From Natural Language Requirements to Formal Specification Using an Ontology

Driss Sadoun, Catherine Dubois, Yacine Ghamri-Doudane, Brigitte Grau

From Natural Language Requirements to Formal Specification using an Ontology

Driss Sadoun*†, Catherine Dubois‡§, Yacine Ghamri-Doudane‡¶, Brigitte Grau*‡
*LIMSI/CNRS B.P. 133 91403 Orsay Cedex, France
†Université Paris-Sud, 91400 Orsay, France
‡ENSIIE, 1 square de la résistance, 91000 Evry, France
§CNAM-CEDRIC 292 Rue St Martin FR-75141 Paris Cedex 03, France
¶(LIGM) Université Paris-Est-Marne-la-Vallée, 75420 Champs sur Marne, France

***Abstract*—In order to check requirement specifications written in natural language, we have chosen to model domain knowledge through an ontology and to formally represent user requirements by its population. Our approach of ontology population focuses on instance property identification from texts. We do so using extraction rules automatically acquired from a training corpus and a bootstrapping terminology. These rules aim at identifying instance property mentions represented by triples of terms, using lexical, syntactic and semantic levels of analysis. They are generated from recurrent syntactic paths between terms denoting instances of concepts and properties. We show how centring on instance property identification allows us to precisely identify concept instances explicitly or implicitly mentioned in texts.**

***Keywords*-Knowledge representation, Ontology population, Ontology reasoning, Requirement specification.**

I. Introduction

Any software system development process is based on a primary phase involving requirement specifications. Usually, formal methods only apply to formal specifications in order to verify their correctness and consistency. However, experience shows that requirement specifications are mostly written in natural language (NL) and thus are informal. Their verification then requires them to be transformed into formal specifications. Hence, it is quite natural to question the possibility of automatizing the transformation of NL requirement specifications into formal requirement specifications. This now long-standing issue has been addressed in different ways [Gordon and Harel, 2009], [Kof, 2010], [Njonko and El Abed, 2012]. These studies underline the common difficulty in obtaining a direct transformation and the need of an intermediate model to bridge the gap between NL specifications and formal specifications. The generally proposed intermediate representations do not check the consistency of what they model, such as for message sequence charts (MSCs) [Kof, 2010], live sequence charts (LSCs) [Gordon and Harel, 2009] or Semantics of Business Vocabulary and Business Rules (SBVR) [Bajwa et al., 2012].

We have chosen an OWL ontology to model formally the domain knowledge and to represent the requirement specifications by its population so that its formal context and its reasoning capability allows us to make verifications at an early stage. An ontology models concepts and properties, defining the conceptual vocabulary of a domain and gives a formal framework which associates semantics to terms in NL. The joining of concepts and properties with their formulations in texts is done by using a lexical ontology in Simple Knowledge Organization System (SKOS) that contains the terminology associated to the conceptual vocabulary.

Populating an ontology consists in adding new instances, associated to concepts and properties recognized in texts, without changing its conceptual structure. Their identification may focus on the recognition of concept instances [Thongkrau and Lalitrojwong, 2012], or on the recognition of relation instances [Nakamura-Delloye, 2011]. In both cases the recognition is limited to the surface of the text, whereas the recognition of identical instances and implicit knowledge requires to infer its meaning at a semantic level.

In this article, we propose a new method which guides the ontology population process by the identification of triples in texts. These triples correspond to mentions of property instances. We distinguish two types of triples: complete and partial. Complete triples contain the mentions of a property and the two concepts it links while partial triples aim at recognizing properties in which one of the linked instances is not explicitly mentioned. Triples of terms corresponding to property instances are extracted using extraction rules formed from recurrent syntactico-semantic forms appearing in a training corpus using a bootstrapping terminology. By guiding the ontology population through the identification of property instances we can resolve ambiguities of terms and infer implicit informations in the ontology. In this way, the identification of instances of concepts not only relies on the recognition of their mentions in texts but also takes into account the conceptual properties where they are involved, providing a semantic context of interpretation.

We show how our approach allows us to reliably identify property instances, and then to infer the associated instances of concepts. Furthermore, the consistency of identified instances is checked using the OWL reasoner before the creation of consistent operation rules in order to represent the user requirements. Throughout the paper examples are taken from our application domain: smart spaces.

II. Related work

It is commonly adopted that a direct transition between NL specifications and formal specifications is not conceivable. Intermediate representations aim at restricting [Gordon and Harel, 2009] or structuring [Ilieva and Boley, 2008] the NL specifications, as well as overcoming their inherent lack of context [Bajwa et al., 2012]. However most of the proposed representations remain semi-formal. In our approach, we use an OWL ontology to formally represent the background knowledge to tackle ambiguities resulting from the mismatch between lexical and conceptual meaning.

The usual assumption for the recognition of concept and property mentions in texts is that pairs of entities appearing in the same context can be considered as instances of the same relationship. Some methods restrain the definition of the context on the presence of a verb and the recognition of its neighbourhood [Lin and Pantel, 2001], [Makki et al., 2008]. Other methods rely on the classification of pairs of entities known to be linked by a semantic relationship [Hasegawa et al., 2004], [Nakamura-Delloye, 2011], [Thongkrau and Lalitrojwong, 2012]. The majority of these approaches exploit lexical and syntactic knowledge. However the context is more likely to be captured by exploiting semantic as a third level of knowledge. Thus, we propose to acquire automatically rules based on three levels.

Extracting from text pieces of information necessary to represent requirements such as operation rules may consist in matching lexical formulation with their semantic meaning [Njonko and El Abed, 2012], [Bajwa et al., 2012]. We propose to represent these pieces by means of ontology population. Reasoning on the ontology allows us to infer the semantic classes to which instances of concepts belong. We propose to go beyond instance classification, by identifying instances representing identical individuals, through the use of unique constraints formed on instance properties. This way, the consistency of matched elements is checked.

III. Extraction rules acquisition

We use the following notations throughout the article. Concept and property names appear in *italic* and begin with a capital letter. Instances of concept names appear in *italic* and are written in lower case. A concept is noted C_i and an instance of a concept is noted i_{C_i}. Properties are defined on a domain and a range[1]. Properties linking concepts are noted $P_k(C_i, C_j)$ and an instance of a property is noted $P_k(i_{C_i}, i_{C_j})$. In texts, instances of concepts and properties are denoted by terms. Thus, mentions of an instance of a property between two instances of concepts is a triplet noted $(t_{P_k}, t_{C_i}, t_{C_j})$ with t_{P_k}, t_{C_i} and t_{C_j} terms denoting respectively instances of the property P_k and the concepts C_i and C_j.

A. Two kinds of rules

Identification of concept instances is guided by the identification of instance properties. Hence, each extraction rule aims at identifying a mention of a property defined in the ontology. We distinguish two sorts of triples. On the one hand complete triples where both mentions of domain and range of the property are explicit in the text, $(t_{P_k}, t_{C_i}, t_{C_j})$. On the other hand partial triples where one of these mentions is not explicit in the text. It is then marked as unknown in the triple, with either $(t_{P_k}, ?i, t_{C_j})$ or $(t_{P_k}, t_{C_i}, ?i)$. For example, in the sentence: *when a person moves into the kitchen, switch on the light.*, we can identify the triple *(Occurred-in, move, kitchen)* which denotes an instance of the property *Occurred-in* linking an instance of a concept $Phenomenon$ to an instance of a concept $Location$. However, the agent which has to turn on the light is not explicit. In this case the triple to be identified is *(Turn-on*[2]*,?A,light)* which denotes an instance of the property *Turn-on* linking an instance of a concept *Actuator* to an instance of a concept *Physical-process*.

B. Acquisition method of extraction rules

The acquisition of extraction rules is performed automatically by bootstrapping using a training corpus and a starting terminology. Rules are acquired from the most frequent syntactic paths between pairs of terms. These terms are issued from the semantic classes defined in the ontology. The extraction of the two sorts of triples implies the acquisition of two sorts of extraction rules. Partial triples are acquired from the most frequent paths between terms denoting instances of a property and an instance of its domain or range. Complete triples are formed using two partial paths between pairs of terms denoting an instance of a property and its domain and the same property and its range.

Algorithm 1 describes the rule acquisition process. Each sentence is analysed and its syntactic dependency tree generated. Then, three functions are called for path extraction between pairs of terms. These functions identifies paths linking three sorts of pairs: pairs of terms denoting instances of domain and range of a same property, pairs of terms denoting instances of a property and its range and pairs of terms denoting instances of a property and its domain. Extracted paths are compared according to their syntactic dependencies and the lemmatized form of their terms. Identical paths are grouped and the n most frequent paths of each sort of pairs are returned. Finally, extraction rules are generated from the features of the returned paths.

[1]The domain is a concept or a set of concepts and the range is a concept, a set of concepts or a data type.

[2]Turn-on is the preferred formulation of switch on.

Input: corpus, ontology, skos
for *each sentence* S *in corpus* **do**
  $Tree \leftarrow getDependencyTree(S)$;
  $lemT \leftarrow lemmatizeTree(Tree)$;
  $C \leftarrow getConcepts(ontology)$;
  for *each* c *in* C **do**
    $T_{Domain} \leftarrow getTerms(c, skos)$;
    $P \leftarrow getPropertiesOf(c, ontology)$;
    for *each* p *in* P **do**
      $T_{Property} \leftarrow getTerms(p, skos)$;
      $Rg \leftarrow getRangesOf(p, ontology)$;
      for *each* rg *in* Rg **do**
        $T_{Range} \leftarrow getTerms(rg, skos)$;
        // - instances of complete triples -
        extractPaths($lemT$, T_{Domain}, T_{Range});
        //- instances of partial triples -
        extractPaths($lemT$, T_{Prop}, T_{Range});
        extractPaths($lemT$, T_{Prop}, T_{Domain});
$Paths \leftarrow getMostFrequentPaths()$;
for *each* ch *in* $Paths$ **do**
  $createCorrespondingRule(ch)$;

Algorithm 1: Extraction rule acquisition

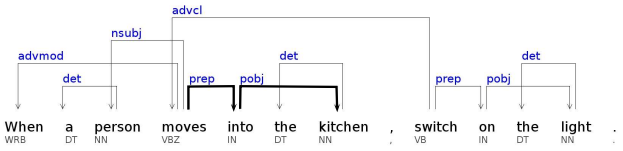

Figure 1. Syntactic dependency tree

C. Syntactic path

A syntactic path is a sequence of syntactic dependencies in a dependency tree, linking two terms that can be related by the mention of the studied property. Each node in the tree is labelled by a term and its morpho-syntactic category. Figure 1 represents the syntactic dependency tree of the sentence *When a person moves into the kitchen, switch on the light.* The dependency path between the terms *moves* denoting an instance of a concept *Phenomenon* and *kitchen* denoting an instance of a concept *Location* is $prep(moves, into) - pobj(into, kitchen)$ (bold in figure 1).

D. Generation of extraction rules

Extraction rules aim at identifying property instances. Hence, they model the semantic context in which terms denoting instances of concepts appear. The generation of extraction rules is automatically done and exploit the features of the selected paths. These features are of three forms:

1) syntactic dependencies, transformed into predicates;
2) morpho-syntactic categories.
3) termino-ontological categories (semantic features);

Example: One of the recurrent paths between the two sets of terms of the concept *Phenomenon* and *Location* which are the domain and range of the property *Occurred-in* is $prep(t_P, t_O) \wedge pobj(t_O, t_L)$, as in figure 1, thereby generating the extraction rule associated to the property *Occurred-in(Phenomenon,Location)* with $T_{Loc}, T_{Occ-in}, T_{Phen}$ three sets of terms from the lexical ontology.

$$prep(t_P, t_O) \wedge pobj(t_O, t_L) \quad (1)$$
$$\wedge isVerb(t_P) \wedge isPrep(t_O) \wedge isNoun(t_L) \quad (2)$$
$$\wedge T_{Phen}(t_P) \wedge T_{Occ-in}(t_O) \wedge T_{Loc}(t_L) \quad (3)$$
$$\rightarrow Occurred - in(t_P, t_L)$$

IV. Terminology expansion

The terminology is represented in SKOS. This formalism enables to define for each term a preferred formulation as well as a list of synonyms. In order to enlarge the starting terminology, we apply the acquired rules on the training corpus for extracting new terms. Each rule is applied three times. Each application aims at extracting terms denoting a semantic set involved in the rule. For example, let R an extraction rule and $T_{Domain}, T_{Range}, T_{Prop}$ three sets of terms. From rules that extract complete triples, we generate:
$R(T_{Domain} + T_{Prop}) \rightarrow T_{Range}$
$R(T_{Range} + T_{Prop}) \rightarrow T_{Domain}$
$R(T_{Domain} + T_{Range}) \rightarrow T_{Prop}$

When all possible rules have been applied, we compute the pertinence of a term t extracted by the rules R_i through the formula: $Pertinence(t) = (\sum_{i=1}^{n} freq(t, R_i)) * n$ with $freq$ the frequency of t relative to a rule, n the number of rules of the same type from which it was extracted. Thus, a term extracted twice by three rules has a greater score than a term extracted six times by the same rule. This formula aims to promote the terms extracted by several rules.

V. Ontology population

The aim of NL specification analysis is to recognize user requirements of a system in order to formalize them. These requirements describe the behaviour of the system as needed by the user and are created according to the property instances extracted by the extraction rules. These requirements correspond to operation rules of the form $antecedent \rightarrow consequent$. Such a rule means that if the *antecedent* hold then the *consequent* must also hold. Each of the *antecedent* and *consequent* is a conjunction of predicates that match concepts and properties modelled in the ontology. Our final aim is to check the consistency of these rules according to their interactions. However, these verifications cannot be done by an OWL reasoner, which cannot reason on different sequencing of rules or represent state changes due to OWL open world assumption. Thus we have choose to represent the operation rules in the ontology so that they can be translated into a formal specification language in order to perform model checking.

A. Requirement pattern

Analogically to LSCs [Gordon and Harel, 2009], MSCs [Kof, 2010] representations used to guide requirement identification, but bearing in mind the need for consistency, we propose to model requirement patterns formed on ontology predicates whose consistency has been previously verified. Therefore, we defined a concept named *Requirement-pattern* with two data properties, *Antecedent* and *Consequent*, the values of which being a predicate that is either a unary predicate corresponding to a concept or a binary predicate corresponding to a property. Predicate arguments are either variables (Variable names start by "?") or constants (individuals or data type values). An instance of the concept *Requirement-pattern* will have as many values for the property *Antecedent* than it has predicates. However, value of the property *Consequent* is only a binary predicate which corresponds to the property to deduce between concept instances of the antecedent. Predicates of the *Consequent* part and constants are text dependent elements that we call the dynamic part of the pattern in the rest of the paper.

Each instance of *Requirement-pattern* corresponds to a type of operation rule. These instances will guide the identification of user requirements. It consists in replacing elements of the dynamic part by the corresponding elements extracted from specifications. For example, the instance of *Requirement-pattern* illustrated in Figure 2, defines a type of operation rules which states that a phenomenon $?e$ that occurs in the zone of sensing l of a sensor $?s$ and shares the same type t, will entail the corresponding actuator $?a$ (same location and same managed type) to perform an action $Actuate\text{-}on$ on the physical-process p. Dynamic predicates that have to be retrieved from text are those of the property *Antecedent* containing the constants l or t, and for the property *Consequent* a predicate representing a sub-property of *Actuate-on* i.e. *Turn-on*, *Turn-off*, *Increase* or *Decrease*.

	Requirement-pattern	User Requirement
Antecedent	: Actuator(?a)	: Actuator(?a)
	: Sensor(?s)	: Sensor(?s)
	: Physical-process(?p)	: Physical-process(?p)
	: Phenomenon(?e)	: Phenomenon(?e)
	: Actuate-on(?a,?p)	: Actuate-on(?a,?p)
	: Occured-in(?e,**l**)	: Occured-in(?e,**kitchen**)
	: Zone-of-sensing(?s,**l**)	: Zone-of-sensing(?s,**kitchen**)
	: Fixed-in(?a,**l**)	: Fixed-in(?a,**kitchen**)
	: Has-type(?e,**t**)	: Has-type(?e,**movement**)
	: Perceived-type(?s,**t**)	: Perceived-type(?s,**movement**)
	: Managed-type(?a,**t**)	: Managed-type(?a,**movement**)
Consequent	: **Actuate-on**(?a,?p)	: **Turn-on**(?a,?p)

Figure 2. An instance of *Requirement-pattern*

B. User requirement recognition

A *user requirement* is created if one of the *requirement-pattern* can apply i.e. if all elements of the dynamic part are identified. The algorithm 2 describes the ontology population process. We apply all acquired extraction rules on each sentence of the NL specifications. Extracted property instances are then checked according to the dynamic parts of the requirement patterns. Each predicate of the dynamic parts is searched in the set of property instances. When a dynamic predicate is found, we replace its dynamic elements by the values in the corresponding property instance, then we add it to the list of found predicates. After all dynamic predicates are checked, if no predicate is missing for some types of requirement, property instances are created in the ontology. Finally, if no inconsistency occurs when reasoning on them, the user requirement is created. A user requirement is represented by an instance which has the same description as the *Requirement-pattern* where dynamic elements are replaced by arguments of a corresponding property instance.

Input: corpus, ontology, skos
for *each sentence* S *in corpus* **do**
  $PI \leftarrow ApplyExtractionRules(S)$;
  $patterns \leftarrow getReqPatterns(ontology)$;
  for *each pattern in patterns* **do**
    $P_{Fix} \leftarrow getFixPart(pattern)$;
    $P_{Dyn} \leftarrow getDynamicPart(pattern)$;
    for *each* p_{Dyn} *in* P_{Dyn} **do**
      if *not* $PI.contain(p_{Dyn})$ **then**
        $P_{missing}.add(p_{Dyn})$;
      else
        for *each* pi *in* PI **do**
          if $pi.name = p.name$ **then**
            $p_{found} \leftarrow replaceElements(p, pi)$;
            $P_{found}.add(p_{found})$;
            $PI_{associated} \leftarrow pi$;
    if $P_{missing}.isEmpty()$ **then**
      $createInstances(PI_{associated})$;
      if $Reasoning(ontology)$ **then**
        $createRequirement(P_{Fix}, P_{found})$;
      else
        $print(PI_{associated}+$ "generate inconsistences." $)$;
    else
      $print(P_{missing}+$" are missing !");

Algorithm 2: Ontology population

VI. Reasoning process

After property instances are identified, concept instances have to be classified in the ontology. OWL formal semantics allows us to do inferences on individuals according to properties in which they are involved. The reasoning process is done on the whole set of individuals of the ontology in order to infer implicit knowledge and also to detect errors. Hence, ambiguous terms are solved according to their semantic context. Classifying instances in the ontology is not enough to represent the knowledge derived from texts

in a consistent way. It is also necessary to determine which instances refer to the same individuals.

A. Instance classification

1) Domain and range of a property: Each instance participating to a property is associated to the semantic class denoted by the domain or the range of the property.

2) Necessary and sufficient conditions (NSC): Equivalences between a concept and some of its properties can be defined, forming NSC that make possible to infer the concept to which an instance belongs from its properties.

B. Instance identification

In OWL, identical instances are identified using the property *SameAs*. Inference of *SameAs* exploits the property which, like the primary key defined in databases, enables us to identify individuals in a unique way. Properties representing a unique constraint can be defined using two ways.

1) Functional properties: associate to each individual of the domain, a unique individual as a range. Let P_n a functional property, i_i, i_j and i_k three individuals.

$P_n(i_i, i_j) \wedge P_n(i_i, i_k) \rightarrow SameAs(i_j, i_k)$

An *inverse functional property* associates to each individual of the range, a unique individual as a domain. Let P_n an inverse functional property, i_i, i_j and i_k three individuals.

$P_n(i_j, i_i) \wedge P_n(i_k, i_i) \rightarrow SameAs(i_j, i_k)$

2) SWRL constraints: More than one property may be needed to represent a unique constraint, in which case the constraint has to define properties that two individuals must share to be inferred as one. These constraints are defined using SWRL rules. For example, the rule below states that two individuals i_x and i_y are a same individual if they share the same values through properties P_1 and P_2, $P_1(i_x, i_{C_1}) \wedge P_1(i_y, i_{C_1}) \wedge P_2(i_x, i_{C_2}) \wedge P_2(i_y, i_{C_2}) \rightarrow SameAs(i_x, i_y)$

VII. Experimentation

In this section, we present the application domain, the used resources, and the evaluations of term acquisition and property instance extraction.

A smart space is composed of a set of communicating objects (sensors, actuators and control processes), with a general behaviour. A sensor detects the occurrence of a phenomenon or measures a quantifiable phenomenon in a confined zone. A phenomenon to be detected or measured by a sensor must be located in the zone of sensing of this sensor and shares the same type (temperature, movement, ...) with this sensor. An actuator is connected to a physical process within the environment and can actuate it. When a phenomenon (or a set of phenomena) is measured or detected, the collected information is controlled. This control can lead to the activation of one or more actuators that will trigger one or more actions on the devices or physical processes it or they control. The possible actions are the following ones: turn on, turn off, decrease or increase. An actuator can be activated by a sensor (or a set of sensors), if it is located in the its control zone. Furthermore the actuator has the same type than this sensor (or set of sensors).

In order to model this domain we defined an ontology containing 12 concepts, 15 properties between concepts and 9 between concepts and data types, where all individuals are identifiable from their properties of location and type.

In the absence of a sufficiently large corpus of specifications of requirements in the field of smart spaces, we have constituted a training corpus with e-books from the *Anacleto Digital Library* (http://www.gutenberg.us/) covering different domains and literary styles. This diversity permits the acquisition of rules from different writing styles and thus is well-founded for the recognition of extraction rules, properties being transdomains. However its generality limits the extraction of the terminology for the specific concepts of the domain. In order to have an evaluation corpus, we developed a platform for collecting specifications. We collected approximately 80 sentences (1558 words) describing requirement specifications, that was annotated to form a gold standard.

We have acquired 126 extraction rules, including 31 for the identification of complete triples and 95 for the identification of partial triples. Obviously, the application of complete rules has priority over partial rules. Each rule has been generated from the n most frequent syntactic paths, with n fixed experimentally to 4. However, depending on the domain, mentions of certain properties are less frequent in texts. So this number may be too high and thus non pertinent paths may generate rules. In order to avoid this potential noise, we set a second limit corresponding to the number of syntactic dependencies involved in the path. Indeed, the longer a path is and thus terms are distant, the less likely that the path represents a semantic property. In our experiments, we fix the maximal number of dependencies in a path at 4.

The starting terminology contains 109 terms, 18 preferred terms and 91 synonyms. Table I illustrates the acquisition of terms on three semantic categories: phenomena, the physical processes and the possible actions. Extraction rules have been applied on the training corpus as described in Section IV. The first column indicates starting terms of the terminology, the second column, the number of different extracted terms and the third column pertinent terms which were selected by hand by examining the n first terms returned from the formula given in Section IV, with n fixed at 25%.

	Starting	**Extracted**	**Pertinent**
Phenomenon	18	965	49
Physical-process	33	625	23
Actuate-on	19	413	27

Table I
PERTINENT EXTRACTED TERMS

Results of the extraction of triple candidates are described in Table II. The first column indicates the number of instances to recognize. The next column indicates the number of correctly identified triples, next the triples not correctly identified. The last three columns describe precision, recall and F-measure respectively. Extraction is performed considering only pertinent terms from the terminology. The first line represents the results for the three properties *Located-in*, *Fixed-in* and *Occurred-in*, which associate a location to each of the concepts of *Location*, *Physical-process* and *Phenomenon* respectively. The property *Has-type* associates a *Type* to a *Phenomenon*. The last two lines represent instances of concepts *Phenomenon* and *Actuator* which are classified and identified during the reasoning process.

We clearly observe a high accuracy (0.95), which shows the pertinence of the acquired rules. Moreover, the obtained recall (0.63) is relatively high considering that rule and terminology acquisition were done on a non specific corpus. From the instances of property created in the ontology, we correctly classified 22 instances of the concept *Phenomenon* as belonging to 10 different individuals and 17 instances of the concept *Actuator* as belonging to 7 different individuals, with no incorrectly classified instances. 15 user requirements out of 42 were created automatically from identified instances. Other requirements came out with insufficient information or as inconsistent, in which case the user is asked to revise the requirements set aside.

The ontology population and the reasoning process are implemented under a Java application using Jena, a Java framework for building Semantic Web applications.

VIII. Conclusion

We described an approach for ontology population which aims at representing in a formal manner user requirements specified in texts. Our approach centres on the identification of property instance mentions in texts, using extraction rules, acquired from recurrent syntactic paths linking terms which denote concept and property instances. Rules exploit lexical, syntactic and semantic knowledge. They identify property instances even if their domain or range is implicit. Moreover, these rules permit to avoid any ambiguity of extracted terms by capturing the semantic context in which terms appear. By guiding the identification of concept instances through property instance identification, we can then recognize them in context. Thus, we do not only rely on a lexical recognition but also take into account their semantic roles. Furthermore, we demonstrate how using the inference power of OWL, allows us to classify concept instances and identify them in a unique way. The proposed approach has been designed to be independent of the domain and easily adapted to other languages. From a modelled ontology, it requires only a training corpus and a set of starting terms. In future work, we plan to apply our approach to other domains.

	Pertinent	Correct	Incorrect	P	R	F-M
Loc	115	75	9	0.89	0.65	0.75
Has-type	62	35	0	1	0.56	0.72
Actuate-on	90	51	0	1	0.56	0.71
Total	267	164	9	0.95	0.61	0.70
Phenomenon	62	22	0	1	0.35	0.51
Actuator	42	17	0	1	0.40	0.57

Table II
Extraction of triple candidates

References

[Bajwa et al., 2012] Bajwa, I. S., Lee, M., and Bordbar, B. (2012). Resolving syntactic ambiguities in natural language specification of constraints. In *Proceedings of the 13th international conference on Computational Linguistics and Intelligent Text Processing - Volume Part I*.

[Gordon and Harel, 2009] Gordon, M. and Harel, D. (2009). Generating executable scenarios from natural language. In *Proceedings of the 10th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing '09, pages 456–467, Berlin, Heidelberg. Springer-Verlag.

[Hasegawa et al., 2004] Hasegawa, T., Sekine, S., and Grishman, R. (2004). Discovering relations among named entities from large corpora. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*.

[Ilieva and Boley, 2008] Ilieva, M. and Boley, H. (2008). Representing textual requirements as graphical natural language for uml diagram generation. In *SEKE'08*, pages 478–483.

[Kof, 2010] Kof, L. (2010). Requirements analysis: concept extraction and translation of textual specifications to executable models. In *Proceedings of the 14th international conference on Applications of Natural Language to Information Systems*, NLDB'09, pages 79–90, Berlin, Heidelberg. Springer-Verlag.

[Lin and Pantel, 2001] Lin, D. and Pantel, P. (2001). Discovery of inference rules for question answering. *Natural Language Engineering*.

[Makki et al., 2008] Makki, J., Alquier, A.-M., and Prince, V. (2008). Ontology Population via NLP Techniques in Risk Management. In *ICSWE: Fifth International Conference on Semantic Web Engineering*.

[Nakamura-Delloye, 2011] Nakamura-Delloye, Y. (2011). Named entity extraction for ontology enrichment. In *IPSJ SIG Technical Report*, page 1, Japon.

[Njonko and El Abed, 2012] Njonko, P. and El Abed, W. (2012). From natural language business requirements to executable models via sbvr. In *Systems and Informatics (ICSAI), 2012 International Conference on*.

[Thongkrau and Lalitrojwong, 2012] Thongkrau, T. and Lalitrojwong, P. (2012). Ontopop: An ontology population system for the semantic web. *IEICE Transactions*, 95-D(4):921–931.