Open Access

Open Journal of Semantic Web (OJSW)
Volume 7, Issue 1, 2020

http://www.ronpub.com/ojsw
ISSN 2199-336X

Ten Ways of Leveraging Ontologies for Rapid Natural Language Processing Customization for Multiple Use Cases in Disjoint Domains

Tatiana Erekhinskaya[A], Marta Tatu[A], Mithun Balakrishna[A], Sujal Patel[A], Dmitry Strebkov[B], Dan Moldovan[A]

[A] Lymba Corporation, 901 Waterfall Way, Bldg 5, Richardson, Texas, USA,
[B] Independent Researcher, 901 Waterfall Way, Bldg 5, Richardson, Texas, USA,
{tatiana, marta, mithun, sspatel, dstrebkov, moldovan}@lymba.com

Abstract

With the ever-growing adoption of AI technologies by large enterprises, purely data-driven approaches have dominated the field in the recent years. For a single use case, a development process looks simple: agreeing on an annotation schema, labeling the data, and training the models. As the number of use cases and their complexity increases, the development teams face issues with collective governance of the models, scalability and reusablity of data and models. These issues are widely addressed on the engineering side, but not so much on the knowledge side. Ontologies have been a well-researched approach for capturing knowledge and can be used to augment a data-driven methodology. In this paper, we discuss 10 ways of leveraging ontologies for Natural Language Processing (NLP) and its applications. We use ontologies for rapid customization of a NLP pipeline, ontology-related standards to power a rule engine and provide standard output format. We also discuss various use cases for medical, enterprise, financial, legal, and security domains, centered around three NLP-based applications: semantic search, question answering and natural language querying.

Type of Paper and Keywords

Short communication: *ontologies, natural language processing, domain-specific knowledge, labeling, semantic graph, natural language querying*

1 Introduction

Although ontologies have been used for knowledge representation and inference for many decades [28], they have not gained significant adoption within the data science/NLP community, with a few notable exceptions: [42], [18], [38], [31]. This paper attempts to bridge the gap between ontology-driven and data-driven approaches and discusses practical scenarios for a beneficial combination of the two.

Based on our experience with providing enterprise-level NLP solutions for multiple domains, we created a project flow and a resulting NLP pipeline that are driven by ontologies, yet utilize the most recent advances of Deep Learning. At a high-level, an input ontology is used on three major dimensions: (1) to customize the NLP pipeline for a new domain, (2) to use Resource Description Framework (RDF) within the NLP pipeline, and (3) for end-user applications.

As shown in Figure 1, we use ontologies for NLP in 10 different ways. First of all, we use ontologies to describe the labeling schema (e.g., named entity types,

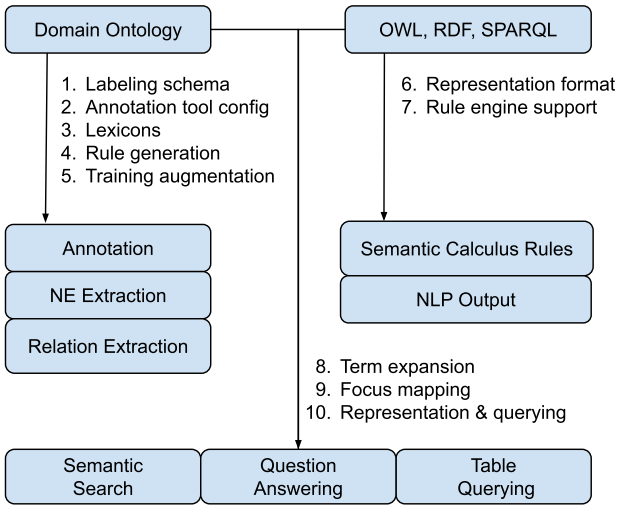

Figure 1: Integration of Ontology, Annotation tool and NLP

relation types, and section tags) that is to be used to annotate the data. Despite the various features provided by annotation tools to improve user experience, the key for successful data labeling is agreement between subject matter experts (SMEs) on the definitions of the labels. Currently, this agreement either (1) happens outside of the annotation tool (as part of the task documentation) or (2) is saved as notes in the annotation tool's custom format. However, ontologies provide a way to capture domain knowledge in a format that is both machine- and human-readable [41].

Furthermore, when reusing labeled data and combining corpora developed by different teams, same tags might have different meanings. This semantic discrepancy corrupts the training process and is hard to trace back. When labeled corpora are accompanied by governing ontologies, SMEs are able to reconcile edge cases and formalize new category definitions.

In addition to keeping SMEs on the same page, we use an input ontology to configure the annotation tool and eliminate invalid labeling. Furthermore, the ontology enables the validation of manually and automatically generated annotations against different versions of the ontology (created as part of the development process).

If ontologies contain instance data, we leverage it in the form of lexicons to extract custom named entities (NEs). Since collecting extensive class hierarchy and/or additional instance data is a labor-intensive process, we use unsupervised NLP to come up with good candidates from unlabeled text. These candidates are then reviewed by SMEs who select the concepts of interest and finalize the IS-A relations as well as create custom relation types.

To support relation extraction and other types of automatic tagging using an intermediate semantic representation of the document, we combine the RDF and SPARQL frameworks within a rule-based engine. These semantic rules can be written by hand, learned from a labeled corpus as well as automatically generated using a relation's labels and domain/range restrictions as defined within an ontology.

Furthermore, we blend configurations required by AI and NLP tools into input ontologies. This flattens the learning curve and provides control to SMEs without concerning them with implementation details, and, therefore, shortening the feedback loop.

Finally, RDF format is used to represent the output of NLP and AI tools, which can then be validated against the input ontology, shared between systems, and used for further inference.

It is important to note that ontology usage does not put any restrictions on implementation details of annotation tools, NLP modules, or applications. Rather, ontologies provide an expressive framework for schema, configuration, and additional features. This information can be leveraged by statistical, rule-based, machine learning (ML) or deep learning (DL) tools.

This article is an extension of the work presented in [16]. We have included more details for all ways of making use of ontologies during NLP, a summary of our motivational use cases, and quantitative characteristics of varying-domain ontologies and the automatically-generated artifacts.

2 Related Work

There is also a substantial body of literature on data modeling in general: entity-relationship modeling [13], entity-attribute-value [7], fact-based modeling [15], COMN [19], as well as more NLP-oriented slot-based approaches that describe verb arguments, such as FrameNet [4]. Additionally, there are less formalized domain-specific frameworks, e.g., the PICO model (Patient/population/problem, intervention, comparison, outcome) [32] in health care, which can be transferred to other domains, such as products and their technical issues, sports teams, companies, etc. Ontologies provide additional inference capabilities as well as a broad range of commercial and open-source tools supporting the development process and usage in application.

Domain ontology creation was researched for several decades, see for example [28] and [39], suggesting to start with enumerating important domain terms, define ontological classes and their hierarchy, and then defining their corresponding properties. All of the frameworks mentioned above, together with existing universal ontologies, for instance, BFO [2], or domain-

specific ontologies, e.g., FIBO [9], can be leveraged for a new application.

NLP is not the only area to show the benefits of ontology in the modern enterprise domain. Recently, ontologies were used to represent ML schema [30], as well as in information technology (IT) cases [35].

There is a body of literature on using Semantic Web resources (languages/ontologies/knowledge-bases/tools) to improve Information Extraction, and vice versa - using Information Extraction to populate the Semantic Web [22]. Ontologies proved to be useful for named entity/concept/term extraction when combined with deep learning [1], [6], [23]. Non-surprising, these results are mostly in the medical domain that benefits from established ontologies, such as SNOMED CT [33] and UMLS Metathesaurus [11]. While creating an extensive ontology from scratch implies an overwhelming effort, there are ways to leverage unsupervised NLP mechanisms to populate the ontology with recent deep learning techniques [29], [3].

Providing the output in an open standard format can significantly ease the chaining and integration of the various tools implemented in different programming languages. The closest analogies are annotations in UIMA [17], Apache Stanbol [8] or LODeXporter [40]; however, they are focused on XML/Java implementations, while an RDF representation is implementation-agnostic and enables the usage of various tools for visualization and querying. Currently, popular NLP libraries do not support or make use of ontologies. NLTK[1], spaCy[2], Apache OpenNlp[3], AllenNlp[4], Spark NLP[5], and others represent their labeling results using data structures specific to their corresponding programming language, such as string and/or arrays. However, there is at least one attempt to combine NLP with ontologies for the whole NLP pipeline - Teanga [42] that uses linked data format for NLP service interoperability. The system uses the NLP Interchange Format (NIF) [18] to represent output annotations. It also uses ontologies for configuration - a Teanga ontology - that describes properties of the NLP services.

Despite a plethora of annotation tools, only a few support relation labeling with a RDF representation/ontology, whereas most represent annotations in a tool-specific format, thus requiring tool-specific parsers and converters for interoperability. This need for RDF-based interoperability is proven by ongoing attempts to create converters from a task-oriented format, e.g., CoNLL, to RDF [14]. INCEpTION [21] is a notable annotation tool that supports RDF-based knowledge bases for knowledge management as well as external extensions.

One of the interesting directions of deep learning and ontology convergence is doing symbolic reasoning with deep learning networks to improve scalability and tolerance of uncertain data [20].

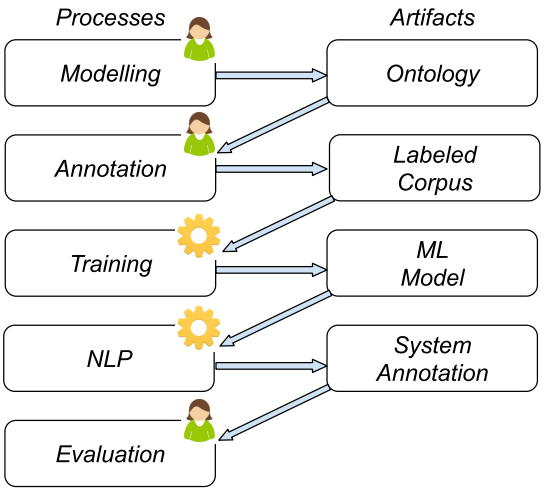

Figure 2: NLP solution development cycle

3 Motivational Use Cases

Our approach was iteratively developed based on a number of projects across multiple domains. Most representative use cases are described in Table 1.

Across all domains, the various business goals boil down to the core functionality of populating a knowledge graph based on the text and supporting the user's interaction with the knowledge graph. The key is the rapid customization of a NLP pipeline for a new knowledge goal.

4 Typical Project Flow

The overall development cycle of a project is depicted in Figure 2. We start with the domain knowledge modeling and create an ontology that will impact all major steps of the development process. In a nutshell, this input ontology defines the schema of the semantic graph built from a given text using NLP. We use the ontology to drive the annotation process. We then train the models used by a pipeline of NLP tools, using special controls that are embedded in the ontology. Furthermore, RDF and SPARQL are directly used as part of the NLP pipeline to power rule-based engines. During

[1] https://www.nltk.org
[2] https://spacy.io
[3] https://opennlp.apache.org
[4] https://allennlp.org
[5] https://www.johnsnowlabs.com/spark-nlp

Table 1: Enterprise use case examples

#	Customer	Business Goal	NLP Task
1	Large medical organization	Understand treatment best practices	Extract demographic characteristics of population, symptoms, and diagnosis; intervention options, and their outcomes
2	Large research organization	Collect information about antibiotic-resistant bacteria	Pinpoint mentions of bacteria, genes, antibiotics, etc., and their relations. Infer resistance based on facts extracted from multiple documents.
3	Aerospace company	Speed up repair process, optimize technician access to knowledge	Identify numeric values of parameters like temperature, torque, pressure, part codes, and their compatibility from documentation, including tabular data. Support English querying.
4	International car manufacturer	Understand car problems as recorded by mechanics and customer support	Label VIN, car model, make, and year, car parts and the issues. Support English querying and visual analytics
5	Online retailer	Wrangling product inventory. Improve online search	Single out key concepts and their attributes/functions in product descriptions
6	Large financial organization	Inferring product taxation category according to government regulations	Find product attributes and functions from product descriptions and legal text. Convert semantic graph to logic expression, perform ontology-based inference.
7	Large financial organization	Provide access to human resources information stored in a graph	Extract key concepts and their relations from English questions. Convert to formal graph queries and execute against the store.

the evaluation phase, test documents are converted to semantic graphs using the ontology-driven NLP pipeline. The resulting feedback leads to updates of the labeled data and its corresponding annotations, the semantic rules, the learned models, and even updates of the ontology itself.

Once the ontology is finalized, its corresponding NLP pipeline can be used to process a document collection of interest. Its resulting graph is then available to power various applications, to provide user interaction with the extracted knowledge, including, but not limited to monitoring for specific events, visualization, business analytics, search, and querying.

The following sections describe in detail the 10 ways for leveraging ontologies within an NLP engine.

5 Domain Knowledge Modeling

For the given NLP-based task, we start by modeling the domain knowledge, defining the domain's key concepts and the relations between them. Are we talking about patients, symptoms, and medications or about companies, acquisitions, and financial metrics? These high-level concepts and relations become classes and relations in our ontology, typically anywhere between 30 and 100 types. An example of a small domain ontology is shown in Figure 3. This is our **Way 1** of using ontologies for NLP.

The ontological classes become named entity types (e.g., "Company" is a type of entity of interest for the M&A domain), where we target the most specific types for automatic labeling and use high-level types for inference-purposes only. More details on how the ontology affects the NLP customization process are included in Section 6. In this section, we discuss three potential issues that may arise when creating ontologies as NLP-labeling schemas and our solutions.

From our experience, SMEs and ontologists tend to come up with labels that are fairly general as well as abstract, meaning "not widely used in text". This limits the application of the ontology in a data-driven context. For example, an abstract label "software agent" is unlikely to be mentioned in a document and will require explicit annotation of the training samples, whereas "software system" may be found in text and leveraged directly.

Similarly, for ontological relations between classes, labels closer to actual text fragments can be more reliably used for weak supervision. Therefore, an ontology with abstract labels can be made more NLP-friendly by adding alternative labels or introducing more specific subclasses. For example, a label IS_BUYER_IN is unlikely to occur in the text. While the system can parse the label, identify *buyer* as the main word, and even link it to the verb *buy*, this process may be unsuitable for a bigger ontology with more complex

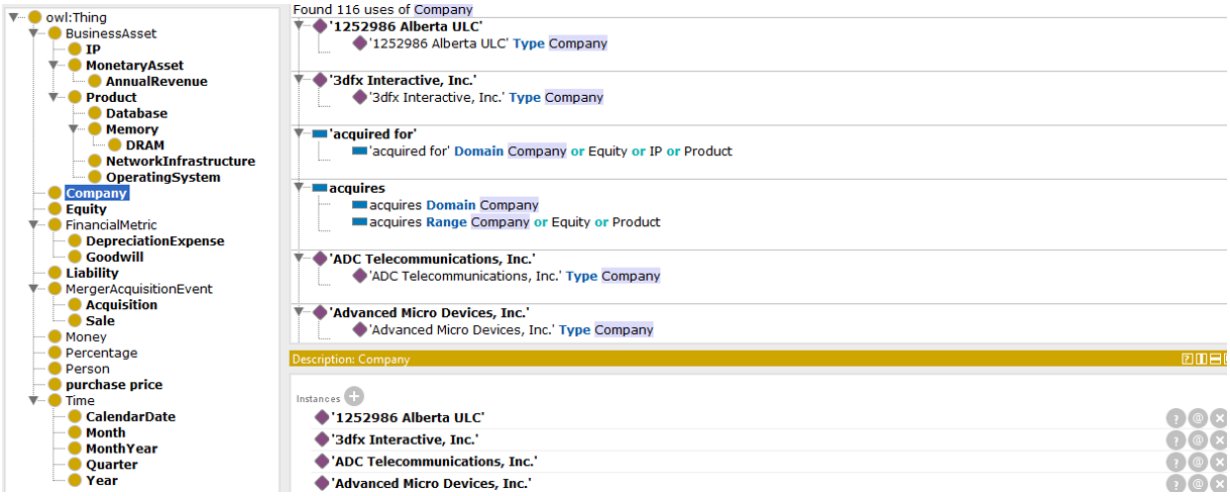

Figure 3: A sample ontology for Merger & Acquisition domain (displayed in Protégé). Its list of classes include "Company", "MergerAcquisitionEvent", various business assets, and several attributes of M&A events: "Percentage", date, "PurchasePrice". These classes are linked by various properties, for instance, ACQUIRES **– done by a "Company" to another "Company", "Equity", or "Product". Additionally, instances are included for certain classes, e.g., "3dfx Interactive, Inc.", "ADC Telecommunications, Inc.", and "Advanced Micro Devices, Inc." are companies.**

labels. Adding alternative labels, such as ACQUIRES, provides additional hints to the system and boosts the confidence for noun-verb augmentation, as well as expansion of the verbs and nouns with their synonyms.

And, while ontology classes are often too abstract, relation types tend to be too specific, as it is tempting to define a separate relation type for each meaningful pair of classes. This results in additional effort to annotate training data for each relation type. This process is optimized if the number of relations targeted for NLP is limited to fewer, more general types. The more specific relation can be inferred downstream based on the underlying general relation type and the types of its arguments. For example, COMPANY - HAS_INCOME - INCOME and COMPANY - HAS_REVENUE - REVENUE are expressed in text using similar linguistic patterns and, therefore, can be generalized to COMPANY - HAS_METRIC - METRIC (assuming INCOME and REVENUE are subclasses of METRIC).

Some relation declarations that look legitimate as part of an ontology may introduce unwanted ambiguity when used for labeling text. For example, "Marry" (*to wed*) can be defined as an ontological class with links to the "Person" class. However, identifying these concepts and relations in "Alice and Barbara are married to Andrew and Bob, respectively" will result in the "married" event/state being linked to all four "Person" concepts as its participants, which obscures the pairing expressed in the sentence. It is better to introduce a relation type MARRY and use it to connect two instances of "Person" (e.g., "Alice" to "Andrew" and "Barbara" to "Bob").

5.1 Automated Taxonomy Creation

The manual creation of ontologies can be a labor-intensive process. In this section, we describe how NLP can be used to create a taxonomy from unlabeled text semi-automatically.

This process consists of the iterative expansion of an upper-level ontology in a semi-automated manner via mining unlabeled corpora for additional concepts and relations, thus significantly reducing the amount of manual work required of SMEs.

Our concept detection methods range from the detection of simple nominal and verbal concepts to more complex NEs and phrasal concepts. This hybrid approach to *concept extraction* makes use of machine learning classifiers, a cascade of finite-state automata, and lexicons to label more than 80 types of concept classes, including *technical terms* such as *insider trading*, *causal reasoning*, and *organizations* such as *World Bank*, *United Nations*, and *ASEAN*, various types of *locations*, *quantities*, *numerical values*, etc.

The domain ontology generation module extracts and organizes key domain concepts from a document's section headers. Our algorithm to automatically extract domain relevant concepts and semantic relations consists of the following steps:

1. Process the document content to identify:

   (a) Section header information such as part, chapter, section, and subsection titles.

   (b) Textual content of headers (i.e., the natural language title of the section without its numbering/label information; e.g., *ASEAN Free Trade Area (AFTA)* from "Chapter 5. ASEAN Free Trade Area (AFTA)").

2. Extract concepts from section titles to identify the overarching terms relevant to the domain. For instance, from the title *ASEAN Free Trade Area (AFTA)*, we extract the concepts: *ASEAN Free Trade Area*, *AFTA*, *Free Trade*, *Trade*, and *Trade Area*.

3. Extract relationships between:

   (a) Concepts in the same header. For instance, from the title *ASEAN Free Trade Area (AFTA)*, we extract the relations: *ASEAN Free Trade Area* SYNONYMY *AFTA*, *ASEAN Free Trade Area* RELATED *Free Trade*, and *Free Trade* ISA *Trade*

   (b) Concepts in a parent header with concepts in a child header linked via:

      i. Optional seed taxonomy that was manually created by an SME. For example, we add the semantic relation *Trade Union* SYNONYMY *Labor Union* based on an entry in the SME seed taxonomy and merge their respective sub-trees under a single node.

      ii. Optional seed taxonomy that was automatically created from index terms.

      iii. Semantic relations extracted from the textual content of identified section headers. For example, we add an ISA semantic relation between two concepts *implication* and *reasoning* occurring in a child and parent section header, respectively, because the relation was extracted from a sentence included in the child section "The use of reasoning, such as implication...."

4. Classify the concepts and relations into an ontological hierarchy using the SYNONYMY and ISA derivation procedures described in [5].

The following three semantic relations are encoded in domain ontologies:

1. SubClassOf (ISA): specifies that a concept is sub-class of another concept, for example, *implication* ISA *reasoning*.

2. Synonymy (SYNONYMY): connects two synonymous concepts, for example, *WHO* SYNONYMY *World Health Organization*.

3. RelatedTo (RELATED): specifies that two concepts are related to one another. It is a coarse-grained semantic relation that includes PART-OF/CONTAINS, ASSOCIATED-WITH, SEE-ALSO relations. Examples include *absence of proof* RELATED *causal reasoning*, *part-of-speech tagging* RELATED *Markov chain*. All these relation types are merged into one since SMEs do not see the practical use of the fine-grained representation as well as to simplify the annotation task.

Once created, SMEs curate the domain ontology generated using the steps described above, which is much faster than manually creating the ontology from scratch.

6 Ontology-Based NLP Customization

As shown in Figure 2, once the knowledge extraction goals are captured in the ontology, we proceed with the customization of the NLP pipeline. We note that we are not developing a new suite of NLP tools for the new domain/extraction schema. We are employing an existing set of NLP modules that are now re-configured for the new domain/ontology. New domain-specific models are created to be used during the NLP of domain-specific input documents.

This rapid NLP customization process includes training and configuring NLP tools to extract mentions of ontology elements from natural language texts, mainly NEs (corresponding to the ontology's classes) and relations between them (i.e., the ontology's object properties) to form a knowledge graph that instantiates the ontology from text.

In order to identify the set of target NE types, we use the ontology's ISA-hierarchy of classes (a small sample is shown in Figure 4).

First, the most specific classes are used to tag instances in text. We use a leaf class's *prefLabel* to create two corresponding named entity types that will be automatically output by the NLP as labels of matching words or phrases. In order to distinguish between mentions of classes and mentions of instances in text, we use a special "_G" suffix (stands for "general") for class-based named-entity types. For example, for "Country" (leaf class in Figure 4), two named entity types are generated: (1) COUNTRY_G, which will label the word "country", and (2) COUNTRY, which is the correct named entity label for "USA". Using this setup for ontology-based named entity recognition (NER), "[COUNTRY US]

is one of the G7 [COUNTRY_G countries]." is an accurate labeling of the text using the ontology as input.

However, according to the ontology, a "Country" is also a "Geopolitical entity" as well as a "Political agent". If we were to extend our tagging framework beyond the leaf classes of the ISA-hierarchy, this would mean "[POLITICAL_AGENT [GEOPOLITICAL_ENTITY [COUNTRY US]]] is one of the G7 [POLITICAL_AGENT_G [GEOPOLITICAL_ENTITY_G [COUNTRY_G countries]]]", which is a cumbersome setup for the NER module as it would require annotations, models and resources for the additional NE types.

Nonetheless, this hierarchical information can be converted into inference rules that can be used within the knowledge base created using all the leaf-class label triples to also derive the more general information about each word/phrase. Therefore, "Country" ISA "Geopolitical entity" becomes `?X example:neType ''COUNTRY'' → ?X example:neType ''GEOPOLITICAL_ENTITY''`, which produces `<http://example.com/US> example:neType ''GEOPOLITICAL_ENTITY''` from `<http://example.com/US> example:neType ''COUNTRY''` (triple created by serializing the NLP output into RDF format).

In addition to making use of the ontology classes to create named entity types and inference rules involving these types, the labels of individuals defined in the ontology are collected and grouped to create lexicons for their corresponding classes and used as training data. In the examples shown above, "USA" (as the `rdfs:label` of the individual URI) as well as any synonyms included in the ontology for this individual of class "Country" are listed as entries for the COUNTRY lexicon (e.g., "United States of America"; "USA"; "US"; "United States"). We note that any lexicons automatically generated using ontology content can be directly leveraged by the NER module during NLP.

When it comes to ontological relations, we make use of (1) the property's `rdfs:label`[6] as the name of the target domain/custom relation, as well as (2) its defined domain and range classes/sets of classes. Other relational properties, such as transitivity, are mapped to inference rules.

If no `rdfs:label` is provided within the ontology, a value can be derived by parsing the URI (uniform resource identifier) of a given class, individual, and/or property to pinpoint its local name and applying camel case or underscore-to-space transformations to separate any multiple tokens that make up a name. For example, (1) as a class,

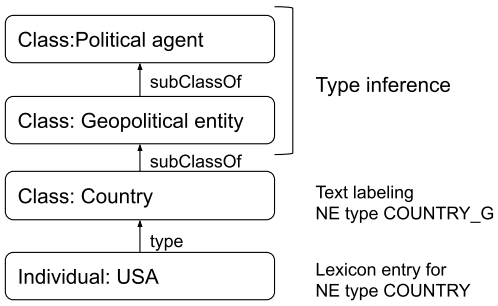

Figure 4: Mapping an ontology's class hierarchy to named entity types. Leaf classes (e.g., "Country") produce two corresponding named entity types for NLP (e.g., COUNTRY_G **and** COUNTRY**). All other classes are used for inference only (e.g.,** COUNTRY_G → GEOPOLITICAL_ENTITY_G**). All instance labels are used to produce lexicons for their respective NE type.**

`<http://example.com/GeopoliticalEntity>` is converted into GEOPOLITICAL_ENTITY and GEOPOLITICAL_ENTITY_G named entity types; (2) the individual `<http://example.com/United_States>` produces "United States" as a lexicon entry; and (3) the property `<http://example.com/is-buyer-in>` defines a new relation type IS_BUYER_IN.

6.1 Annotation Schema

A domain ontology formalizes a labeling schema agreement between SMEs, which we automatically leverage via integration with the annotation tool - this is our **Way 2** of leveraging ontologies.

In our practice, we use the BRAT annotation tool[7] [34] for named entity and relation labeling, but the same technique applies to other annotation tools. The BRAT annotation tool must be configured with fixed sets of labels for named entity and relation annotations.

Additionally, relation types may have restricted sets of entity types for their domain and range arguments. This information can be directly translated from the input ontology's definitions. However, since BRAT's configuration files (e.g., annotation.conf) use its own YAML-like format (Figure 5), we implemented a simple converter from OWL format that will list the types of NEs targeted for NLP (e.g., COMPANY, COMPANY_G, etc.) as well the custom relation types that we aim to identify as defined in the ontology (e.g., IS_BUYER_IN linking COMPANY to ACQUISITION_G). Having a semantically-accurate configuration file for the annotation tool leads to correct annotations, since non-complaint labeling is not permitted.

[6] `https://www.w3.org/TR/rdf-schema`

[7] `https://brat.nlplab.org/`

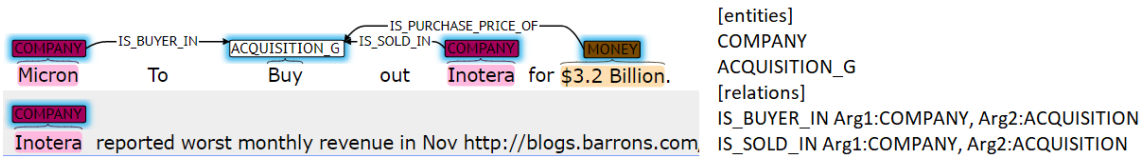

Figure 5: BRAT annotation example and configuration fragment

This automatic process of producing BRAT configurations from an ontology is especially vital for ontologies undergoing active development. If the input ontology is updated, it can be used to automatically re-generate BRAT configurations, therefore ensuring up-to-date annotations. Updated annotation configurations can also be used to validate previously annotated documents against a newer version of the ontology and any conflicts can then be reconciled.

The entire process helps (1) keep SMEs in agreement with respect to extraction targets, (2) represent the agreement in machine-readable form, and (3) reduce the annotation error rate.

6.2 Ontology-Based Lexicons for NER

We use the set of instances defined within an input ontology to generate lexicons for NER - our **Way 3** of leveraging ontologies. By default, we use SKOS[8] `prefLabel` and `altLabel` properties of an individual to identify its canonical form and synonyms, respectively. These are then added as entries within a lexicon for the named entity type corresponding to the individual's class. However, custom properties (e.g., `asText`) may be also used to convey this information within an ontology and can be easily added as configurations for the ontology-as-input tool. For instance,

```
example:USA
    a                 example:Country;
    skos:prefLabel    "United States of America";
    skos:altLabel     "USA";
    skos:altLabel     "US";
    skos:altLabel     "United States";
    example:asText    "US of A";
```

is used to create an entry in the COUNTRY lexicon: "United States of America"; "USA"; "US"; "United States"; "US of A".

We note here the distinction we make between the canonical form of a concept/entity and its synonyms. Since the normalized form is used for indexing the extracted knowledge and when matching a question to its answer as part of question answering and other query-based applications, we recommend the separation of the canonical form from alternative labels.

[8] `https://www.w3.org/2004/02/skos`

Alternatively, if an ontology is accompanied by a knowledge base with individuals for ontological classes[9], corresponding lexicons can be created by querying the knowledge base and performing any required transformations on these instances in a fashion similar to the procedure described above.

6.3 Ontologies for NER Control

In addition to using an ontology as a source for named entity types and their corresponding lexicons, we support custom OWL properties that provide SMEs with additional control over the system's identification of names. Without such controls, the development cycle involves: (1) SMEs providing ontology and annotations, (2) data scientists configuring the tools and generating results, (3) SMEs validating these results, providing more annotations and communicating feedback to the data scientists. We shorten this feedback loop by adding tool configuration parameters directly into the ontology so that SMEs can change them and run experiments independently.

While implementation details depend on the actual software used to recognize NEs, the general approach is to augment ontologies with additional triples that define software behavior. These triples can be stored in a separate file so that the schema itself is not polluted with application-specific triples.

Within our application, we allow SMEs to setup part-of-speech (POS) constraints for candidate tokens. For example, we restrict instances of class "Liability" to match only nouns by including an annotation property to a regular expression literal: `:Liability :instancePOS ''NN.*''`. Please note that we distinguish between POS restrictions for classes and instances.

Additionally, and, in a similar fashion, we allow SMEs to control the amount of fuzziness that can be used when identifying names of a particular type in text as well as any case variations that may exist within a class. More than 10 different types of variations are currently supported within our application to account

[9] Best practices dictate that ontologies define the model/schema of the domain and known individuals (instances of ontology classes) are stored outside of the ontology, especially when there are many such known instances.

for hyphenated/no-hyphenated spellings (e.g., "Anne-Marie" vs. "Anne Marie"), special characters or removal of stopwords (e.g., "AT&T" vs. "ATT"), abbreviated company names (e.g., "Microsoft Corporation" vs. "Microsoft Corp." vs. "Microsoft"), and common spelling variations (e.g., "MySql" vs "Mysql").

SMEs can also indicate whether the surface form of a word should be matched as is against a lexicon or if its lemma should be used by including a custom annotation property in the input ontology, e.g., `:instanceLexOptions ''add-lemmas''`.

All these options save considerable manual effort without compromising the output of the NLP tools.

6.4 Ontology-Based Relation Extraction

Within our NLP-based applications, the NLP pipeline includes a suite of tools that begin by performing low-level NLP tasks, such as tokenization, POS tagging, lemmatization, and continues to NER and syntactic parsing. Lastly, tools that make use of this underlying lexical and syntactic knowledge identify the semantic relations conveyed by the text.

We divide the semantic parsing process into two steps. First, a standard relation extraction module identifies generic relations, including semantic roles, e.g., AGENT, THEME, LOCATION, MANNER, etc. as well as nominal relations, e.g., IS-A, PART-WHOLE, KINSHIP, and others [24]. These relations abstract out linguistic structures and provide a higher-level representation when compared with one based on syntactic parsing alone. For instance, *Amazon* is the AGENT of *acquire* (i.e., AGENT(*Amazon*, *acquire*)) for both "Amazon acquired ..." and "... acquired by Amazon", which have very different surface forms as well as underlying syntactic dependencies.

However, the standard relation types provide little value to an end-user who targets a custom set of relations relevant to their domain/application, e.g., IS_BUYER_IN.

Therefore, in order to identify custom high-level semantic relations defined in an input ontology, we make use of Semantic Calculus [10] and its semantic rules, where the standard relations serve as building blocks. In a nutshell, Semantic Calculus rules are conditional first-order logic axioms that add (or delete) the annotations specified on the right-hand side of a rule whenever the patterns specified on its left-hand side are instantiated by the underlying semantic representation of the input text. The following Semantic Calculus rule: $\mathrm{ne}(X, COMPANY)$ && $\mathrm{ne}(Y, ACQUISITION\_G)$ && $\mathrm{R}(X, Y, AGENT) \rightarrow \mathrm{R}(X, Y, IS\_BUYER\_IN)$ creates a new instance for the IS_BUYER_IN relation between a COMPANY name and an ACQUISITION_G event when the company is the agent of the event. Given "Amazon acquired ...", this Semantic Calculus rule will identify IS_BUYER_IN(*Amazon*, *acquire*). More information on a viable implementation of Semantic Calculus that makes use of RDF and SPARQL technologies is included in Section 7.2.

Whereas one can manually create Semantic Calculus rules as well as learn these rules from human annotations of the ontological relations on domain-specific training data, we use the ontology's relation declarations to automatically generate an initial set of rules for custom relation extraction based on relation labels and their corresponding domain/range restrictions - **Way 4**.

Here, we rely on the assumption that the ontology includes object properties with names that can be linked back to domain-relevant texts, as opposed to very abstract naming schemes that cannot be utilized (by an automated system) in conjunction with the data that it models. For instance, "isOwnerOf" is a better name for an ownership property when compared to "isHolderOf", which is less likely to be found in texts that discuss ownership.

Given this assumption, we use the label information of an object property (its `skos:prefLabel` or `rdfs:label`, if defined; otherwise, its local name as parsed out from its unique URI) to create an artificial "natural language" sentence about the object property. This sentence combines the information provided within the ontology for the custom relation: its label and domain/range classes as "`<domain> <label> <range>`.", where we know that the custom relation holds between the `<domain>` and `<range>` tokens. Given this effortless "gold" annotation, semantic rules can be automatically generated: (1) the rule's left-hand-side constrains are a direct translation of the underlying semantic graph produced for this sentence, and (2) the right-hand-side is the new custom relation predicate.

For example, for an object property "Bought" between "Company" and "Company":

```
example:Bought
    skos:prefLabel   ''bought'';
    skos:altLabel    ''purchase'';
    skos:altLabel    ''acquired'';
    rdfs:domain      example:Company;
    rdfs:range       example:Company;
```

a basic "Company bought company." sentence is generated, where BOUGHT(*Company, company*) holds. The standard NLP understanding for this sentence produces: an AGENT(*Company*, *bought*) relation as well as a THEME(*company*, *bought*), meaning that a COMPANY/COMPANY_G entity is the AGENT of verb "buy", which also has as its THEME another COMPANY/COMPANY_G. Therefore, an initial semantic rule can be automatically generated as:

$$\text{lemma}(A, \text{“}buy\text{”}) \;\&\&\; (\text{ne}(X, COMPANY) \;||\; \text{ne}(X, COMPANY\_G)) \;\&\&\; \text{R}(X, A, AGENT) \;\&\&\; (\text{ne}(Y, COMPANY) \;||\; \text{ne}(Y, COMPANY\_G)) \;\&\&\; \text{R}(Y, A, THEME) \;\rightarrow\; \text{R}(X, Y, BOUGHT)$$

Please note that both COMPANY and COMPANY_G are named entity types derived from `example:Company` (the ontological class) and become part of the automatically generated semantic rule for BOUGHT.

This initial rule identifies instances of a company purchasing another company in natural language text (e.g., BOUGHT(“Amazon”, “Whole Foods”) from “Amazon bought Whole Foods”).

Any additional labels defined for this object property in the ontology are used to further expand this rule to provide additional coverage for relation extraction. For example, the object property “Bought” has “purchased” and “acquired” as alternate labels and, thus, the $\text{lemma}(A, \text{“}buy\text{”})$ predicate of rule shown above can be expanded to $(\text{lemma}(A, \text{“}buy\text{”}) \;||\; \text{lemma}(A, \text{“}purchase\text{”}) \;||\; \text{lemma}(A, \text{“}acquire\text{”}))$, which can now identify the BOUGHT relationship in “Amazon acquires Whole Foods” or “Amazon purchases Whole Foods”.

When no alternate labels are included in the ontology, we use the degree of semantic similarity between two tokens to ease the strict nature of a lemma-only check. For instance, $\text{similarity}(A, \text{“}buy\text{”}, 0.8)$ matches words/concepts whose lemmas have a word embedding-based cosine similarity to “buy” greater than 0.8.

Additionally, paraphrasing can be used to automatically generate alternate labels and improve recall. Lexical paraphrases can be used within additional lemma constraints (as shown above for any `skos:altLabels` defined within the ontology). However, syntactic paraphrases of the initial sentence may provide drastically different semantic rules. For instance, “Company bought company.” can be paraphrased as “Company is buyer of company.”, which results in a rule that uses an ISA relationship:

$$\text{lemma}(A, \text{“}buyer\text{”}) \;\&\&\; \text{R}(X, A, ISA) \;\&\&\; (\text{ne}(X, COMPANY) \;||\; \text{ne}(X, COMPANY\_G)) \;\&\&\; (\text{ne}(Y, COMPANY) \;||\; \text{ne}(Y, COMPANY\_G)) \;\&\&\; \text{R}(Y, A, THEME) \;\rightarrow\; \text{R}(X, Y, BOUGHT)$$

We note that changes from active to passive voice for the verb denoting the custom relation have no effect on the semantic rules generated for the sentence – the underlying semantic graph, which contains the AGENT and THEME relations from COMPANY/COMPANY_G to “buy” for “Company bought company.” remains unchanged for “Company was bought by company”.

When several classes serve as the domain or range of a object property, their corresponding named entity types are combined into a single type constraint using the || (“or”) operator. For instance,

```
example:Bought
    ...
    rdfs:range      example:Company;
    rdfs:range      example:Equity;
    rdfs:range      example:Asset;
```

is automatically translated to the following conditions for the Y variable (the second argument of the BOUGHT relation):

$$(\text{ne}(Y, COMPANY) \;||\; \text{ne}(Y, COMPANY\_G) \;||\; \text{ne}(Y, EQUITY) \;||\; \text{ne}(Y, EQUITY\_G) \;||\; \text{ne}(Y, ASSET) \;||\; \text{ne}(Y, ASSET\_G))$$

When no domain/range restrictions are provided formally for an ontological relation, but a triple store contains instance information for the input ontology, a given relation’s instances can be used to (1) identify likely domain/range classes for the object property and (2) constrain the relation’s corresponding semantic rules to the type combinations that occur in the data. We note that this is a situation encountered when natural language questions are used to query structured data already stored within a graph database.

Another aspect that must be handled when it comes to deriving semantic rules for ontological relations is negation. Recognizing and representing negations [25] is a complex task and a detailed discussion on this topic is out of the scope of this article. However, our Semantic Calculus rule generation approach accounts for frequently used negation expressions to remove a positive/stated relation and generate a negated relation. We distinguish between negation of the argument and negation of the relation. For example, we extract a ACQUIRED_BY_NEGATED relation for “List companies that are not acquired.”, where *acquired* is negated (the company took part in no acquisition event), but ACQUIRED_BY_NEGATED_ARG2 for “List companies that are not acquired by Amazon”, where *Amazon* is negated (companies may have been acquired by companies other than Amazon).

6.5 Ontology for Data Augmentation

Ontologies identify and store important domain-specific lingo, for instance, synonyms and abbreviations, which can be used to augment the training data required by data-driven approaches (**Way 5**). For data augmentation purposes, we replace ontological concepts with their corresponding synonyms (as defined within an ontology)

in annotated texts to generate semantic variations of already labeled data, therefore, increasing the size of a training dataset. This is a straightforward way for data-driven approaches to benefit from ontological knowledge.

For instance, given an already manually annotated sentence that mentions "United States of America", we can quickly replace this phrase with its synonyms as provided by an ontology (e.g. "USA", "United States", "US", etc.), to produce new sentences that are semantically equivalent to the manually annotated sentence and, therefore, can automatically be assigned the annotations manually added to the original sentence at a fraction of the cost.

7 RDF Framework for NLP

In addition to all the benefits that ontologies directly bring to the NLP task, there are several other advantages to using ontologies and ontology-driven frameworks for NLP.

7.1 Standardized Format for NLP Output

As **Way 6** of leveraging ontologies as a framework, we use an RDF standard format [12] to represent the knowledge extracted by an NLP pipeline from input texts/documents. While the detailed schema is out of the scope of this article, the key elements include the document and its corresponding metadata, sentences, tokens, named entities, and relations, as well as text fragment labels (Figure 6). In addition to resources representing the knowledge extracted directly from the document's content, we use meta named entity resources for concepts provided as metadata, e.g., author, publication date, collection name, etc.

Most commonly, ontological relations are compactly represented as properties: `<Ne1Uri> example:IS_BUYER_IN <Ne2Uri>.`

However, these relations can be represented as resources when we need to include additional information, such as confidence score or source semantic rule, while their `example:lhs` and `example:rhs` properties link to their respective left-hand-side and right-hand-side argument URIs.

```
<RelationUri> example:type ''IS_BUYER_IN'';
    example:lhs <Ne1Uri>;
    example:rhs <Ne2Uri>;
    example:confidence ''0.86'' .
```

Within our application, we use a separate namespace for such properties to allow easy querying for all relations without the need to list their types.

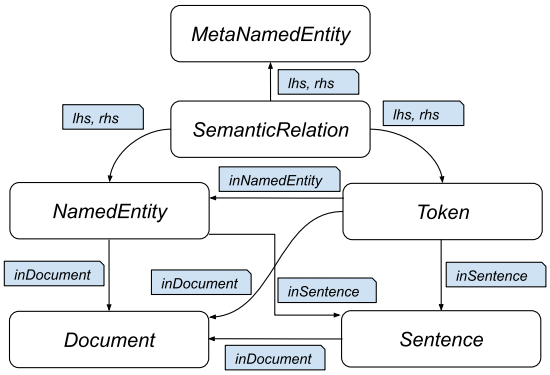

Figure 6: RDF schema to represent NLP output: identified concepts (*NamedEntity*) and their corresponding tokens with links to source sentences and documents, as well as extracted semantic relations and their corresponding arguments

We note that the RDF representation of the knowledge extracted from text contains a unique URI for each mention of a concept in the text as opposed to the URI of the concept itself. In other words, for "Microsoft bought LinkedIn", we represent *Microsoft* as a unique text span (e.g., `<http://example.com/doc1/Microsoft/0-8>`) and link it to the named individual "Microsoft" of type COMPANY from the ontology (`<http://example.com/Microsoft>`), thus, making it possible to preserve the origin of each concept and relation as well as their connection to the ontology.

7.2 RDF and SPARQL for a Semantic Rule Engine

With deep learning methods gaining most of the attention in current NLP research, in real-world projects, there are cases where writing a few rules is more efficient than collecting a large set of annotated samples for training. Within our NLP pipeline, the Semantic Calculus tool [10] makes use of conditional first-order logical rules (without functions and quantifiers), whose left-hand-side must unify with a text's underlying semantic information, so that its right-hand-side is activated. More specifically, the rule $\mathrm{lemma}(X, \text{“}Microsoft\text{”}) \rightarrow \mathrm{ne}(X, COMPANY)$ matches all tokens with lemma "Microsoft" and labels them as COMPANY names.

Since we already use RDF to represent the information extracted by the NLP pipeline (Section 7.1), SPARQL becomes the natural choice as the back-end for these semantic rules (**Way 7**). Therefore, we use SPARQL queries to manipulate the RDF representation of the

underlying NLP output. Specifically, a semantic rule's left-hand-side is converted into a SELECT query to bind the rule's variables, and then its right-hand-side actions are applied for each of the bindings. For the sample rule shown above, its left-hand-side predicate is converted into the following SPARQL query:

```
SELECT ?X
WHERE {
  ?X example:lemma ''Microsoft'' .
}
```

Its right-hand-side predicate is translated to a set of triples that will be added to the text's semantic graph for each `?X` returned by the SPARQL query (e.g., `?X example:neType ''COMPANY''`).

We note that SPARQL queries can be formulated for all logical operations used within Semantic Calculus rules. For left-hand-side operators, we use: (1) conjunction, `and` (&&), (2) disjunction, `or` (||), which is converted to `UNION`, and (3) negation, `not` (!), which is converted into SPARQL using `MINUS` or `FILTER NOT EXISTS` depending on the SPARQL standard supported by the graph store. For instance, as the left hand side of a rule shown above,

$$\text{lemma}(A, \text{“}buy\text{”}) \;\&\&\; (\text{ne}(X, COMPANY) \;||\; \text{ne}(X, COMPANY\_G)) \;\&\&\; \text{R}(X, A, AGENT)$$

becomes

```
SELECT ?A ?X
WHERE {
  ?A example:lemma ''buy'' .
  { ?X example:neType ''COMPANY'' } UNION
  { ?X example:neType ''COMPANY_G''} .
  ?X example:AGENT ?A
}
```

The right-hand-side of rules are restricted to the conjunction of predicates to support multiple atomic actions.

To achieve this, for each rule predicate (e.g., $\text{lemma}()$, $\text{ne}()$, $\text{R}()$), three templates are defined:

1. triples that need to be matched, when the predicate is used on the left-hand-side of a rule;
2. triples that need to be inserted, when the predicate is used on the right-hand-side;
3. triples that need to be deleted, when the predicate is negated on the right-hand-side of a rule.

By making use of these standard frameworks as described above, semantic rules can be created not only to produce new instances of custom semantic relations/object properties (Section 6.4), but also custom NEs (when the entity labeling decision relies heavily on the semantic context of the ambiguous name), classification labels, updated part-of-speech tags, etc. For instance, money values can be reliably tagged by ignoring their context using regular expressions. However, when identifying prices, certain money values must be re-labeled based on their context and a simple semantic rule can be used for this purpose $\text{ne}(X, MONEY)$ && $\text{lemma}(Y, \text{“}pay\text{”})$ && $\text{R}(X, Y, THEME) \rightarrow \text{ne}(X, PRICE)$.

By using RDF and SPARQL, one can quickly extend the set of predicates for custom applications, e.g., validating that a question contains all information necessary to provide a meaningful answer, or blending the knowledge extracted from the text with external knowledge to access historical or reference data.

7.3 Leveraging ISA-Hierarchies within Rules

Having detailed how RDF and SPARQL can be used to implement a semantic rule engine, we now describe how a semantic rule developer can benefit from using a domain-specific ontology. More specifically, semantic rule engines configured to use the class hierarchy formed by the ontology's `rdfs:subClassOf` property instances require a considerably smaller number of generalized and compact semantic rules that are easy to read and/or edit. With a new predicate $\text{neIsaType}(T, SUPER\_T)$ that will check if a named entity type can be generalized to another (according to the domain ontology's ISA hierarchy), rules can be greatly simplified to reduce a disjunction of several $\text{ne}(X,...)$ predicates, e.g.,

$$(\text{ne}(X, REVENUE) \;||\; \text{ne}(X, MARKET\_SHARE) \;||\; \text{ne}(X, SALES) \;||\; ne(X, CASH\_FLOW) \;||...)$$

to only two:

$$\text{ne}(X, T) \;\&\&\; \text{neIsaType}(T, FINANCIAL\_METRIC)$$

With this notation, the named entity type itself becomes a variable. With this rule generalization framework, semantic rules do not require updates when new classes are added within existing hierarchies. For instance, rules that include $\text{neIsaType}(T, FINANCIAL\_METRIC)$ require no changes if a new financial metric, e.g., “Net Margin”, is added to the financial ontology.

We note that the $\text{neIsaType}()$ predicate works for both “regular” named entity types as well as general ones (i.e., _G types). However, the ISA hierarchies are parallel and cannot be combined.

Based on our experience, the most common use for rules that require the $\text{neIsaType}()$ predicate is the classification of a generic NE (e.g., MONEY, DATE,

NUMBER, etc.) into a domain specific sub-class (e.g., REVENUE, SALES, GROSS PROFIT, etc.). Semantic rules are preferred for these tasks because the classification requires contextual information; it relies on the semantic relations around the value of interest. For instance, given "The company's revenue for 2019 is $4M.", *$4M* is not only a MONEY value, but also a REVENUE because VALUE(*$4M*, *revenue*) – it is the value of *revenue* (a REVENUE_G instance). The semantic rule implementing this reasoning is

$$\mathrm{ne}(X, REVENUE\_G) \;\&\&\; \mathrm{ne}(Y, MONEY) \;\&\&\; \mathrm{R}(Y, X, VALUE) \;\rightarrow\; \mathrm{ne}(Y, REVENUE)$$

However, the same linguistic pattern is used, in natural language, for other financial metrics, e.g., "The company's gross profit for 2019 is $1.6M." and without the $\mathrm{neIsaType}()$ predicate, corresponding rules are required for each sub-class of FINANCIAL_METRIC_G. In order to simplify the semantic rule set, one can use

$$\mathrm{ne}(X, T) \;\&\&\; \mathrm{ne}(Y, MONEY) \;\&\&\; \mathrm{neIsaType}(T, FINANCIAL\_METRIC\_G) \;\&\&\; \mathrm{R}(Y, X, VALUE) \;\rightarrow\; \mathrm{neByClass}(Y, T)$$

as the single rule that accounts for all financial metrics involving MONEY values. We note that, here, T is a general type (a _G type, e.g., REVENUE_G) and the $\mathrm{neByClass}(Y,T)$ predicate labels Y with T's non-general variant, e.g., REVENUE, to indicate that Y is an instance of the class.

8 Statistics on Use Cases

Quantitative characteristics of ontologies and generated artifacts for the use cases summarized in Section 3 are listed in Table 2.

Based on our experience, we note the following:

1. Ontologies with larger amount of classes are updated more frequently. Combining ontologies, annotation, and NLP allows us to streamline the update steps.

2. Generating rules automatically is a considerable time saver, as it takes an experienced NLP engineer about 5 minutes to write a single rule.

3. The effort required to create an ontology might seem an unwanted upfront investment. In practice, however, small ontologies can be created in a few days. Larger ontologies are produced with data-driven approaches or by borrowing concepts from existing industry ontologies.

9 Ontologies for NLP Applications

In our experience, most NLP applications are centered around semantic search and question answering (QA). Users are interested in responses to queries, questions, or commands that need to be answered or executed against a knowledge base. The user interface can range from interactive visualization dashboards to chatbots, while the QA component supports the capabilities. For example, to see a breakdown diagram of Apple's revenue by country, the system needs to pull data matching a query like "List revenue of Apple by country" and display it. Similarly, a chatbot interface would consult a QA component to extract the answer, and potentially expand the functionality with clarifying user questions by asking "Do you mean ...?" counter-questions.

The knowledge base used to address user questions can be (1) an enterprise graph store already populated with structured data (e.g., personnel/company directory information) or (2) a semantic graph produced from unstructured documents using NLP tools customized for a given ontology. The extraction supports multiple document formats – plain text, HTML, PDF, email, Microsoft Word, etc. – and targets textual parts of the documents as well as embedded tables.

9.1 Semantic Search and Question Answering

Ontologies help semantic search applications by providing synonyms and hyponyms (ISA-hierarchies) for search terms in order to expand user queries [26]. Furthermore, ontologies can guide the indexing schema, where named entity types derived from the ontology can be indexed as separate fields and have separate boosting weights to provide more flexibility for relevance tuning. Also, using ontologies and their correspondence between class mentions (e.g., "price" as PRICE_G) and instance mentions (e.g., "$13.7M" as PRICE), queries that specify a class retrieve instances from the indexed documents. For example, given "price of Amazon's acquisition of Whole Foods", documents containing PRICE NEs are more relevant than ones that include just PRICE_G (e.g., "price").

In addition to semantic search, question answering systems benefit by using ontologies in two ways: (1) using RDF/SPARQL as a technology framework and (2) using type hierarchy as a source of knowledge [36], **Way 10**. By employing a customized NLP pipeline (Section 6), an input natural language question is converted into a semantic graph consistent with the ontology-based schema. Within this graph, the focus of the question, the concepts (named entities or tokens) that represent the requested/unknown information, is marked to become a variable. Next, the graph is serialized into

Table 2: Quantitative characteristics of domain ontologies and counts for automatically-generated NLP artifacts

#	Domain	# Classes	# Triples	# Lexicons	# Lexicon entries	# Rules
1	Medical	7	6,735	14	6,456	5,189
2	Research	11	13,284	22	12,583	126
3	Aerospace	15	42	30	80	62
4	Auto	14	97	28	4272	32
5	Retail	2,200	270,000	4,400	13,000	58
6	Taxes	4,734	12,458	9,468	8,475	120
7	Human Resources	253	14,199,818	506	808,646	1,180

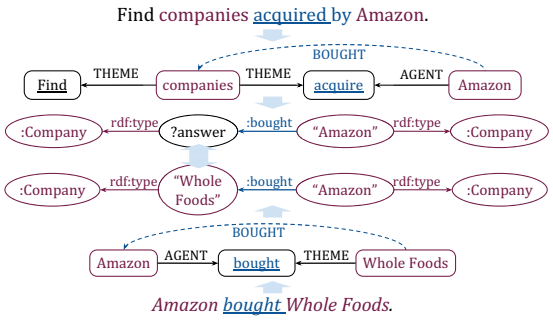

Figure 7: Ontology-based Question Answering is done by matching semantic graphs of questions to semantic representations of documents; all consistent with the schema defined within a given ontology

	2018	2019
Alice Smith	$ 130,000	$140,000
Bob Johnson	$ 120,000	$135,000

Table 5. Employees salary.

Figure 8: Example for table understanding. Each content cell is linked to its row-header (left-most column) and its column-header (top row).

a SPARQL query or other formal query language [37], which is then executed against the graph store to retrieve precise answers (Figure 7).

As part of this process, the ontology provides domain-specific synonyms and hypernyms that can be used for term expansion and for matching focus terms to answers – **Ways 8, 9**. In some use cases, these expansions can be automatically learned from the data or bypassed using word embeddings. However, for complex domains with little data available, ontologies are a convenient tool for SMEs to convey their understanding explicitly.

For example, "results" can mean particular financial metrics for different applications or user roles, which can be represented using different RESULT classes and their corresponding `rdfs:subClassOf` relations to particular metric classes, such as *gross domestic product* for a global analyst, or *company's revenue in the region* for a business analyst. Then, queries like "What are China's 2019 results?" can be interpreted as either "What is China's 2019 GDP?" or "What was our revenue in China in 2019?".

Ontologies help identify answers more intelligently using ISA hierarchies, relation transitivity, and inference. For example, for "List employees from Canada", we can identify *John Smith* who works in *Vancouver*, due to a triple stating that *Vancouver* is located in *Canada*.

9.2 Table Querying

Recognizing and parsing tabular data is a related but distinct task from a regular NLP. The location of each table and its individual cells must be accurately detected using visual clues along with the text contained in each cell. Tables can have complex structures with multiple layers of headers at the top as well as on the left-hand side, rotated text, nested tables, inline lists or even hierarchical list, merged cells, etc. Table extraction is a much larger topic, therefore, in this section, we only discuss table representation and its usage for querying.

Tabular text data must also be handled differently from other natural language data as far as syntax is concerned. For tables, the relationships between entities cannot be parsed from natural language syntax, but must instead be discovered from the positions of the entities within the rows and columns of a table. We simplify complex table layouts by explicitly repeating merged cells, so that the overall table structure can be represented as a matrix. We also store which top rows and left-hand-side columns are headers, as shown in Figure 8.

Such relationships between cells are represented using a table-specific RDF output, where the table, its caption and all its cells become RDF resources, with links between tables and the source document, tables and their captions, cells and their tables, as well as relations

between cells to capture column/row header information. For instance, the following triples partially represent the structure of the table shown in Figure 8:

```
example:t5
    rdf:type          example:Table;
    example:document example:doc1;
    example:caption  example:c5;
.
example:c5
    rdf:type      example:Caption;
    example:text  ''Table 5. ...'';
    example:table example:t5;
.
example:c-2-0
    rdf:type       example:Cell;
    example:text   ''Bob Johnson'';
.
example:c-2-2
    rdf:type       example:Cell;
    example:text   ''$135,000'';
    example:table  example:t5;
    example:row    example:c-2-0;
    example:column example:c-0-2;
```

The NLP-based RDF representation is then produced for each of the cells and table captions. For instance, "[MONEY $135,000]" for the bottom-right cell of the table (`example:c-2-2 example:neType ''MONEY''`). Additionally, for the table's caption, we have "Table 5. [EMPLOYEE_G Employees] [SALARY_G salary]." as well as HAS_SALARY(*employee*, *salary*).

The joint representation is the basis for inferring domain-specific relations within tables. For example, ISA($135,000, *salary*) and, furthermore, HAS_SALARY(*Bob Johnson*, $135,000), which is equivalent to the representation of the custom relation as identified from the sentence "Bob Johnson earns $135,000". Using domain-specific relations, textual and tabular data are uniformly represented and can be queried seamlessly as described above.

If the domain-specific relations are not inferred, we fall back to structural representation. For instance, to answer a natural-language query, such as "What was Bob Johnson salary in 2019?", we first identify the concepts: "Bob Johnson", "salary" and "2019". Then, we use a SPARQL query to pinpoint them within table headers and captions, and return the values found at the intersections of structural relations. This graph representation approach allows expansion in any dimension and can support complex queries that require joining multiple tables.

10 Conclusion

In this paper, we summarized our experience with various ways of leveraging ontologies throughout the NLP development cycle as well as within NLP applications. The key benefits include (1) capturing SMEs agreement in a human- and machine-readable format and (2) providing domain knowledge that is directly used for generalization and inference, which is hard to achieve with purely data-driven approaches. We use ontologies to yield more control to SMEs and provide a standard format for tool configuration and integration. Using RDF to represent NLP output reaps the benefits of an open standard supported by numerous open-source and commercial-grade tools, thus making the integration of an existing knowledge management platform with NLP tools fairly straightforward.

While the creation of an ontology in the beginning of a project seems a considerable upfront investment, it pays off later by reducing disagreement between SMEs and, as we have shown, it can be leveraged in numerous ways to lower the barrier between SMEs and NLP systems. An ontology is especially beneficial when it can drive a series of applications within the same knowledge domain, as it allows keeping the applications in sync with each other. An additional benefit of ontologies is the opportunity to have multiple layers of representation to streamline the application to similar problems [27]. For instance, an upper ontology can be used for overall domain modeling (e.g., the automotive industry), while manufacturer-specific or dealer-specific information can be expressed on top of the upper ontology as additional files.

Ontologies do not put any restriction on the NLP methodology. The approach described in this article can be implemented with multiple annotation tools, NLP frameworks, or graph stores to ease and speed up the project flow.

References

[1] M. Arguello Casteleiro, G. Demetriou, W. Read, M. J. Fernandez Prieto, N. Maroto, D. Maseda Fernandez, G. Nenadic, J. Klein, J. Keane, and R. Stevens, "Deep learning meets ontologies: experiments to anchor the cardiovascular disease ontology in the biomedical literature," *Journal of Biomedical Semantics*, vol. 9, no. 1, p. 13, Apr 2018.

[2] R. Arp, B. Smith, and A. D. Spear, *Building Ontologies with Basic Formal Ontology*. The MIT Press, 2015.

[3] A. Ayadi, A. Samet, F. de Bertrand [de Beuvron], and C. Zanni-Merk, "Ontology population with deep learning-based nlp: a case study on the biomolecular network ontology," *Procedia Computer Science*, vol. 159, pp. 572 – 581, 2019,

knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES2019.

[4] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The berkeley framenet project," in *Proceedings of the 17th International Conference on Computational Linguistics - Volume 1*, Stroudsburg, PA, USA, 1998, pp. 86–90.

[5] M. Balakrishna, D. Moldovan, M. Tatu, and M. Olteanu, "Semi-automatic domain ontology creation from text resources," in *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta, may 2010.

[6] E. Batbaatar and K. H. Ryu, "Ontology-based healthcare named entity recognition from twitter messages using a recurrent neural network approach," *International journal of environmental research and public health*, vol. 16, no. 19, p. 3628, Sep 2019. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/31569654

[7] S. Batra, S. Sachdeva, and S. Bhalla, "Entity attribute value style modeling approach for archetype based data," *Information*, vol. 9, p. 2, 2017.

[8] W. Behrendt, *The Interactive Knowledge Stack (IKS): A Vision for the Future of CMS*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 75–90.

[9] M. Bennett, "The financial industry business ontology: Best practice for big data," *Journal of Banking Regulation*, vol. 14, no. 3-4, pp. 255–268, July 2013.

[10] E. Blanco and D. Moldovan, "Composition of semantic relations: Theoretical framework and case study," *ACM Trans. Speech Lang. Process.*, vol. 10, no. 4, Jan. 2014.

[11] O. Bodenreider, "The unified medical language system (umls): Integrating biomedical terminology," 2004.

[12] D. Brickley and R. Guha, "RDF Vocabulary Description Language 1.0: RDF Schema," World Wide Web Consortium, W3C Recommendation, 2004. [Online]. Available: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/

[13] P. P. Chen, "The entity-relationship model - toward a unified view of data." *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9–36, 1976.

[14] C. Chiarcos and C. Fäth, "CoNLL-RDF: Linked Corpora Done in an NLP-Friendly Way," in *Language, Data, and Knowledge*, J. Gracia, F. Bond, J. P. McCrae, P. Buitelaar, C. Chiarcos, and S. Hellmann, Eds., Cham, 2017, pp. 74–88.

[15] Y. T. Demey, "Adapting the fact-based modeling approach in requirement engineering," in *Proceedings of the Confederated International Workshops on On the Move to Meaningful Internet Systems: OTM 2014 Workshops - Volume 8842*. New York, NY, USA: Springer-Verlag New York, Inc., 2014, pp. 65–69.

[16] T. Erekhinskaya, D. Strebkov, S. Patel, M. Balakrishna, M. Tatu, and D. Moldovan, "Ten ways of leveraging ontologies for natural language processing and its enterprise applications," in *The International Workshop on Semantic Big Data*, 2020.

[17] D. Ferrucci, A. Lally, K. Verspoor, and E. Nyberg, "Unstructured information management architecture (UIMA) version 1.0," OASIS Standard, 2009. [Online]. Available: https://docs.oasis-open.org/uima/v1.0/uima-v1.0.html

[18] S. Hellmann, J. Lehmann, S. Auer, and M. Brümmer, "Integrating nlp using linked data," in *The Semantic Web – ISWC 2013*, H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. X. Parreira, L. Aroyo, N. Noy, C. Welty, and K. Janowicz, Eds., 2013, pp. 98–113.

[19] T. Hills, *NoSQL and SQL Data Modeling: Bringing Together Data, Semantics, and Software*, 1st ed., ser. 10. Technics Publications, 2016, vol. 4.

[20] P. Hohenecker and T. Lukasiewicz, "Ontology reasoning with deep neural networks," *CoRR*, vol. abs/1808.07980, 2018. [Online]. Available: http://arxiv.org/abs/1808.07980

[21] J.-C. Klie, M. Bugert, B. Boullosa, R. E. de Castilho, and I. Gurevych, "The inception platform: Machine-assisted and knowledge-oriented interactive annotation," in *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, June 2018, pp. 5–9.

[22] J. L. Martinez-Rodriguez, A. Hogan, and I. Lopez-Arevalo, "Information extraction meets the semantic web: A survey," *Semantic Web*, pp. 1–81, 10 2018.

[23] M. A. Magumba, P. Nabende, and E. Mwebaze, "Ontology boosted deep learning for disease name extraction from twitter messages," *Journal of Big Data*, vol. 5, no. 1, p. 31, Sep 2018.

[24] D. Moldovan and E. Blanco, "Polaris: Lymba's semantic parser," in *Proceedings of the Eighth*

International Conference on Language Resources and Evaluation, Istanbul, Turkey, May 2012, pp. 66–72.

[25] R. Morante and C. Sporleder, Eds., *Proceedings of the Workshop on Negation and Speculation in Natural Language Processing*. Uppsala, Sweden: University of Antwerp, Jul. 2010. [Online]. Available: https://www.aclweb.org/anthology/W10-3100

[26] V. M. Ngo and T. H. Cao, "Ontology-based query expansion with latently related named entities for semantic text search," *CoRR*, vol. abs/1807.05579, 2018. [Online]. Available: http://arxiv.org/abs/1807.05579

[27] I. Niles and A. Pease, "Towards a standard upper ontology," in *Proceedings of the International Conference on Formal Ontology in Information Systems - Volume 2001*, ser. FOIS '01. New York, NY, USA: ACM, 2001, pp. 2–9.

[28] N. F. Noy and D. L. Mcguinness, "Ontology development 101: A guide to creating your first ontology," Stanford Knowledge Systems Laboratory, Technical Report KSL-01-05, March 2001.

[29] G. Petrucci, C. Ghidini, and M. Rospocher, "Ontology learning in the deep," in *20th International Conference on Knowledge Engineering and Knowledge Management - Volume 10024*, ser. EKAW 2016, Berlin, Heidelberg, 2016, p. 480–495.

[30] G. C. Publio, D. Esteves, A. Lawrynowicz, P. Panov, L. N. Soldatova, T. Soru, J. Vanschoren, and H. Zafar, "Ml-schema: Exposing the semantics of machine learning with schemas and ontologies," *CoRR*, vol. abs/1807.05351, 2018. [Online]. Available: http://arxiv.org/abs/1807.05351

[31] R. Rak and S. Ananiadou, "Making UIMA truly interoperable with SPARQL," in *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*. Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 89–97. [Online]. Available: https://www.aclweb.org/anthology/W13-2311

[32] D. L. Sackett, W. S. Richardson, W. Rosenberg, and R. B. Haynes, "How to practice and teach evidence-based medicine," *New York: Churchill Livingstone*, pp. 118–128, 1997.

[33] SNOMED International, "SNOMED CT: 5-step briefing," 2019. [Online]. Available: http://www.snomed.org/snomed-ct/five-step-briefing

[34] P. Stenetorp, S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou, and J. Tsujii, "Brat: A web-based tool for nlp-assisted text annotation," in *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, ser. EACL '12, Stroudsburg, PA, USA, 2012, pp. 102–107.

[35] V. Tarasov, U. Seigerroth, and K. Sandkuhl, "Ontology development strategies in industrial contexts," in *BIS*, 2018.

[36] M. Tatu, M. Balakrishna, S. Werner, T. N. Erekhinskaya, and D. I. Moldovan, "A semantic question answering framework for large data sets," *Open J. Semantic Web*, vol. 3, no. 1, pp. 16–31, 2016. [Online]. Available: https://nbn-resolving.org/urn:nbn:de:101:1-201705194921

[37] M. Tatu, M. Balakrishna, S. Werner, T. N. Erekhinskaya, and D. I. Moldovan, "A semantic question answering framework for large data sets," *OJSW*, vol. 3, no. 1, pp. 16–31, 2016. [Online]. Available: https://nbn-resolving.org/urn:nbn:de:101:1-201705194921

[38] R. Troncy and M. Bruemmer, "Nerd meets nif: Lifting nlp extraction results to the linked data cloud," in *In Proceedings of the 5th International Workshop on Linked Data on the Web*, 2012.

[39] M. Uschold and M. Gruninger, "Ontologies: principles, methods and applications," *The Knowledge Engineering Review*, vol. 11, no. 2, p. 93–136, 1996.

[40] R. Witte and B. Sateli, "The LODeXporter: Flexible generation of linked open data triples from NLP frameworks for automatic knowledge base construction," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation)*, Miyazaki, Japan, May 2018.

[41] P. Wongthongtham, N. Kasisopha, E. Chang, and T. Dillon, "A software engineering ontology as software engineering knowledge representation," in *2008 Third International Conference on Convergence and Hybrid Information Technology*, vol. 2, Nov 2008, pp. 668–675.

[42] H. Ziad, J. P. McCrae, and P. Buitelaar, "Teanga: A linked data based platform for natural language processing," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*, Miyazaki, Japan, May 2018.

AUTHOR BIOGRAPHIES

Tatiana Erekhinskaya studied Mathematics at the Nizhniy Novgorod State University, Russia. She worked at Dictum Ltd focusing on natural language processing of the Russian language. The key projects include syntactic parser robust to spelling mistakes and opinion mining using dependency parsing. She received her MBA degree in 2010. She received her PhD in Computer Science from The University of Texas at Dallas in 2014. Her dissertation focused on probabilistic models for text understanding. Today, she is a research scientist at Lymba Corporation. Her primary areas of research are: deep semantic processing with special focus in medical domain and big data.

Marta Tatu received her PhD in Computer Science from The University of Texas at Dallas in 2007. As a research scientist at Lymba Corporation, she focuses on applying knowledge extraction and question answering technologies to various unstructured data sources such as contracts, scientific proposals, and research publications. She has been instrumental in the development of Lymba's state-of-the-art question answering system, PowerAnswer, which topped NIST TREC QA evaluations for seven years. As the PI of an NSF award, she has focused on developing a solution to process large amounts of unstructured data. By aligning the extracted knowledge against a common ontology, the solution provides customized semantic search to intelligence analysts.

Mithun Balakrishna received his PhD in Computer Science from The University of Texas at Dallas in 2007. His academic research focused on extraction and application of high-level linguistic knowledge to improve spoken language recognition and understanding. He currently leads Lymba Corporation's research and business thrusts in the area of Knowledge Engineering and Management. His research is focused on the development of tools to automatically build semantically rich knowledge models for specific domains using relevant unstructured data. He has been the PI on several enterprise and government projects, including Spoken Dialog Question Answering, Automatic Extraction of Biographical Profiles, Ontologies for Education, and Antibiotic Resistance Knowledge Extraction.

Sujal Patel received his Master's degree in Computer Science in 2017 from The University of Texas at Dallas. He joined Lymba Corporation in February 2018. He has focused his efforts on classification projects for retail inventory wrangling, as well as natural-language querying projects for multiple domains, including financial, retail, and legal. He has expertise in natural-language querying. His current research interests include customized natural-language search using ontologies and contextualized word embeddings.

Dmitry Strebkov received his Master's degree in Applied Informatics from the Nizhniy Novgorod State University, Russia. While working at Dictum Ltd., he focused on development of the full NLP pipeline for opinion mining. The solution included rule-based components, such as synactic parsing for Russian and Arabic languages, as well as data-driven components such as Arabic diacritization system. With Lymba Corporation, he mostly worked on linked data processing and usage of ontologies for NLP customization and inference. His main areas of interest are lexical semantics and open domain QA.

Dan Moldovan received his diploma degree in Engineering in 1969 from the Polytechnic Institute of Bucharest, Romania. He received his PhD in Computer Science in 1978 from Columbia University, New York. He is a Professor of Computer Science at the University of Texas at Dallas and the co-Director of the Human Language Technology Research Institute at UTD. Previously he held faculty positions at the University of Southern California and Southern Methodist University in Dallas. He was a Program Director at NSF while in sabbatical from USC. He is the Founder and Chief Scientist of Lymba Corporation, a Texas based company specializing in Natural Language Processing products and solutions. His current research interests are in lexical semantics, in particular studying the representation and reasoning of explicit and implicit text knowledge, and transforming unstructured data into semantic triples.