Context Aware Ontology based Information Extraction

Sapan Shah and Sreedhar Reddy

Tata Research Development and Design Center,
Tata Consultancy Services Limited,
Pune 411013
India
{sapan.hs, sreedhar.reddy}@tcs.com

Abstract

We have developed an ontology based information extraction system where property and relation name occurrences are used to identify domain entities using patterns written in terms of dependency relations. Our key intuition is that, with respect to a given ontology, properties and relations are much easier to identify than entities, as the former generally occur in a limited number of terminological variations. Once identified, properties and relations provide cues to identify related entities. To achieve this, we have developed a pattern language which uses the grammatical relations of dependency parsing as well as linguistic features over text fragments. Ontology constructs such as classes, properties and relations are integral to pattern specification and provide a means for extracting entities and property values. The pattern matcher uses the patterns to construct an object graph from a text document. The object graph comprises entity, property and relation nodes. We have developed a global context aware algorithm to determine the ontological types of these nodes. Type of one node can help determine the types of other related nodes. We use the concept of entropy to measure the uncertainty associated with the type of a node. The type information is then propagated through the graph from low entropy nodes to high entropy nodes in an iterative fashion. We show how the global propagation algorithm does better than a local algorithm in determining the types of nodes. The main contributions of this paper are: an ontology aware pattern language; a global context aware type identification algorithm.

The 18th International Conference on Management of Data (COMAD), 14th-16th Dec, 2012 at Pune, India.

1. Introduction

We live in a networked world where information is growing at an explosive rate. The ability to draw useful insights from this information is going to be a key competitive advantage for enterprises. New business models are emerging that require highly dynamic configurations of supply chains. Effective management of such supply chains requires constant monitoring and analysis of information on suppliers, consumers, competitors, their operating environments and so on. This calls for a highly flexible and dynamic information architecture that allows us to collect and integrate information not only from within the enterprise but also from outside the enterprise such as online sources, social media sites and so on. The ability to dynamically discover and integrate relevant information sources is a key feature of this architecture.

With this in mind, we have developed an information integration architecture (see fig. 1) where ontologies and ontology driven information extraction play a key role. We have an enterprise level ontology that provides a unified view of information at the enterprise level. This ontology is mapped to source level ontologies. A source level ontology provides a conceptual view of information available at the source.

Integration of a new source into the framework involves specifying the relevant ontology and building an adaptor. The adaptor is responsible for extracting information and presenting it as an instance of the source ontology. Integration of structured sources is relatively easier and we will not discuss that in this paper. Integration of unstructured sources is more complex. First we have to identify the relevant ontology fragment (using ontology discovery techniques) and then we have to build a suitable information extraction component. Building an

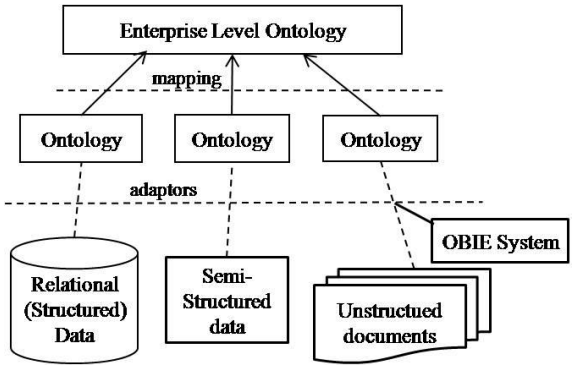

Figure 1: Enterprise Information Integration Framework

information extraction component using traditional IE techniques is a fairly involved job as they require extensive customizations (training, mark-up, tweaking rules, and so on). This is not a viable approach in a dynamic discovery and integration scenario. We need a more nimble approach. We discuss one such approach where information extraction can be driven entirely by the ontology, without any domain specific customizations. This obviously has its trade-offs. The approach places a higher premium on precision than on recall, as reliability of information is much more critical in a dynamic integration scenario where there is minimal expert intervention.

1.1. Ontology based Information Extraction

Information Extraction (IE) is the task of extracting structured information from unstructured or semi-structured sources. IE systems are supplied with the information of *what is to be extracted* in the form of output templates. Ontology based information extraction (OBIE) has recently emerged as a sub-field of IE where ontologies are used in the information extraction process. Output of the extraction process may also be represented in terms of an ontology. Ontology is defined as a formal and explicit specification of a shared conceptualization [11]. An ontology models a domain terminology in terms of concepts, properties and relations which can be used to specify information extraction targets. OBIE systems are broadly classified as ontology learning systems and ontology population systems. The task of an ontology learning OBIE system is to construct domain specific concepts and properties from unstructured text. Whereas, an ontology population OBIE system extracts instances of domain specific concepts and their property values for a given ontology. In this paper, our focus is on an ontology population system.

1.2. Our Approach

The key idea behind our approach is that it is much easier to identify property (and relation) name occurrences than entity name occurrences. The reason for this is that while an entity name may occur without an associated concept name reference, a property value rarely ever occurs without the associated property name reference. To illustrate, suppose we have an ontology fragment having one concept i.e. *Country* and two properties i.e. *Country.population* and *Country.capital*. Sentences such as the following are quite common:

`India has a population of 1.2 billion. Its capital is Delhi.`

While references to India frequently occur without the associated concept name reference (i.e. *Country*), it is difficult to imagine property values '1.2 billion' and 'Delhi' without the associated property name references (*population* and *capital*). Similarly it is difficult to imagine relation values without the associated relation name references. Also, while there can potentially be an infinite number of entity name occurrences, property (relation) names typically only occur in a limited number of terminological variations (e.g. population, populace). Thus in our approach we start by identifying occurrences of property and relation names and use them to identify entities. To achieve this, we have developed a pattern language which uses the grammatical relations (such as subject, verb, object, etc.) of dependency parsing to locate entities once the properties and relations are identified. The language also provides constructs to refer to ontology elements. These constructs serve two purposes: one, to specify constraints over ontology elements, and two to provide semantics for extracting information.

The pattern matcher uses the patterns written in the pattern language to construct an object graph from the input text document. The nodes of the object graph represent entities, properties and relations found in the document. The next step is to determine their ontological types for which we have developed a global context aware algorithm. We use the concept of entropy to measure the uncertainty associated with the type of a node. The type information is then propagated through the graph from low entropy nodes to high entropy nodes in an iterative fashion. The intuition behind this approach is that a node with a higher degree of certainty about its type can help determine the types of related nodes that have a lower degree of certainty about their types. For example, consider an ontology with classes such as *City*, *State* and *Country* and object property[1] *located_in* between: City and State; State and Country. Let's say a text document contains a sentence: `Gujarat is located in India`. Here the relation occurrence *located in* is not enough to decide the type of Gujarat which can potentially be *City* or *State*. Similarly, the type of India can be *State* or *Country*. Let's say the same document contains another sentence: `India is a country in South Asia`. This sentence provides the information that the type of India is *Country*. Now, if the information

[1] We will use the terms *relation* and *object property* interchangeably (similarly, *property* and *data type property*).

from *India* (a node with higher certainty about its type) is propagated to *Gujarat* (node with lower certainty), we can decide that the type of Gujarat is *State*.

The rest of the paper is organized as follows. Section 2 describes some of the systems developed for OBIE in the past. Section 3 discusses how domain ontology can be enriched to facilitate IE. Section 4 discusses details of input text pre-processing. Section 5 discusses pattern language constructs and their semantics. We present the global context aware type identification algorithm in section 6. Section 7 discusses experimental results. Section 8 ends with concluding remarks.

2. Related Work

Ontology based information extraction has recently emerged as a sub-field of IE. Research in this field has mostly concentrated on finding instances of domain specific concepts and learning taxonomic relations. Not much work has been done on finding non-taxonomic relations.

One of the first IE systems using ontology was the Embley's system [8] based on extraction ontologies, where ontologies are extended with regular expression based linguistic rules for ontological classes and properties. Other notable systems based on linguistic rules include FASTUS [1], PANKOW [4,5], OntoX [19], Ontosyphon [13], KIM [15]. FASTUS uses a cascade of finite state automata to extract the events and entities of interest. To extract instances of domain specific concepts, PANKOW, Ontosyphon and KnowItAll [9] systems use a set of generic Hearst patterns. These patterns are instantiated with ontological constructs for extraction purposes. For example, <Concept>s such as <Instance> is one of the Hearst patterns [12]. Here, Concept can be instantiated with country class to extract country instances. PANKOW [4,5] system first finds all proper nouns in a document and then conducts web based searches for every combination of proper noun and ontological class for a set of Hearst patterns. It then uses the number of hits recorded for each class to determine the correct class label for the proper nouns. Ontosyphon system uses a similar approach where instead of focusing documents, it uses web based searches to find possible instances of classes in the ontology. In addition to the linguistic rules, systems such as KIM [15] and iDocument [3] use gazetteer lists for some classes to facilitate IE.

As the constituency based parsers are typically closer to the syntactic structure than the semantics of the sentence, other parsing mechanisms such as dependency parsing, link grammar, etc. are used by different systems for relation extraction. Fundel et al. [10] have built RelEx system for BioInformatics domain. It uses dependency tree paths to extract interaction between genes and proteins. It uses gazetteer lists for extracting genes and protein names from natural language sentences. Similarly, Schutz and Buitelaar have developed RelExt system [17], where the goal is to extract relations between concepts for ontology learning. The authors motivate the use of verbs to express relation between classes that specify domain and range of some action or event. Similarly, Banko et al. [2] present open information extraction approach, where binary relationships between the entities can be obtained using verb-centric lexico-syntactic patterns.

IE systems perform linguistic processing (e.g. tokenizing, POS tagging, chunking, etc.) over the input text before the actual task of extraction. The generated linguistic features then can be used as part of extraction rules. Various NLP tools such as GATE[2], Stanford CoreNLP are used for this purpose. As we are building a new pattern language, it is worthwhile to compare it with JAPE[3] component of GATE. JAPE provides finite state transduction over document annotations based on regular expressions. It is used by Saggion et al. [16] and KIM [15] to write regular expression for entity extraction. It is possible to write ontology aware JAPE transducers where classes in the ontology can be referred as part of regular expressions. However, The JAPE regular expressions are written in terms of annotations over tokens while the pattern language we have developed can be used to write regular expression over trees in additions to tokens. Second notable difference is that, one has to write explicit JAVA code using GATE ontology APIs to store extracted information into the ontology. In our case, we have extended the pattern language itself with a set of constructs that specify how to store the extracted information into the ontology.

3. Ontology Enrichment for IE

To facilitate IE, ontologies in [8,19] are enriched with annotations. On similar lines, we have added following annotations to the domain ontology (A domain expert assigns values for these annotations).

- **Description**: The classes of ontology should be enriched with description annotations describing their meaning. This can be useful for assigning initial probability of an entity having a particular class type. For each class, the similarity between the words in the context of a given entity and the words in the class description is calculated. These similarity values are then normalized to get initial probability values.
- **Identification Weight**: For each ontological class, relative identification weights are assigned for its data and object properties. These weights indicate the relative importance of a property or relation in identifying the class. For example, consider an

[2] GATE – General Architecture for Text Engineering: java suite of tools to perform NLP tasks developed at University of Sheffield. (http://gate.ac.uk/)

[3] JAPE – Java Annotation Patterns Engine: regular expression language in GATE.

Organization domain with two classes i.e. *Employee*, *Department* and three properties i.e. *Employee.name*, *Department.name*, *Employee.reports_to*. Here, the occurrence of *reports_to* in text can provide cues that the type of the associated entity is *Employee*. The same is not true for *name*. Hence, *reports_to* is given more identification weight than *name*.

- **Synonyms**: While finding out property and relation mentions in the text, we also want to consider their synonyms. Hence, we provide an annotation to manually add synonyms for properties, relations and classes. The WordNet[4] synonyms can also be added as part of this annotation.
- **Value Patterns**: Stricter constraints on the values of data type property may be required in some cases. Hence, we enrich the ontology with value patterns annotation that specifies the regex patterns that the values of the data type property should match. For example, consider a Camera Review domain with a property *Camera.megapixel*. As observed in the Camera Review corpus, the regex for the value pattern can be: **\d+(\.\d+)?(mp|megapixel).**

The pattern matcher uses above mentioned annotation for identification as well as classification of entities and their property values.

4. Pre-Processing

IE from unstructured text generally employs a series of pre-processing steps where linguistic features of the input text are collected. This section describes these pre-processing steps and builds a data structure which will be used by the pattern matcher.

4.1. Linguistic features using Stanford CoreNLP

Stanford CoreNLP[5] is an integrated suite of natural language processing tools for English. The input text is first tokenized and passed to sentence splitter which converts the input text document into a sequence of sentences. The sentences are then POS tagged using Maximum Entropy based tagger. It uses Penn Tree bank tag set for POS tagging. The sentences are then parsed using lexicalized PCFG parser and the constituent parses are stored in a data structure. Stanford has also developed rules for converting phrase structure trees to dependency trees. A dependency tree provides a representation for grammatical relations between words in a sentence. It uses the concepts of dependency parsing [14] such as *relation*, *governor* and *dependent*. Pronominal co-reference resolution is also important for information extraction. We use Stanford's co-reference resolution system for this purpose.

4.2. Induced Tree data structure

As mentioned earlier, we use the grammatical relations of dependency parsing to locate entities once the property (relation) occurrences are found. We use Stanford dependencies for this purpose. Stanford dependencies (SD) [7] provide a representation of grammatical relations between words in a sentence. These relations are binary in nature and can be represented in the form of triplets: <name of relation, governor, dependent>. Few examples of the relations follow.

- **Nominal Subject (*nsubj*)**: It is a noun phrase (dependent) which is a syntactic subject of a clause (governor).
- **Direct Object (*dobj*)**: The direct object of a VP is the noun phrase (dependent) which is the (accusative) object of the verb (governor).

SD representation contains a total of 53 grammatical relations [6]. The words of a sentence along with their grammatical relations form a tree called dependency tree where nodes represent the words and edges represent the grammatical relations (as an example see figure 2a).

We need regular expressions to be matched over this tree structure as part of our pattern matching algorithm. Stanford provides Tree Regular Expression (TRegex): a utility for matching patterns in trees. The regular expressions in TRegex are written in terms of node labels and they do not consider edge labels. The regular expression patterns that we need to apply use edge labels in addition to node labels. To solve this problem, we created a tree data structure different but derived from

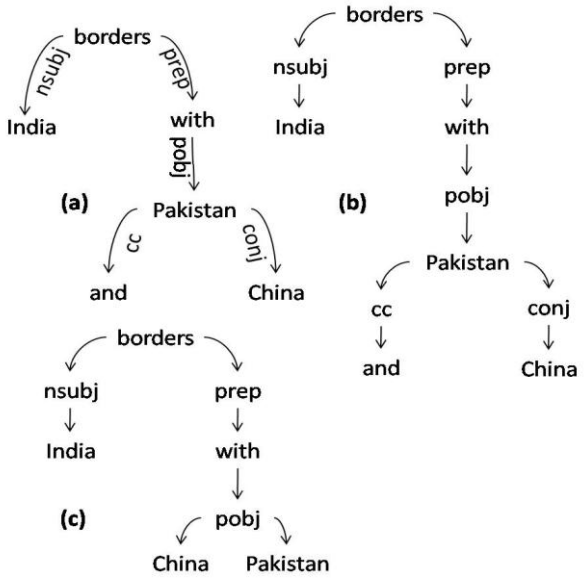

Figure 2: Induced Tree data Structure for an Example Sentence

[4] WordNet is a large lexical database of English developed at Princeton University. (http://wordnet.princeton.edu/)

[5] Stanford CoreNLP: a set of natural language analysis tools provided by Stanford University. (http://nlp.stanford.edu/software/corenlp.shtml)

Table 1: Tree Transformation Patterns

TreeTransformation	TRegex Pattern - condition	TSurgeon Operations	Remarks
ConjunctionAnd	/.*/=head < (cc=vCC < and=vAnd) < (conj=vConj < /.*/=brother)	move brother $- head; delete vConj	All the conjuncts in *and* conjunction becomes siblings; children of Parent of head conjunct. (India borders with *Pakistan* ***and*** *China*)
CompoundNoun	/.*/=head < (nn=vNN < /.*/=compound)	accumulate compound head compound; excise vNN compound	Words in a compound noun are considered as single unit e.g. India borders with Sri Lanka; *Sri Lanka* is stored as a single induced tree node.
ModifierList	/.*/=head < (/.*mod.*/=vMod < /.*/=modifier)	accumulate modifier head modifier; excise vMod modifier	All modifiers are stored along with an induced tree node of word that they modify.
CompoundNumber	/.*/=head < (number=vNumber < /.*/=compound)	prune vNumber	All the words in compound number are treated as a single node *e.g.* I lost $ 3.2 billion. Here, *$ 3.2 billion* is treated as a single node of number type.

dependency tree. This data structure contains nodes for words as well as grammatical relations as shown in figure 2b. The grammatical relation nodes are internal nodes: used only for patterns, not for extraction. We will refer to this data structure as *induced tree* in the rest of the paper. It should be noted that Stanford provides a utility for pattern matching over dependency trees called Semgrex. However, it does not provide any means of integrating ontology information.

Node description in a TRegex pattern is specified using literal or regular expression (specified between /). During pattern matching, it matches with node labels of the tree. Relations are specified between the node descriptions. All relations in a pattern are relative to the first node. Parenthesis can be used to group related nodes. For example, A < B < C mean A is the parent of B and C; A < (B < C) means A is a parent of B and B is a parent of C. Named nodes are used to bind a variable with the value matching the specified regex. For example, /NN.*/=Var is a named node and variable Var can be used to refer to the actual node label that matches with regex NN.*.

4.3. Tree Transformations

A set of tree transformations are applied to the induced tree before the actual pattern matching starts. Stanford provides TSurgeon - a tree transformation language. TSurgeon pattern consists of a single TRegex pattern P and a number of TSurgeon operations that are executed when P matches on the tree. These operations refer to the named nodes in the TRegex pattern for tree manipulations. Suppose we want to perform IE for GeoPolitical Entities domain having a *Country* class and *borders_with* relation. Figure 2b shows an induced tree for a sentence from this domain. A TRegex pattern to extract this relation is

$$/.*/=\text{Verb} < (\text{nsubj} < /.*/=\text{Source}) < (prep < (with < (pobj < /.*/=\text{Target}))) \quad (1)$$

Where, Verb, Source and Target are TRegex variables. When this pattern is applied to the induced tree, it returns a match where the variable bindings for Verb, Source and Target are *borders*, *India* and *Pakistan* respectively. As Verb matches with the relation name, we can extract an RDF triple viz. (India, borders_with, Pakistan). If we observe the example sentence closely, we missed extracting one more RDF triple viz. (India, borders_with, China). To solve this problem, the induced tree needs to be transformed such that China-Node becomes the child of pobj-Node. Stanford dependencies handle *and* conjunctions the following way: one of the conjuncts is selected as head (Pakistan here); the rest of the conjuncts become children of the head conjunct with conjunction (*conj*) relation. Let's denote the parent of the head conjunct as H (pobj here). First we need to apply a TRegex pattern to find *and* conjunction and then apply TSurgeon operations such that all the conjuncts become children of H. Figure 2c shows the induced tree after the application of this tree transformation (see table 1: ConjunctionAndTransformation). As we can see, the missed RDF triple can be extracted now, as it matches with the TRegex pattern in 1. Table 1 lists a set of tree transformations we have used.

5. Pattern Language

We have developed a pattern language for processing the induced tree and extracting information. Due to space constraints, we present only a subset of the grammar of this language (see text box 1 below).

A pattern consists of a premise and a sequence of

```
patterns:- pattern* <EOF>
pattern:- patternID "{" premise "}"
          "->" "{" actions "}"
patternID:- (DIGIT)+
premise:- (treePath ";")+
       (ontologyConstraint ";")+
              ("{" boolean_expression
               "}" ";")?
treePath:-element| element"--" treePath
ontologyConstraint:-
       ontologyElement = variable
actions:- ("{" action + "}")+
action :- LHS = RHS ";"
LHS:- ontologyActionElement | variable
RHS:-variable|identifier
       |action_function
```

1. Grammar for Pattern Language

actions. A premise is a set of conditions that should hold true for the actions to be executed. It consists of,

- **Tree paths**: A tree path specifies a sequence of elements. These elements are matched against node labels in the induced tree. An element can be a variable, identifier or a regular expression. A variable can be bound or unbound. While an unbound variable is bound with a value during pattern matching, a bound variable specifies a constraint: a matching tree node label must have the same value.
- **Ontology Constraints**: An ontology constraint is of the form '<lhs> = <rhs>'. It specifies that the value bound to a variable on the right hand side (rhs) must match with an ontology element on the left hand side (lhs). An ontology element can be a class, property, relation or an instance. Looking at the example sentence in figure 2, one would like to check whether the variable Verb gets a binding that matches with some ontology relation or its synonyms (which happens to be *borders_with* in the example). If so, we have a possible relation extraction with corresponding source and target entities. We can specify this constraint using

  $$relation = < Verb >$$

  This way, our pattern language provides language constructs to explicitly refer to various ontological elements.
- **Boolean Expression**: We support two boolean operators: And, Or. The basic operand in a boolean expression is a Boolean function. We support boolean functions over ontological constructs as well as linguistic features. For example, to check whether the type of the value bound to a variable matches with a pre-defined data type in the ontology, we have a function – isTypeMatching.

The actions component in the pattern specifies a sequence of actions to be performed over variable bindings from the premise. The basic constituent used in an action is assignment. An assignment is of the form '<lhs> = <rhs>'. The left hand side (lhs) of an assignment can either be a variable or an ontology element (ontologyActionElement in the grammar). We have a set of predefined keywords to refer to ontology elements with the following semantics,

- **relation (property)**: value of the right hand side (rhs) expression must be interpreted as an object (data) property in the ontology.
- **class**: value of the rhs expression must be interpreted as a class (concept) in the ontology.
- **source (target)**: value of the rhs expression must be interpreted as a source (target) entity of the property or relation occurring in the action.
- **entity**: value of the rhs expression must be interpreted as an entity (class instance) in the document.
- **previous_entity**: value of the rhs expression must be interpreted as an entity matched in the previous sentence in the document.

When lhs is a variable, it specifies that the values of both lhs and rhs expressions refer to the same underlying domain entity. This essentially says that lhs and rhs are to be treated as aliases of the same domain entity.

The rhs expression of an assignment can be,

- **Variable**: bound value of the variable is used in the action assignment.
- **Literal**: literal value specified as an identifier is used in the action assignment.
- **Action Function**: We may want to manipulate the bound value of a variable before it can be used for extraction. To do this, we provide action functions. The value returned by executing the action function is used as an action assignment. For example, if we have an instance of country and want to assign value for the official name of the country, we can use a function – concat(Republic, of, <Country>). During execution, if variable Country is bound to India, we can get the official name Republic of India using this function.

As mentioned, an action is specified by a group of assignments. For example, a relation extraction with source and target entities are specified by,

{source=<Entity1>;target=<Entity2>;relation=<Relation>}

Similarly there are actions to specify extraction of property with source entity and target value; extraction of class instance pair; extraction of an equivalent name (name aliases) for an entity (India and Republic of India).

5.1. Example Patterns

We will go through an example to see how one specifies patterns in this language. Consider GeoPolitical Entities domain with a *Country* Class and *coastline* property. Let's look at a sample sentence (Table2 – Pattern 1):

`India has a coastline of 7517 km.`

In the dependency tree of this sentence, `has` is the root verb; `India` is a subject and `coastline` is a direct object of `has`; `7517 km` is the prepositional object of preposition-`of` which modifies the direct object. So, paths that a pattern should look for in the induced tree are,

has -- dobj -- <Property> -- prep -- of – pobj -- <Value>;
has -- nsubj -- <Entity>

In addition, the direct object should match with some data type property in the ontology. An ontology constraint to specify this would be,

property=<Property>;

The premise built using paths and an ontology constraint above can match any data type property in the ontology hence it matches with `coastline`. The actions part for this pattern should perform property extraction and can be specified as,

{source=<Entity>;target=<Value>;property=<Property>}

Table 2: Generic Domain Independent patterns - Examples

India has a coastline of 7515 km.	**property extraction**

1 {
<HAS=has> -- dobj -- <Property> -- prep -- of -- pobj -- <Value>; property = <Property>;
<HAS> -- nsubj -- <Entity>; {isRoot(<HAS>) && isTypeMatching(<Value>, Number)};
} -> {
source=<Entity>; target=<Value>; property=<Property> }	

Ratan Tata launched Tata Nano in 2010.	**relation extraction**

2 {
<Verb> -- nsubj -- <Subject>; <Verb> -- dobj -- <Object>;
relation = <Verb>; {isRoot(<Verb>)};
} -> {
source = <Subject>; target = <Object>; relation = <Verb>; }	

India is a country in South Asia.	**Class Identification**

3 {
<Concept> -- nsubj -- <Instance>; <Concept> -- cop;
class = <Concept>;
} -> {
class = <Concept>; entity = <Instance>; }	

If we look closely at the dependencies exhibited in this example sentence, they are generic and can happen across sentences from different domains. As long as a sentence has a direct object matching with a data type property from a domain specific ontology, the entity and property value extraction is possible. In that sense the pattern described above is generic and can be used across different domains. We have compiled a set of such generic, domain independent patterns. There are a total of 18 patterns out of which we list only 3 patterns in Table 2 due to space constraints. First two patterns in the table are based on property extraction and relation extraction respectively. The last pattern shows one of the class identification patterns.

6. A Greedy Algorithm for Type Identification

We will first describe the ontology we have used for our experiments. We will be referring to it in the rest of the paper. We have downloaded FAO (Food and Agriculture Organization of the United Nations) Geopolitical ontology and modified it for our experiments. Figure 3 shows a section of this ontology.

As described in section 4 and 5, the text document is converted to a sequence of induced trees. The pattern matching algorithm then applies a set of patterns on these trees and generates a graph structure. We will refer to this graph structure as *object graph* in the rest of the paper. The object graph contains three types of nodes *viz.*

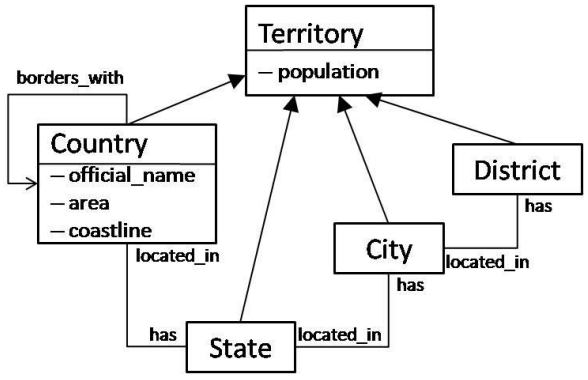

Figure 3: GeoPolitical Entities Ontology fragment

- **Entity Node**: represents an instance of a domain entity found in the document.
- **Property Node**: links an entity node with its property values.
- **Relation Node**: links two entity nodes that represent domain and range of some ontological object property.

These nodes just represent the entities, properties and relations identified in the document; their ontological types still have to be determined. The possible ontological types for the three types of nodes are: classes for entity nodes; data properties for property nodes; object properties for relation nodes. As we also account for co-references, the same entity node is used if the entity is referred in different parts of a document.

Table 3 gives a simple algorithm to determine the ontological types for the nodes in the object graph. We will refer to this algorithm as *LocalIE* in the rest of the paper. The first step in the algorithm applies a set of class-identification patterns to determine types for the entity nodes. We have used the Hearst patterns [10] for class-identification. The type of an entity which matches these patterns can directly be inferred; one does not have to rely on property or relation occurrence for its type identification. For example, consider a sentence: `India is a country in South Asia`. The type for the entity *India* can directly be determined using the pattern: *<Instance>* is a *<Concept>*. Pattern 3 in table 2 captures this pattern in terms of dependency relations. For the entity nodes which do not match these patterns and for the property and relation nodes, the algorithm assigns equal scores for their ontological types.

Table 3: An Algorithm for IE using Local Context

LocalIE – An Algorithm for IE using local context
1. Apply class-identification patterns (e.g. Table 2-Pattern 3) to get the type information for the entity nodes in object graph.
2. Use formula 2 to find the types of property nodes (Similarly find the types of relation nodes).
3. For each entity node (whose type is not determined in step 1):
a. Find the score for each ontology class using formula 3. As shown, this formula uses the local context (related property and relation nodes) along with their identification weights.
b. Assign class with the highest score as the correct type for the entity node (formula 4).
4. Convert the object graph to RDF triples.

$$type(A) = \underset{1 \le i \le |property|}{\text{argmax}} \{similarity(P_i.words, A.words)\} \quad \textbf{(2)}$$

where,
P_i= i[th] data property in the ontology;
P_i.words = words in the data property P_i (including its synonyms);
A.words = words occurring in the property node A.

$$score\left(C_i/_E\right) = \sum_{j=1}^{|property|} \left[score\left(P_j/_A\right) * C_i.iWeight(P_j)\right] + \sum_{k=1}^{|reltion|} \left[score\left(R_k/_B\right) * \sum_{L \in Range(R_k)} [C_i.iWeight(R_kL)]\right] \quad \textbf{(3)}$$

where,
E is an entity node in focus having a property node A and a relation node B.
$C.iWeight(P)$ = identification weight of property P for class C;
$C.iWeight(RL)$ = identification weight of relation R for class C; $L \in Range(R)$;
C_i= i[th] class in the ontology; P_j= j[th] property in the ontology;
R_k= k[th] relation in the ontology.
$score(T/N)$ = score for an ontological type T given node N.

$$type(E) = \underset{1 \le i \le |class|}{\text{argmax}} \left\{score\left(C_i/_E\right)\right\} \quad \textbf{(4)}$$

The types for the property and relation nodes are found by matching them with ontological data and object properties respectively (step 2). Here, the edit-distance based similarity scores are calculated between the words of a property (relation) node and an ontology data (object) property. The synonyms of a data (object) property are also taken into account. The data (object) property with the highest similarity score is then chosen as the correct type for the property (relation) node (formula 2). To determine the type of an entity node, the scores found for the neighboring property and relation nodes as well as their identification weights are used (formula 3). This algorithm uses only the local context to find the correct type of an entity node.

More informed decision for the type of an entity node can be made if the global context is also taken into account. Let us first motivate the need of such a global context aware algorithm. Consider the ontology in figure 3. It contains an object property *located_in* between *State* and *Country*; *City* and *State*; *District* and *City*. Whenever this relation occurs in the text document, there is an ambiguity about the types of the source and target entity nodes as the same name is used to refer to three different object properties in the ontology. Consider a text fragment from this domain,

Surat is located in Gujarat. It is recognized for its textile and diamond businesses. Vadodara is also located in Gujarat. It is the third most populated city with a population of almost 1.6 million.

The underlined phrases in this fragment are the instances of domain entities and their properties (relations). As we can see in the first sentence, the relation *located in* cannot provide correct type information for the related entities i.e. *Surat* and *Gujarat*, as they may refer to any of the four classes *viz.* District, City, State or Country. However from the last sentence, we can easily infer that the type of the entity *Vadodara* is City. If we use the type information of *Vadodara* along with the *located in* relation in the third sentence, we can infer that the type of *Gujarat* is State. Now, if we use the type information of *Gujarat* in sentence 1, we can infer that the type of *Surat* is City. The local algorithm we described in table 3 neither takes global context into account nor performs this kind of information propagation.

6.1. Entropy - Information Theory

We use the concept of entropy from information theory [18] to quantify the uncertainty associated with the type of a node. Entropy is a measure of uncertainty associated with a random variable and defined in terms of its probability distribution. Let's denote X as a discrete random variable having a set of possible values $\{x_1, x_2, \dots, x_n\}$ and a probability mass function $p(X)$ (such that $\forall i: p(x_i) \in [0,1]$; $\sum_{i=1}^{n} p(x_i) = 1$;). The entropy of X is then defined as,

$$H(X) = -\sum_{i=1}^{n} p(x_i) * \log_b p(x_i) \quad \textbf{(5)}$$

For example, consider two experiments: tossing a fair coin ($p(head) = 0.5$ and $p(tail) = 0.5$); tossing a two-headed coin ($p(head) = 1$ and $p(tail) = 0$). The outcome of the former experiment is most uncertain and thus has highest entropy, while the later has a definite

Table 4: An Entropy based Greedy Algorithm for IE

GlobalIE – An Entropy based Greedy Algorithm for IE

1. Execute step 1 to step 3a of the LocalIE algorithm to determine the types of property and relation nodes, and to get initial scores for entity nodes.

2. Normalize the class-score for each entity node E such that,

$\forall i: 1 \leq i \leq |class|; score(C_i/E) \in [0,1]; \sum_{i=1}^{|class|} score(C_i/E) = 1$

Calculate entropy values of all entity nodes.

3. Create a min-priority queue Q; add all entity nodes in Q.
visited_nodes = ϕ;

4. While(Q != empty) {
E = remove a node from Q with the least entropy value;
Assign correct type for node E using formula 4.
Add E to visited_nodes;
propagate_score(visited_nodes, E);
}

5. Convert the object graph to RDF triples.

propagate_score(visited_nodes, entity_node E) {
For(each relation B where E is the source entity) {
X = target entity for relation B;
propagetIfLow(E, X);
}
For(each relation B where E is the target entity) {
Y = source entity for relation B;
propagateIfLow(E, Y);
}
}

propagateIfLow(entity_node E, entity_node A) {
If ($entropy(A) > entropy(E)$ AND $A \notin visited\_nodes$)) {
For each class $C_i; 1 \leq i \leq |class|$,
Update $score(C_i/A)$ using formula 6.
Normalize the class-score for node A;
Re-calculate the entropy of node A ;
propagate_score(visited_nodes, A);
}
}

$$score\left(C_i/_D\right) += \sum_{j=1}^{|relation|} \left[score\left(R_j/_B\right) * \left(\sum_{L \epsilon Range(R_j)} C_i.iWeight\left(R_j L\right) * score\left(L/_E\right) \right) \right] \quad \textbf{(6)}$$

where, D and E are source and target of relation node B

outcome and the entropy is 0. The entropy of a random variable is proportional to the uncertainty of the outcome.

In our context, we use the concept of entropy to measure the uncertainty associated with the type of a node. For example, for an entity node the possible types are the classes in the ontology. Let $C_i; 1 \leq i \leq |class|$ denote the classes. If we do not have any information about the type of an entity node E (highest uncertainty and entropy), we assign uniform score for the classes i.e. $score(C_i/E) = 1/|class|$. In our algorithm, we use the local formula in 2 to assign initial scores for the class types of the entity nodes.

6.2. An Entropy based Greedy Algorithm

To find the correct type of an entity node, the LocalIE algorithm just uses the neighbouring property and relation nodes. If the information about the correct type of some entity node in the object graph is available, it should be used for classification of other related entity nodes in the graph. Table 4 describes a global context aware algorithm which uses related entity nodes in addition to the property and relation nodes for classification. We will refer to this algorithm as *GlobalIE* in the rest of the paper.

GlobalIE uses edit-distance based similarity score for classifying property and relation nodes (same as LocalIE). The main difference is the use of related entity nodes to classify current entity in focus. The entity nodes in the object graph are ordered according to their entropy values. The rationale behind this ordering is: the nodes with high information about their correct type can help determine the types of other related nodes having low information about their types.

In GlobalIE, once the types for the property and relation nodes are determined, the entity nodes are added to a min-priority queue (step 3). The nodes in this queue are ordered in the increasing order of their entropy values. To calculate the entropy value correctly, the scores for the class types of an entity node E must satisfy two conditions:$\forall\, i: 1 \leq i \leq |class|;\ score(C_i/E) \in [0,1]$ and $\sum_{i=1}^{|class|} score(C_i/E) = 1$. To achieve the same, we normalize these scores in the following way $\forall i: 1 \leq i \leq |class|;\ score(C_i) = score(C_i)/\sum_{k=1}^{|class|} score(C_k)$.
During each pass of the while loop in step 4, an entity node with the least entropy value is removed from the queue and assigned its correct type using formula 4. The information contained in this node is then propagated to other nodes through the graph structure. In particular, the type information is propagated through the graph from low entropy nodes to high entropy nodes (see function: propogate_score). As we do not want to update the score of a node which is already assigned its type, we maintain a list of visited nodes (visited_nodes list in step 3). The time complexity of GlobalIE is in the order of the size of the object graph. Let's now go through an example to demonstrate how the information is propagated between the nodes and how the entropy based ordering is beneficial for entity classification.

6.3. An Example demonstrating Global IE

Consider again the GeoPolitical entities domain (fig. 3) and the example text fragment mentioned earlier in this section. We used a set of generic patterns as described in section 5 for information extraction and applied the pattern matcher over this fragment. Figure 4 shows the

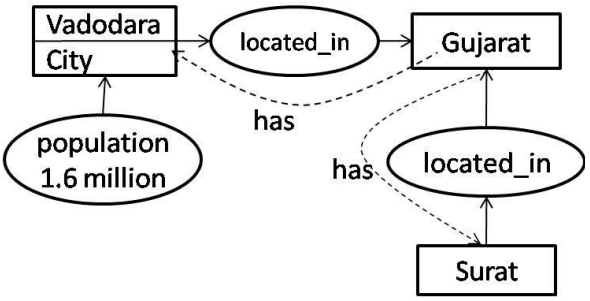

Figure 4: Object graph for text fragment

generated object graph. Let's now go through the execution of GlobalIE. Table 5 shows the scores of class types of the entity nodes and their entropy values at various points in time during the execution of the algorithm. Initially, the scores are equal for all entity nodes (except *Vadodara*, as it is directly assigned its correct type by the class-identification pattern) as shown in row 1. The scores of the property and relation nodes (along with their identification weights) are then used to update the scores of the entity nodes (step 1). Row 2 shows these scores after normalization (step 2). During the first pass of the while loop in step 4, *Vadodara* is selected and removed from the priority queue, as it has the least entropy value. The scores of the class types of *Vadodara* are then propagated through the graph structure. The object graph has a relation node *located_in* for which *Vadodara* is a source entity and *Gujarat* is a target entity. Hence, the scores for the class types of *Gujarat* are updated using the scores of *Vadodara* (row 4). In the second pass, *Gujarat* is selected and removed from the priority queue as it has the least entropy value now. The class type of this node is then determined using formula 4. Now, this node is connected to two entity nodes in the object graph i.e. *Vadodara* and *Surat*. As the entity node *Vadodara* is already visited earlier, it is ignored and the scores for the class types of *Surat* are updated using the scores of *Gujarat* (row 5). In the third pass, we are left with only one entity node i.e. *Surat*. Hence, it is selected and removed from the priority queue (row 6) and its class type is determined using formula 4. The priority queue is empty now and the algorithm terminates. The class types assigned by this algorithm for the entity nodes are *City*, *State* and *City* for *Vadodara*, *Gujarat* and *Surat* respectively. As we can see, the algorithm finds the correct values for the class types of the entity nodes. When we executed LocalIE algorithm on the same text fragment, it incorrectly assigned class types *District* and *Country* for the entity nodes *Surat* and *Gujarat* respectively (The class type having highest score in table 5 - row 2 is selected as the correct type of the entity node in LocalIE).

7. Experiments

7.1. Digital Camera Reviews domain

Yildiz et al. [19] have developed an ontology driven IEs – OntoX. It focuses mainly on identifying property mentions and their values. The ontology contains one class i.e. *camera* having five data properties. It is enhanced with a set of keywords for each data type property. The system uses regular expressions to find the instances of pre-defined XML data types in the text document and looks for keywords in their vicinity. The property whose keyword is closest to the data type instance and having the same XML data type is selected. For example, consider a sentence: Powershot A95 is a 5.0 megapixel camera. Here, 5.0 is XSD:float and megapixel is a property having keyword megapixel and data type XSD:float. Hence, 5.0 is a value of megapixel property. The dataset consists of 138 digital camera reviews. The focus of this experiment is to show how the patterns based on grammatical relations are useful for relating entities with their property values.

It should be noted here that the task performed by OntoX system is to just find property values. In our case, we also find entities and associate them with their property values. We have used the set of generic patterns described in section 5.1 for IE over camera reviews dataset. Table 6 shows the precision and recall values for some of the camera properties. The precision of our system is better than the OntoX system while the recall values are very low. The reason is in our approach we only identify those properties for which entities are identified. Thus, we miss some of the properties. Whereas OntoX focuses only on property values, so its recall is higher. It is interesting to note that we get very high precision values which suggest that our approach is conservative. The system may not be able to extract all the entities and property values but whatever is extracted

Table 5: The scores of class types of the entity nodes in the example text fragment. The first column specifies the algorithm step; the rest of the columns specify the scores of class types of the entity nodes using the format: (Territory, State, District, Country, City)

Step	Gujarat	Vadodara	Surat
Init.	(0.2, 0.2, 0.2, 0.2, 0.2) – 1.61	(0, 0, 0, 0, 1) - 0	(0.2, 0.2, 0.2, 0.2, 0.2) – 1.61
2	(0.05, 0.25, 0.05, 0.41, 0.25) – 1.35	(0, 0, 0, 0, 1) - 0	(0.08, 0.24, 0.37, 0.08, 0.24) – 1.44
While loop of step 4			
pass 1	(0.03, 0.48, 0.03, 0.28, 0.17) – 1. 24	**(0,0,0,0,1) - 0**	(0.08, 0.24, 0.37, 0.08, 0.24) – 1.44
pass 2	**(0.03, 0.48, 0.03, 0.28, 0.17) – 1. 24**	(0,0,0,0,1) - 0	(0.05, 0.17, 0.26, 0.05, 0.47) – 1.31
pass 3	(0.03, 0.48, 0.03, 0.28, 0.17) – 1. 24	(0,0,0,0,1) - 0	**(0.05, 0.17, 0.26, 0.05, 0.47) – 1.31**

Table 6: Comparision of Our System with OntoX on Camera Review domain

Property	Our System		OntoX	
	Prec.	Rec.	Prec.	Rec.
Megapixel	0.93	0.39	0.52	0.51
Display Size	0.88	0.2	0.80	0.82
Model Name	0.76	0.64	0.79	0.79

Table 7: Results on GeoPolitical Entities Domain

Concept/Property /Relation	Precision	Recall
Country	0.85	0.69
borders_with	0.72	0.39
located_in	0.86	0.78
official_name	1.0	0.74
population	0.92	0.57
coastline	0.57	0.80
area	1.0	0.60
Total	**0.82**	**0.54**

is extracted with high accuracy. If we look at the recall values closely, the recall for the property model_name is high. It then decreases for megapixel and very low for display_size. If we observe any file from the corpus, the model_name property is same as the name of an extracted entity. The megapixel property occurs very near to the entity occurrence (mostly in the same sentence). The display_size property is mentioned very far from the entity (mostly in the next paragraph), thus decreasing the probability of associating the property with the entity. The induced tree paths used in our patterns do not consider word relations across sentences. We rely on co-reference resolution when the entity and property mentions are in different sentences. We have also provided a language construct called previous_entity using which a pattern can refer to the entities found in earlier sentences. Despite this, it is not easy to relate an entity with its property if they are widely separated in the text.

7.2. GeoPolitical Entities domain

We have downloaded 36 Wikipedia pages of country profile, converted them to text and manually tagged them for correct entity and property values. As part of this experiment, we have considered the data and object properties of only the *country* class (see fig. 3). We used the generic patterns described in section 5.1 for IE. Our experiments helped us identify these patterns and during the course of the experiments our initial set went through several additions and modifications. We randomly selected 10% of corpora (4 pages) to analyze whether the generic patterns we have are good enough for extraction, especially we looked at the entity, property and relation occurrences and how they are related by the dependency relations. At the end of this exercise, we had to add 3 new patterns and modify 4 existing patterns. In total we used 14 patterns and performed the experiments. Table 7 lists the precision and recall values for classes, properties and relations. The overall precision is 0.82 and recall is 0.54 which again strengthens our argument that the system is conservative and makes fewer mistakes (high precision). The reason for higher precision is that unlike in traditional approaches where identification is primarily text pattern based (which can throw up spurious matches), we also consider an entity's property and relationship context which reduces spurious matches. However, this can have an adverse impact on recall as some of the valid matches might also be turned down on account of not having matching property and relation contexts. As explained earlier, this behaviour of higher precision and lower recall is fine, as reliability is a key concern in our enterprise information integration framework.

We would like to point out here that the extra patterns that we had to add were due to the peculiar ways in which some properties were written in the text corpora. The generic patterns we have collected will work best when the sentences in the text document are property formed and follow the English grammar, such as in published articles. The text documents in different genres may have different styles of writing English sentences (publications vs. blog posts) and it's important to capture them in the form of dependency relations. For this reason, we may have to analyze different genres of text documents and augment the list of generic patterns.

7.3. Analysis of our OBIE system

The key constituents of our system are: a pattern language and a global type identification algorithm. A relevant question in this context is what varieties of patterns can be expressed in our pattern language. The constituents of the language (dependency relations, boolean functions, ontology constraints) provide the necessary power to write various kinds of patterns mentioned in the IE literature. A lot of systems in the literature have used Hearst pattern [12] and lexico-syntactic patterns [2] for extraction. We could successfully convert these patterns into equivalent patterns in our pattern language.

Once the object graph is generated by the pattern matcher, the type of the object graph nodes has to be identified. The accuracy of type identification can improve if we go beyond the local context and make use of all the relevant information available in the document. That's what our global propagation algorithm aims to achieve. The direction of propagation is determined by entropy ordering where information flows from nodes of high certainty to nodes of low certainty. In many cases mere presence of properties and relations is sufficient to uniquely identify an entity's type. This is possible when the names of these properties and relations are unique in the ontology. However duplicate names are quite common in real-life ontologies. For example, the *located_in* object property given in section 6 relates three different class pairs. Similarly, *reports_to* structure in an organization ontology; *part_of* structure in a product ontology, and so

on. A global propagation algorithm can make a big difference in such cases.

8. Conclusion and Future Work

We presented an information extraction approach where we first identify property and relation name occurrences in the text and then use patterns written in terms of dependency relations to identify related entities. To achieve the same, we have developed an ontology aware pattern matcher which uses these patterns to generate an object graph from a text document. We have also developed a global context aware algorithm to identify the ontological types of the object graph nodes. The algorithm is greedy and it uses the entropy ordering to decide information propagation between the nodes where type information is passed from low entropy nodes to high entropy nodes. The main contributions of this paper are: an ontology aware pattern language; a global context aware type identification algorithm.

We have experimented with GeoPolitical entities domain with a small set of text documents from Wikipedia. The result looks promising. An immediate (also important) task at hand is to test our approach on larger and varied set of corpora to check its applicability in general. We also want to integrate our system into the larger enterprise information integration framework to check its utility.

References

[1] Douglas E. Appelt, Jerry R. Hobbs, John Bear, David J. Israel, and Mabry Tyson, "FASTUS: A Finite-state Processor for Information Extraction from Real-world Text," in *IJCAI*, Chambéry, France, 1993, pp. 1172-1178.

[2] Michele Banko, Oren Etzioni, Stephen Soderland, and Daniel Weld, "Open information extraction from the web," *Communication of ACM*, vol. 51, no. 12, pp. 68-74, 2008.

[3] Adrian Benjamin, Hees Jorn, van Elst Ludger, and Dengel Andreas, "iDocument: Using Ontologies for Extracting and Annotating Information from Unstructured text," in *KI*, 2009, pp. 249-256.

[4] Philipp Cimiano, Siegfried Handschuh, and Steffen Staab, "Towards the self-annotating web," in *Proceedings of the 13th international conference on World Wide Web*, NY, USA, 2004, pp. 462-471.

[5] Philipp Cimiano, Gunter Ladwig, and Steffen Staab, "Gimme' the context: context-driven automatic semantic annotation with C-PANKOW," in *Proceedings of the 14th international conference on World Wide Web*, Chiba, Japan, 2005, pp. 332-341.

[6] Marie-Catherine de Marneffe and Christopher D. Manning, "Stanford typed dependencies manual," Stanford University, 2008.

[7] Marie-Catherine de Marneffe and Christopher D. Manning, "The Stanford typed dependencies representation," in *22nd International Conference on Computational Linguistics*, Manchester, United Kingdom, 2008, pp. 1-8.

[8] David W. Embley, "Towards Semantic Understanding -- An Approach Based on Information Extraction Ontologies," in *Proceedings of the Fifteenth Australasian Database Conference*, Dunedin, New Zealand, 2004, pp. 18-22.

[9] Oren Etzioni et al., "Web-scale information extraction in knowitall: (preliminary results)," in *Proceedings of the 13th international conference on World Wide Web*, New York, NY, USA, 2004, pp. 100-110.

[10] Katrin Fundel, Robert Kuffner, and Ralf Zimmer, "RelEx - Relation extraction using dependency parse trees," *Bioinformatics*, vol. 23, pp. 365-371, 2007.

[11] Thomas R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199-220, July 1993.

[12] Marti A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *14th Internation Conference on Computational Linguistics*, Nantes, France, 1992, pp. 539-545.

[13] Luke K. McDowell and Michael Cafarella, "Ontology-driven information extraction with ontosyphon," in *ISWC*, Athens, GA, 2006, pp. 428-444.

[14] Joakim Nivre, "Dependency Grammar and Dependency Parsing," Vaxjo University: School of Mathematics and Systems Engineering, 2005.

[15] Borislav Popov, Atanas Kiryakov, Damyan Ognyanoff, Dimitar Manov, and Angel Kirilov, "KIM – a semantic platform for information extraction and retrieval," *Natural Language Engineering*, vol. 10, no. 3, pp. 375-92, 2004.

[16] Horacio Saggion, Adam Funk, Diana Maynard, and Kalina Bontcheva, "Ontology-Based Information Extraction for Business Intelligence," in *ISWC*, 2007, pp. 843-856.

[17] Alexander Schutz and Paul Buitelaar, "RelExt: A Tool for Relation Extraction from Text in Ontology Extension," in *ISWC 2005*, 2005.

[18] E. Claude Shannon, "A mathematical theory of communication," *Bell System technical journal*, vol. 27, pp. 379-423, 1948.

[19] Burcu Yildiz and Silvia Miksch, "ontoX - a method for ontology-driven information extraction," in *ICCSA'07*, vol. 3, Kuala Lumpur, Malaysia, 2007, pp. 660-673.