart delivery" is considered as an optional feature because its associated obligation degree indicates "should", which generate this two abstract partial products:
AbstractPartialProduct1 = Voice command, vacuum cleaner, smart delivery
AbstractPartialProduct2 = Voice command, vacuum cleaner

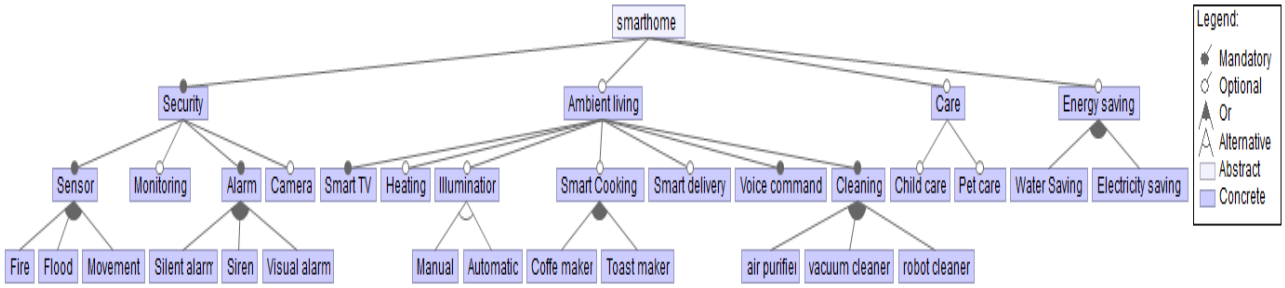

Figure. 1. Smart home SO-DSPL

TABLE I
EXTRACTED THREE CORE REQUIREMENTS

Elements	Core requirement1	Core requirement2	Core requirement3
feature	voice command	vacuum cleaner	smart delivery
obligation degree	must		should
goal	operate	clean	order
item	activities from bathrooms	floors	food
condition		when anything spills	voice command

However, the extracted partial products are note completed since the user has been restrictive in his choices. Thus, partial products are then enriched by other recommended features to satisfy the customer, with respect to SO-DSPL constraint. Then, the extracted features are then mapped to the correspond smart home features (sensors, actuators and web services) to derive the partial smart home product and activate the corresponding services at instant t.

Product1 = [Sensor, Fire, Alarm, Visual alarm, Smart TV, Voice command, cleaning, vacuum cleaner, smart delivery, air purify, Care, child care]
Product2 = [Sensor, Fire, Alarm, Visual alarm, Smart TV, Voice command, cleaning, vacuum cleaner, vacuum cleaner, air purify]
Product3 = [Sensor, Fire, Flood, Alarm, Siren, Smart TV, Voice command, cleaning, vacuum cleaner, robot cleaner, smart delivery, air purify]

Then, the extracted features are then mapped to the correspondent smart home features (sensors(denoted by S) and web services (denoted by WS) that trigger actuators) to derive the partial smart home product and activate the corresponding services at instant t.

SO-Product1 = [Fire WS35, S_fire, S_Movement, Visual alarm WS10, Smart TV WS13, S_voice,Voice command WS1101, vacuum cleaner WS125, smart delivery WS22, air purify WS45, child care WS234]
SO-Product2 = [S_fire, Fire WS35, S_Movement, Visual alarm WS10, Smart TV WS13,S_voice, Voice command WS1101, vacuum cleaner WS125,S_Dirt_detect, air purify WS136]
SO-Product3 = [Fire WS35, S_fire, S_Movement, S_flood, Flood WS321, Alarm WS1, Siren WS132, Smart TV WS13, S_voice, Voice command WS110, vacuum cleaner WS125, robot cleaner WS127, smart delivery WS22, S_Dirt_detect, air purify WS136]

To achieve this goal, features, obligation degree, goals, items and conditions of the three core requirements are used. The selection of the features mentioned in the core requirements in a product and their deselection in another one, are managed by the obligation degree of the core requirement. For example, if the obligation degree associated to the feature is “should” then the feature is optional, while if it is equals to “must” then the feature is mandatory. Then, based on the partial features selection, the rest of the features will be selected based on the feature model of the SO- DSPL and the contextual elements that influence it. Then, web services, sensors and actuators are associated with the selected features and then selected and so the service composition of the product is derived. This process is performed in first step by using an ontology populated by the extracted core requirements components (feature, obligation degree, goal, item, condition). The populated ontology includes a set of axioms and rules that: 1) allow in one hand the derivation of the product’s service composition, and 2) the selection/ deselection of features based on the constraints of the feature model, and the constraints of contextual elements that influence the products derivation and adaptation (such as, user context, weather, etc.). For example, in the context where the user is a father, the created ontology axioms/rules recommend the feature “child care” in the product because

 ISBN: 978-1-61208-997-3

child existence in the home is deduced. Recall that our aim is to automatically derive such knowledge based on core requirements from given requests and conceptualize them in an ontology to be used to infer new knowledge and used by different SO-DSPL activities, such as contextual product recommendation, derivation and adaptation.

IV. Related works

Requirements management in software reuse paradigms is supported by NLP in numerous tasks. There is an important set of well known techniques, used throughout the entire computer science field for these purposes, from which we can name Part Of Speech tagging [16] (POS), Name Entity Recognition (NER) [16], chunking [17]and types dependencies [18]. In the literature, we can find several proposals that try to combine different requirement engineering tasks with NLP tools. Some papers aim towards automatic or assisted generation of models (such as UML model) [1], [2], [3] and [21], some others try to extract requirements from free text documents [4], [4] and [6], in other cases authors look to improve and extend requirements documents [7] and [8] and some even try to use NLP to manage reusable software artifacts and requirements documents [9], [10] and [11]. For instance, In [25], the authors propose an approach to the interpretation, organization, and management of textual requirements through the use of application-specific ontologies and natural language processing.

Particularly, in SPLE, few works [6], [5], [12] are interested in requirement management using NLP tools. Thus, each requirements source varies from proposal to proposal, using as input software requirements in requirements specifications or extracted, for example, from public software project descriptions and even user opinions from download sites. Following, these requirements are pre-analized and semantically pre-processed with the objective of finding common features between them. Found features are later analyzed to detect which one of them can be represented as a feature model.

In [6], the authors propose a natural language processing approach based on contrastive analysis to identify commonalities and variabilities from the brochures of a group of vendors. In [12], the authors propose a semi-automated approach to extract features for reuse of Natural Language requirements. This approach uses the techniques from IR Information Retrieval and NLP. Latent Semantic Analysis with Singular Value Decomposition has been used to find similar review documents. This is followed by applying various clustering algorithms to cluster similar review documents. In [5], the authors present a semi-automatic approach for feature identification in the existing specifications. This is done by lexical analysis methods. Arias et al. have proposed a Framework for Managing Requirements of Software Product Lines in [13]. The proposed approach aims to define a working framework that allows structuring reusable artifacts of a software product line and retrieving them when new requirements arise. The proposal is composed of five steps, starting from elicited requirements to be pre-processed and divided into elemental pieces. Then, pre-processed requirements are semantically expanded and used to retrieve software artifacts. Then, the final product is a list of reusable software artifacts.

Based on the studied works, we conclude that requirements are used as input to be managed and to extract feature models. However, the success of requirements-based approaches is related to their relevance,the knowledge that they conceptualize and their ability to derive other relevant knowledge. Thus, the activity of requirement recognition to derive knowledge is very important. Such a user expresses his requirements with different forms in order to be satisfied by a product.To tackle this challenge, we propose in this work a framework for Core requirement Recognition from user requirement and their use to infer relevant knowledge to derive service-oriented product. The extracted core requirements are used then to populate the part of user context sub-ontology [14]in order to exploit the relationship between the requirements and other SO-DSPL concepts to infer new relevant knowledge.

V. Proposed framework for SO-DSPL Knowledge Extraction

In previous works we have defined an ontology for SO-DSPL [14] named “OntoSO-DSPL” that provides a knowledge capitalization in SO-DSPL framework by structuring, unifying, reasoning, disseminating SO-DSPL data and to be used in SO-DSPL activities such as services recommendation and selection, dynamic adaptation, runtime variability management and SO-DSPL context reasoning. Thus, the ontology should harmonize the SO-DSPL terminology and help engineers, configurators, and researchers to configure products, build and propose approaches that address the SO-DSPL’s activities. In addition, our proposed ontology includes swrl rules that are responsible for inferring new knowledge and providing new facts through its reasoning capabilities. In this work, we used a fragment of the proposed ontology which includes concepts, data objects, data properties impacted by the user requirement knowledge, that we present by Figure 2. The knowledge extraction starts by the derivation of core requirements from textual user requirements. The obtained result is used by the reasoner engine to infer new knowledge and derive SO-DSPL products. As Figure 3 shows, the SO-DSPL knowledge extraction process is composed of three steps: 1) Requirements pre treatment and token extraction 2) Mapping of the extracted token and the core requirements elements by linguistic rules and core requirements derivation, 3) SO-DSPL ontology population and knowledge inferring. As we have mentioned in section 3, we define a core requirement as a product requirement that have the Following pattern:
CoreReq= <feature>+ <obligationdegree>? <goal>+ <item >* <condition>*

Our input is a product request represented as an informal text. The process of recognition and extraction of core requirements from this text involves three modules as shown in Figure3. The first module is an initialization one. It consists of request parsing and analyzing where grammatical information

 ISBN: 978-1-61208-997-3

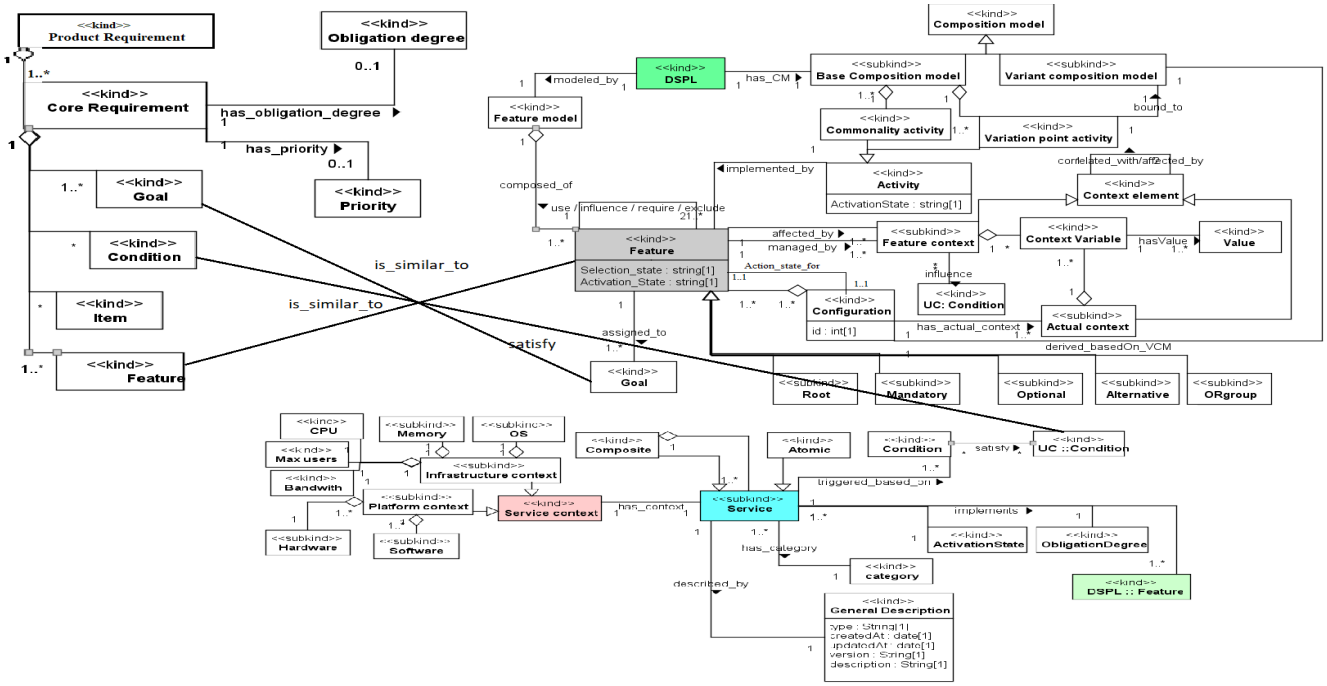

Figure. 2. OntoSO-DSPL meta-model

(ie part of speech (POS) [16]) and syntax information (ie type dependencies) [18]) are generated for each sentence) with the appropriate tokens. Many parsers, such as Stanford parsers and parsers in NLTK(Natural Language Toolkit),have been developed to recognize sentences and determine their corresponding parse trees. Based on the experimental evaluation performed in [14], the Stanford analyzer (which has also been integrated into NLTK) can perform better than various existing analyzers. We used Stanford Parser in this module to analyze the user requirements. We note that we can extract more than one core requirement from a product request.

The second module consists in the mapping between the pretreated sentence results and the core requirement by identifying the corresponding element to each requirement token. In the last module, the core requirement is automatically derived in accordance with the core requirement template. Notes that it can be a simple core requirement or a combination of several ones as previously shown in the running example.

A. *Pre-treatment phase*

We start with request segmentation with a tokenization and lemmatization process for morphological and syntactic analysis. A tokenization is the result of parsing a document down to its atomic elements named tokens. To this end, each token is labeled with PoS as a noun, verb, adverb, etc. Their dependencies are then analyzed. These two activities are performed using two natural language processing methods that are part of speech (PoS) and dependency analysis. The result is then passed in a " Phrase chunking " phase, which consists in segmenting and separating a sentence into its sub-constituents, such as noun, verb, and prepositional phrases.

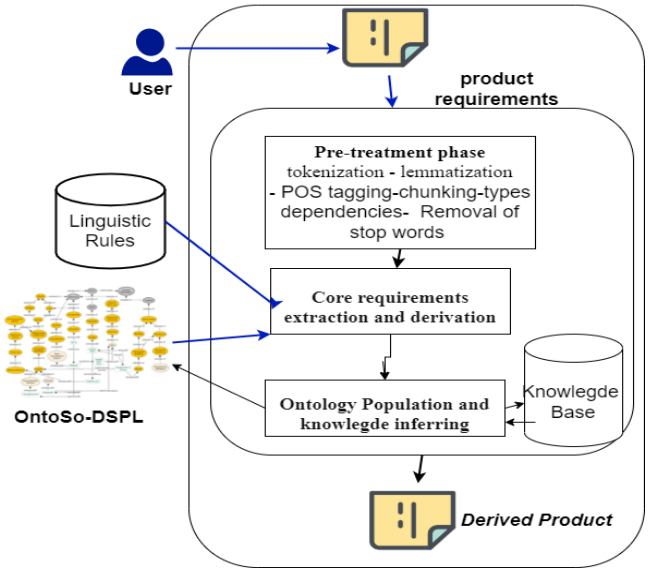

Figure. 3. SO-DSPL Knowledge Extraction Framework architecture

We built a few filter rules based on POS tagging to extract the appropriate content from the client's request. This is accomplished by creating a shallow parser. The latter parses the first step's tagged output chunk by chunk and word by word, applying filter rules to extract relevant content based on the chunk tag and the grammatical category (verb, singular noun, etc.) of the words that make up the chunk.

We maintain just Noun Chunks (NC), Verb Complex (VC), specifically those composed of infinitive verbs, ADJeCtive

chunks (ADJC) and Prepositional Chunk (PC)(starting by the preposition IN followed by a Proper Noun NP). This stage produced a list of relevant components, each of which is made up of one or more words. After that, stop words (the most common words in a language and based on the analysis of the dataset) were removed from these chunks. This final step was again accomplished through the use of a filter rule, but this time by extracting the common terms from each chunk and preserving them as relevant components.

B. Core requirements extraction and derivation

To associate each token of the chunks results to its accorded Core requirement form element, we have compiled a list of linguistic rules that cover the most possible cases. Each rule refers to: 1) the grammatical category of the token (the used "PoS" are presented in Table II), 2) its linguistic environment that is the series of units that precede and follow it and 3) its typed dependencies with the other tokens. By applying the suitable rule for the token, we can associate it with the corresponding core requirement element and then, a textual core requirement will be derived based on its corresponding pattern. Moreover, a certainty factor is assigned to each rule serving later as a degree of membership to the core requirement element attributed with the rule.

1) Core requirements extraction linguistic rules: In this step, we split each component derived from the first module into semantically coherent tokens. Secondly, we apply linguistic rules according to the grammar category of each token. By applying the appropriate rule for the token, we can link it to the corresponding element of the core requirement form. We present some of these rules in Table II.

To describe these rules, we use a set of tags which represent the Part of Speech (PoS) of a token with the combination of dependency types as shown in II. Each rule consists of an antecedent and types dependencies that must be true to execute the rule. The antecedent concerns the grammatical category of the token (PoS) and its linguistic environment (the PoS of the token in question is in bold in the Table II), that is, the series of units that precede and follow the token. The type dependencies denote dependencies between the running token (mentioned t1) and another token (mentioned t2) (a conjunction of one or more conditional statements) and a consequent (a conclusion that can be made if the conditions in the antecedent hold true). Based on our running example, we have the user requirement: "My sensor should detect movement". The chunk "My sensor" is composed of the token PRP for personal pronoun "My" and the token NN for the noun "sensor". They are followed by a modal verb "MD", and a verb "VB". Thus, by applying rule R1 from Table II, we conclude that the token "sensor"is a feature.

2) Core requirement rule's Certainty Factor : The challenge of understanding natural language writings may be fraught with uncertainty. As a result, we need to be able to deal with ambiguous reasoning. A more accurate representation of knowledge needs to assign a weight to each rule. This weight can be interpreted as an evaluation measure of a rule of its correctness and pertinence.

Inspired by the Shortliffe's research [19], where the relation between the antecedents and the consequent of the rule is measured by a certainty factor which is associated with each rule, we propose to automatically compute this factor. It represents uncertainty.

Its value is greater when there is a close relation between the antecedents and the consequent. Thus, a factor CF is assigned to a rule. To calculate such weights, we conducted an empirical study of the set of linguistic rules on a dataset of users requirements [20]. It consists in running our system on the dataset and preserving the history of application of each rule for all the examples in test. Then, we calculate the measures of belief and disbelief, defined by experts, for each rule with (1) and 2 respectively:

$$MB(Ri) = \frac{NCA}{NA} \tag{1}$$

with NCA:he number of correct applications of the rule Ri
NA:total number of applications of the rule Ri

$$MD(Ri) = \frac{NWA}{NA} \tag{2}$$

with NWA : the number of wrong applications of the rule Ri

$$MB(Ri) + MD(Ri) = 1 \tag{3}$$

MB(Ri) means that the rule Ri leads to a correct classification: the token can really be considered as an instance of a core requirement element, if not, we are talking about MD (Ri) (see (3)). The rules that are responsible only for a small number of correct classification can be deleted from the rule base because they are covering the exceptions in the dataset. An expert can specify a percentage value that has to be reached by each rule to remain in the rule base.

To select the relevant rules, we have fixed a minimum threshold of belief in the truth of a rule at 30 % (CF=0.3) based on methods proposed by Michael Hannon in [24]. Thus, we have trying different thresholds in the interval of [0.2 .. 0.5] to fix the threshold that maximize the precision and the recall of our proposed approach. Therefore, all the rules with a CF below this threshold are deleted because they deal with particular cases and helps to improve the error rate of the system.

After deleting uninteresting rules, we now have a set of linguistic rules each of which encompass an evaluation weight leading to deriving a textual core requirement and core requirement's concepts of the user context ontology by tokens extracted from a user's requirement. The majority of CF values of the remaining rules such indicated in Table II, do not exceed the 94% value. This is due to the fact that we deal with vague text input by clients.

 ISBN: 978-1-61208-997-3

TABLE II
AN EXTRACT OF LINGUISTIC RULES AND THEIR CF MEANS CERTAINTY FACTOR

Rule	Principal Pos	Dependency type antecedent	Grammar antecedent	Description	CF
R1	NN	Nsubj(t2, t1)	(PRP?|JJ*)| (DT?|JJ*) NN(MD|VB| CC)	If the token t1 is a noun (NN), it has a dependency nsubj with a verb (t2) and the antecedent is true then the t1 is a feature	0.93
R2	NN	Conj(t1,t2)	(CC|,) (PRP?|JJ*)| (DT?|JJ*) NN(MD|VB)	If the token t1 is a noun (NN), it has a dependency Conj with a verb (t2) and antecedent is true then the t1 is a feature	0.91
R3	VB	Nsubj(t1,t2)	MD?TO? VB|VBN	If the token t1 is a verb (VB|VBN), it has a dependency nsubj with a noun (NN) the antecedent is true then t1 is a goal	0.9
R4	VB	Nsubj(t1,t2)	MD? TO? VB|VBN |VBP RP	If the token t1 is a verb (VB|VBN |VBP), it have a dependency nsubj with a noun (NN), it is by a token t2 having the type "RP" and the antecedent is true then the concatenation of t1 and t2 is a goal	0.92
R5	MD	Aux(t2,t1)	MD VB	If the token t1 is a modal Verb (MD),its followed by a verb, it has an aux dependency with a verb the antecedent is true then t1 is an obligation degree	0.94
R6	(NN|NNS)	Obj(t2,t1)	VB(PRP?|JJ*)| (DT?|JJ*)NN	If the token t1 is a noun (NN), it is included in a NP chunk,it has an obj dependency with a verb the and antecedent is true then t1 is an item	0.92
R7	NN	Nsubj(t2, t1)	(PRP?|JJ*)| (DT?|JJ*)NN (MD?) (VB?|VBZ)VBN (IN)	If the token t1 is a noun (NN), it has a dependency nsubj with a verb (t2) and the antecedent is true then the t1 is an Item	0.3
R8	(PRP|NN |NNS)	Nsubj(t1,t2)	IN (PRP?|JJ*)| (DT?|JJ*) PRP|NN|NNS (VB|CC)	If the token t1 is a noun or a possessive pronoun (PRP),it is proceeded by an "IN" that is equals to if/ when/where, it has a Nsubj dependency with a verb and the antecedent is true then t1 is a Subject of a condition	0.89
R9	(VB|VPB |VBZ)	mark(t1,t2)	VB?|VPB RP?|VBZ RB	If the token t1 is a verb, it has a mark dependency with a preposition (IN) and the antecedent is true then t1 is an action of a condition	0.92
R10	(NN|NNS)	advmod(t2,1) obj (t2, t1) obl(t2, t1)	(VB|VPB RP?|VBZ RB) DT? JJS? JJ* NP? NN|NNS	If the token t1 is a noun, is preceded at the position p-i by "IN" that is equals to if/ when/where, it has a advmod, obj and obl dependency with a verb then t1 is an entity of a condition	0.82

C. Ontology Population and Knowledge inferring

We intend, in this step, to populate the ontology with the new core requirements instance and derive new knowledge based on the extracted core requirements and knowledge that already exists to conceptualize the running SO-DSPL. As input of this step, we have the list of core requirements components extracted in the previous step to create the core requirement. In the first step, core requirement's concepts are instantiated. Then, with the given input, we start by creating a relationship between the CoreRequirement instance and the ObligationDegree instance. Next, we create instances of Composition relationship between CoreRequirement instance and Goal instances, between CoreRequirement instance and feature instances, between CoreRequirement instance and item instances and between CoreRequirement instance and Condition instances, as many as there are instances in Goals,items, features and condition.

We note that users can express their desired options(i.e. feature) by different terms that are different from the terms used to express DSPL features, thus, for the mapping between the two terms we use similarity to identify the associated DSPL feature (For example vacuum cleaner can be named robot cleaner or Aspirateur Vacuum Cleaner). As well, since core requirement's condition presents the situation to trigger the goal, matching core requirement's condition and the service condition aims to select the WS that satisfy the user core requirement. Thus, we calculate similarity between core requiment's condition and WS condition; if the similarity result is higher than a threshold fixed after a similarity analyses, the web service must be selected in the derived product. to attend our objectif, we use Cosine similarity [22] (eq4) to calculate the mentioned instances, and " sentence transformer" method for sentence embedding [23] (it is the technique to transform (map) words of a language into vectors of real numbers)following its accuracy and its usefulness in measuring similarity between sentence [23]

$$CosineSimilarity = \frac{A.B}{||A||.||B||} \quad (4)$$

where A and B are vectors.

Based on the calculated similarity, we create instances of the similarity concept. The similarity concept denotes the similarity between the core requirement's feature and the DSPL feature in one hand, between core requirement's goal and DSPL goal, and between the core requirement's condition and the service condition in another hand. Then,

relationships between similarity instances and the associated instances are created. For instance, a core requirement feature "has-a-similarity" "calculated-with" the DSPL feature. A core requirement condition "has-a-similarity" "calculated-with" the web service condition. A core requirement goal "has-a-similarity" "calculated-with" the web DSPL goal.

In a second step, the created instance and the semantic relation that relate them are used to infer new SO-DSPL knowledge using SWRL rules mentioned in [14]. In addition to the rules created in [14], we have enriched our ontology with new rules presented in Table III. Based on the instanced concepts and the execution of the SWRL rules, products are derived based on the inferred knowledge.

VI. Illustrative Case

All along the present study, many experiments have been fulfilled to evaluate the applicability and the feasibility of our proposed approach to extract core requirement, populate the ontology and derive a service-based product. In this section, we consider an illustrative case which belongs to a product requirement given by a user and in the context of our running example.

In the remainder of this section, we will show the results of applying the different steps of our approach to the selected example: *"My sensor should detect movement. if I get up late night, lights in my house should turn on to enhance convenience and safety"*.

In the first step of the first module, stop words and empty words are removed then the input text is analyzed with Stanford, the mentioned type of chunks are extracted and the analyzer generates their relative POS tags, and typed dependencies of the requirement's token. The second step applies filter rules. Thus, we obtain the following chunks: my sensor, should, detect, movement. If, I, get up, late night, light, in my house, should turn on, to enhance convenience and safety. Table IV presents the chunks, POS tag and types dependencies result given by stanford analyser. In the second phase we apply linguistic rules to match each token with the corresponding concept of the ontology. On the other hand, we derive the product core requirement as a textual form.

Based on the PoS, Chunk and the types dependencies of each token, we choose the category of linguistic rules to be applied.

From this product requirement, these two core requirements are extracted as a textual form:
Core requirement 1: sensor should detect movement
Core requirement 2: lights should turn on if I get up night

The associated product core requirements became: Sensor should detect movement. Lights should turn on if I get up night.

Then, the OntoSO-DSPL ontology is population and the correspond concepts are instanced. Thus, the mentioned SWRL rules are executed, products are derived and web services are selected.

VII. Evaluation

To validate our approach, we have implemented a tool that supports our proposed approach steps including the linguistic rules. OWL (Web Ontology Language) is used to populate the proposed SO-DSPL ontology using data from the input client's product requirements. A series of experiments were performed via the implemented tool in order to validate our work. We first present in Subsect. 6.1, the dataset used. We then highlight in Subsect. 6.2 the selected evaluation measures to perform the evaluation step.

A. Dataset

To evaluate the performance of our implemented approach, we applied our approach the smart home requirements data set [20]. In order to evaluate the performance of our approach on a larger volume of data, we have augmented the existed dataset using data Augmentation algorithms [23], which consists in altering an existing data to create a new one. The objective is to augment the dataset by generating new product requirements with the same meaning as the existing requirements but written in another form. Thus, we have used "nlpaug" libraries using the Substitution by contextual word embeddings RoBERTA technique. The result data collection consists of 9000 textual product requirements. This collection of examples contains different informal texts that express product requirements in different manners.

B. Evaluation measures

In our experiments, we used recall and precision to evaluate our approach. These measures fit well to our objective which consists in identifying core requirements. It should be noted that the set of customers requests were examined to extract different core requirements. This result was compared to the output of our proposed component. For this purpose, we adopted the evaluation measures to our context of work, which we define as follows:

- Precision represents the number of found relevant instances of concepts or relationships between concepts among the found instances.
- Recall represents the number of found relevant instances of concepts or relationships among all the relevant instances to create an intention instance (detected by the expert).

Precision and recall are calculated using the formula 5 and 6 respectively:

$$Precision = \frac{TP}{TP + FP} \tag{5}$$

$$Recall = \frac{TP}{TP + FN} \tag{6}$$

where:

- TP (true positives): are the correct identification of core requirement elements by the core requirement Recognition component.

 ISBN: 978-1-61208-997-3

TABLE III
SOME SWRL RULES (THERE ARE SEVERAL OTHER RULES BUT SPACE LIMITATION)

ID	SWRL Rule	Description
R1	uc:Feature(?f1) $\wedge dspl:Feature(?f2) \wedge Similarity(?s)$ ∧has-a-similarity(?f1, ?s) $\wedge similarity-calculated-with(?s,?f2) \wedge$ similarityValue(?s, ?v) $\wedge swrlb:greaterThan(?v,80)$ -> is-similar-to(?f1, ?f2)	if the similarity of a core requirement feature and a DSPL feature is greater than 80 then the two feature are similar.
R2	uc:goal(?g1) $\wedge dspl:goal(?g2) \wedge Similarity(?s)$ ∧has-a-similarity(?g1, ?s) $\wedge similarity-calculated-with(?s,?g2) \wedge$ similarityValue(?s, ?v) $\wedge swrlb:greaterThan(?v,80)$ -> is-similar-to(?g1, ?g2)	if the similarity of a core requirement goal and a DSPL goal is greater than 80 then the two goal are similar.
R3	dspl:configuration(?cf) $\wedge uc:coreRequirement(?cr) \wedge$ satisfy(?cf,?uc) $\wedge uc:condition(?c1) \wedge$ is-composed-of(?cr,?c1) $\wedge ws:condition(?c2) \wedge$ triggered-based-on(?s,?c2) $\wedge is-similar-to(?c1,?c2)$ ->selected(?s,true) $\wedge satisfy(?c1,?c2) \wedge implement(?s,?f)$ $\wedge composed-of(?cf,?f)$	if the similarity of a core requirement condition and a web service condition is greater than 75 then the two condition are similar.
R4	dspl:configuration(?cf) $\wedge composed-of(?cf,f1) \wedge$ recommended-with(?f1,?f2) -> composed-of(?cf,?f2)	if a feature is selected in a configuration and it recommends another feature (with the semantic relation "recommended-with" then the recommended feature is with be selected in the running configuration.
R5	swrlx:makeOWLThing(?S, ?y) $\wedge dspl:Feature(?y)) ->$ implements(?y, ?S)) $\wedge ws:Service(?S)$	For each dspl feature an individual service is created and related by the relationship "implements" to execute its functionalities.
R6	swrlx:makeOWLThing(?f, ?c) $\wedge dspl:Feature(?F) ->$ composed-of(?f, ?c) $\wedge Configuration(?c)$	each configuration must be composed by more than one feature.
R7	uc:CoreRequirement(?CR1) $\wedge uc:CoreRequirement(?CR2)$ $\wedge uc:ProductRequirement(?PR)$ $\wedge uc:composed-of(?PR,?CR1) \wedge$ uc:composed-of(?PR,?CR2) $\wedge has-priority(?CR1,Desirable) \wedge$ has-priority(?CR2, Essential) $\wedge dspl:Feature(?F1)$ $\wedge dspl:Feature(?F2) \wedge dspl:alternative-with(?F2,?F1)$ $\wedge dspl:satisfy(?F1,?CR1)$ $\wedge dspl:satisfy(?F2,?CR2) \wedge dspl:Configuration(?CF)$ $\wedge satisfy(?CF,?PR) \wedge dspl:composed-of(?CF,?F1)$ -> dspl:composed-of(?CF,?F2) $\wedge dspl:selected-for(?CF,?F2)$ $\wedge dspl:eliminated-for(?CF,?F1)$	The features associated to a product's core requirements and their are related with the relationship "alternative-with" then an adaptation is triggered to the running configuration by eliminating the optional feature and selecting the essential one.
R8	uc:CoreRequirement(?CR1) $\wedge uc:CoreRequirement(?CR2)$ $\wedge uc:ProductRequirement(?PR)$ $\wedge uc:composed-of(?PR,?CR1) \wedge$ uc:composed-of(?PR,?CR2) $\wedge has-priority(?CR1,Essential) \wedge$ has-priority(?CR2, Essential) $\wedge dspl:Feature(?F1)$ $\wedge dspl:Feature(?F2) \wedge dspl:alternative-with(?F2,?F1)$ $\wedge dspl:satisfy(?F1,?CR1)$ $\wedge dspl:satisfy(?F2,?CR2) \wedge dspl:Configuration(?CF)$ $\wedge satisfy(?CF,?PR) \wedge dspl:composed-of(?CF,?F1)$ -> uc:alternative-conflict(?CR1,?CR2)	The features associated to a product's core requirements and their are related with the relationship "alternative-with" and the two core requirements are essential then a conflict of alternative constraint violation is detected.

- FP (false positives): are the wrong identification of core requirement elements by the core requirement Recognition component.
- FN (false negatives): are the core requirement elements that have not been extracted by the core requirement Recognition component.

C. Experimental Results and Analysis

Table V shows the results across the dataset. We achieve precision scores of up to 89.87 % and recall scores of up to 92.85% by applying the population process while taking into account all the rules. In fact, by analyzing the outputs of our approach corresponding to each client's request in the dataset, errors can be found if the user use a long phrase to express the condition of the requirement. These results can further be improved by intervening, this time, in the first phase of the approach. In fact, as customers freely express their request and requirement, they, sometimes, do not respect the writing rules.

Therefore, the same content can be interpreted differently by the analyzer. For example "4 PM" and "4PM " without spaces, are labeled differently by this chunker. For the first one, it tags the token 4 by the PoS "CD" and by "$" for the "$", but for the second, it considers the whole as "CD" Pos.This type of

TABLE IV
CHUNKS, POS TAG AND TYPES DEPENDENCIES

Chunks	POS tag	Types dependencies
(ROOT (S (NP (PRPMy)($NNsensor$)) (VP (MD should) (VP (VB detect) (NP (NN movement)))) (. .)))	nmod:poss(sensor-2, My-1) nsubj(detect-4, sensor-2) aux(detect-4, should-3) root(ROOT-0, detect-4) obj(detect-4, movement-5) mark(get-3, If-1)	nmod:poss(sensor-2, My-1) nsubj(detect-4, sensor-2) aux(detect-4, should-3) root(ROOT-0, detect-4) obj(detect-4, movement-5) mark(get-3, If-1)
(ROOT (S (SBAR (IN If) (S (NP (PRP i)) (VP (VBP get) (PRT (RP up)) (NP (NP (JJ late) (NN night) (NNS lights)) (PP (IN in) (NP (PRPmy)($NNhouse$))))))) (VP (MD should) (VP (VB turn) (PRT (RP on)) (S (VP (TO to) (VP (VB enhance) (NP (NN convenience) (CC and) (NN safety)))))))	nsubj(get-3, i-2) csubj(turn-12, get-3) compound:prt(get-3, up-4) amod(lights-7, late-5) compound(lights-7, night-6) obj(get-3, lights-7) case(house-10, in-8) nmod:poss(house-10, my-9) nmod(lights-7, house-10) aux(turn-12, should-11) root(ROOT-0, turn-12) compound:prt(turn-12, on-13) mark(enhance-15, to-14) xcomp(turn-12, enhance-15) obj(enhance-15, convenience-16) cc(safety-18, and-17) conj(convenience-16, safety-18)	nsubj(get-3, i-2) csubj(turn-12, get-3) compound:prt(get-3, up-4) amod(lights-7, late-5) compound(lights-7, night-6) obj(get-3, lights-7) case(house-10, in-8) nmod:poss(house-10, my-9) nmod(lights-7, house-10) aux(turn-12, should-11) root(ROOT-0, turn-12) compound:prt(turn-12, on-13) mark(enhance-15, to-14) xcomp(turn-12, enhance-15) obj(enhance-15, convenience-16) cc(safety-18, and-17) conj(convenience-16, safety-18)

TABLE V
RECALL AND PRECISION OF THE OVERALL APPROACH

	TP	FP	FN	Precision	Recall
DataSet	55400	6241	4261	89.87%	92.85%

tagging error can influence the entire instantiation process. A possible contribution to the improvement of results would be, therefore, to have a text filtering or rectification stage of each input according to the NLP rules.

VIII. CONCLUSION

In order to extract information and knowledge from user requirements and for being able to derive the most appropriate product in the context of a SO-DSPL, we have presented an approach that analyses and understands automatically the user requirement. We reuse the structure of user requirements from SO-DSPL [14]ontology and benefit from the use of NLP algorithms. Indeed, the proposed approach extracts user requirements and builds a requirement in accordance with the core requirement structure. This latter is defined by the basic requirements that must be covered by the derived product to satisfy the user. Our recognition approach is based on a set of linguistic rules and the support of uncertainty. These rules facilitate the building of core requirement structure which is then loaded as an instance of the SO-DSPL ontology. Based on the derived core requirements, relevant knowledge are inferred and relevant services are selected to derive the entire user product. The originality of the proposed approach is, on the one hand, that the entire process of the extraction and population approach is made automatically and in a semantic and intelligent way thanks to ontology reasoning capabilities. This will enhance SPL activities such as: product recommendation, user satisfaction, feature extraction and service (DSPL product) adaptation. On the other hand, it can be applied both on a short and a long text. Our near future work is to continue the experimentation of the approach and to enrich it with linguistic rules that allow the extraction of context changes. In a second step, we aim to combine this work with recommender system that will derive the appropriate services for the generated requirement.

REFERENCES

[1] B. Tanmay, N. Nan, S. Juha and M. Anas, "Leveraging topic modeling and part-of-speech tagging to support combinational creativity in requirements engineering", IN Requirements Engineering, vol. 20, no. 3, pp. 253–280, 2015.

[2] G. Sarita and C. Tanupriya, "An efficient automated design to generate uml diagram from natural language specifications," In Cloud System and Big Data Engineering (Confluence), 2016 6th International Conference, pp. 641–648, IEEE, 2016.

[3] Y. Mu,Y. Wang, and J. Guo, "Extracting software functional requirements from free text documents," in Information and Multimedia Technology, 2009. ICIMT'09. International Conference on, pp. 194–198, IEEE, 2009

[4] C. Arora, M. Sabetzadeh, L. Briand,and F. Zimmer, "Automated extraction and clustering of requirements glossary terms," IEEE Transactions on Software Engineering, 2016.

[5] E. Boutkova, and F. Houdek, "Semi-automatic identification of features in requirement specifications," in 2011 IEEE 19th International Requirements Engineering Conference, pp. 313–318, Aug 2011.

[6] A. Ferrari, G. Spagnolo, and F. Dell'Orletta, "Mining commonalities and variabilities from natural language documents," in Proceedings of the 17th International Software Product Line Conference, pp. 116–120, ACM, 2013.
[7] S. J. Korner and T. Brumm, "Natural language specification improvement with ontologies," International Journal of Semantic Computing, vol. 3, no. 04, pp. 445–470, 2009.
[8] Y. Wang, "Semantic information extraction for software requirements using semantic role labeling," in Progress in Informatics and Computing (PIC), 2015 IEEE International Conference on, pp. 332–337, IEEE, 2015.
[9] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Improving ir-based traceability recovery via noun-based indexing of software artifacts," Journal of Software: Evolution and Process, vol. 25, no. 7, pp. 743–762, 2013.
[10] G. J. Hahm, M. Y. Yi, J. H. Lee and H. Suh, "A personalized query expansion approach for engineering document retrieval," Advanced Engineering Informatics, vol. 28, no. 4, pp. 344–359, 2014.
[11] S.-P Ma, C.-H. Li Tsai and C.Lan, "Web service discovery using lexical and semantic query expansion," in e-Business Engineering (ICEBE), 2013 IEEE 10th International Conference on, pp. 423–428, IEEE, 2013
[12] N. H. Bakar, Z. M. Kasirun, N. Salleh and H. A Jalab, "Extracting features from online software reviews to aid requirements reuse," Applied Soft Computing, vol. 49, pp. 1297–1315, 2016
[13] M. Arias, A. Buccella and A. Cechich, "A Framework for Managing Requirements of Software Product Lines", Electronic Notes in Theoretical Computer Science, vol. 339, pp. 5-20, 2018
[14] N. Maalaoui, R. beltaifa, L. Labed and R. Mazo, "An Ontology for Service-Oriented Dynamic Software Product Lines Knowledge Management",16th International Conference on Evaluation of Novel Approaches to Software Engineering, vol. 314, pp 314-322, 2021
[15] R. Capilla,J. Bosch,P. Trinidad,A. Ruiz-Cortes and M. Hinchey, "Overview of Dynamic Software Product Line Architectures and Techniques:Observations from Research and Industry",The Journal of Systems and Software,pp 3-23, 2014
[16] C.D. Manning, "Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics?". In: Gelbukh A.F. (eds) Computational Linguistics and Intelligent Text Processing. CICLing 2011. Lecture Notes in Computer Science, vol 6608. Springer, Berlin, Heidelberg. 2011
[17] A. Mansouri, L.S. Affendey and,A. Mamat, "Named Entity Recognition Approaches",IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.2, February 2008
[18] H. Bo, P. Cook, T. Baldwin, "Lexical normalization for social media text".ACM Transactions on Intelligent Systems and Technology Volume Article No.: 5pp 1–27, 2013
[19] D. Dubois,J. Lang and H. PradeFuzzy, sets in approximate reasoning, Part 2: logical approaches. In "Fuzzy Sets and Systems",pp 203-244,1992
[20] K. Pradeep,A. Nirav, and P. Munindar, "Acquiring Creative Requirements from the Crowd: Understanding the Influences of Personality and Creative Potential in Crowd RE".Proceedings of the IEEE 24th International Requirements Engineering Conference (RE), pp 176–185., September 2016,
[21] A. Arellano, E. Carney, and M.A. Austin, "Natural Language Processing of Textual Requirements," The Tenth International Conference on Systems (ICONS 2015),pp. 93– 97, Barcelona 2015
[22] S. Pinky, P.a. Kritish, T. Pujan and S. Subarna," Comparison of Semantic Similarity Methods for Maximum Human Interpretability",In IEEE, 2019
[23] https://github.com/makcedward/nlpaug,Online; accessed 12-july-2022
[24] M. Hannon, A solution to knowledge's threshold problem. Philosophical Studies: An International Journal for Philosophy in the Analytic Tradition, 174(3), 607–629. http://www.jstor.org/stable/26001716, 2017
[25] A. Arellano, E. Zontek-Carney, A. Austin, Frameworks for Natural Language Processing of Textual Requirements. International Journal on Advances in Systems and Measurements. 8. 230-240,2015

 ISBN: 978-1-61208-997-3