Improving Semantic Queries by Utilizing UNL Ontology and a Graph Database

Petri Kivikangas, Mitsuru Ishizuka
Department of Creative Informatics
Graduate School of Information Science and Technology, The University of Tokyo
Tokyo, Japan
pkivik@gmail.com, ishizuka@i.u-tokyo.ac.jp

Abstract— **This paper describes an approach for improving semantic queries by utilizing Universal Words (UWs) and a graph database. Concept Description Language (CDL) is used for representing the semantic data, and Neo4j graph database is used as the storage back-end. Cypher graph query language is used as the basis for implementing the semantic queries. For improving the queries, query expansion is performed by utilizing semantic relationships between UWs provided by UNL Ontology.**

Keywords- semantic computing; semantic web; semantic search; graph database; inexact graph matching; CDL; CWL; UNL; Neo4j; Cypher

I. Introduction

Currently the most popular semantic representation format in the Semantic Web community is RDF. This paper presents a more general-level semantic representation format called Concept Description Language (CDL). CDL provides a flexible way for representing natural language text in an unambiguous computer understandable form by utilizing Universal Words (UWs) and Universal Relations. [1][2]

UNL Ontology provides semantic relations between Universal Words (UWs). UWs are disambiguated natural language words, each UW representing a single natural language concept. [3]

Previous research on semantic search includes e.g. iSeS search framework [4], FREyA natural language interface [5] and Treo natural language query mechanism [6]. However, the presented approach differs from previous approaches by utilizing UWs, which are unambiguated natural language words. Because of unambiguity, the search engine does not have to guess the correct senses of the words in the query.

II. Concept Description Language

Concept Description Language (CDL) is a declarative formal language for representing semantic data. CDL is a machine-understandable language aiming at becoming the next generation language for the intelligent Web.

A. Syntax

Basic units in CDL are concepts, arcs and statements. Statements can contain concepts and other statements. Statements and concepts both have an identifier unique in the current scope, called Realization Label (RL). A concept consists of a UW and optional attributes, which are used to represent various grammatical categories, such as mood, modality, number and tense.

The terminology used in this paper differs slightly from the official specification. Concept corresponds to elemental entity, and statement corresponds to complex (or composite) entity. Complex entity is an entity with a structure, which means that it can contain other entities. Hence, a statement can contain other statements and concepts whereas a concept cannot.

Figure 1 demonstrates how a simple natural language sentence could be converted into CDL. All concepts and the inner sentence are identified by an identifier that is unique in current scope delimited by curly brackets. In a CDL document, sentences are enclosed by curly brackets, concepts by angle brackets and arcs by square brackets. E.g. *[A1 agt A0]* means that the agent of the entity identified by *A1* (in this case concept *buy(agt>person,obj>thing).@past*) is an entity identified by *A0* (in this case the concept *Jim*). *agt* is one of the universal relations, indicating a thing in focus that initiates an action.

It should be noted that it is possible to refer also outside the current scope, as arc *[A4 agt A0]* does. However, this should be avoided unless it can be made sure that all RLs in a document are unique, because uniqueness of RLs is not guaranteed by the specification.

```
{#
        <A0:Jim>
        <A1:buy(agt>person,obj>thing).@past>
        <A2:car(icl>vehicle)>
        {#A3
                <A4:get(agt>person,obj>thing).@past>
                <A5:salary(icl>money)>
                [A4 agt A0]
                [A4 obj A5]
        }
        [A1 agt A0]
        [A1 obj A2]
        [A1 seq A3]
}
```

Figure 1. CDL data example: “Jim bought a car after he got salary”

B. *Comparison to RDF*

CDL was designed for the following purposes:

1. Intermediate language between natural languages, formal languages and visual media
2. Enable deep semantic processing, in addition to shallow conceptual level processing
3. Intermediate language between syntactic document structure processing and semantic document structure processing

One main purpose of CDL is to function as an intermediary language (or *pivot* language) between natural languages. CDL was also designed for describing context structure, whereas RDF was designed for describing resources found in the Web. Due to their different goals, CDL and RDF are not direct competitors, but can actually benefit from each other. [1][2]

III. NEO4J

Neo4j is a graph database allowing storing data as nodes connected by arcs. Because semantic data is easily represented as graphs, provides graph database more natural abstraction for such data than relational database. A schema-less graph database also allows to easily add new types of data and relations.

A. *Cypher graph query language*

Neo4j provides a query language called Cypher[1], which is a declarative general purpose graph query language. Cypher has been influenced by SQL and SPARQL, and it allows expressive and efficient querying of graph databases without having to write detailed graph traversals. The main difference between Cypher and SPARQL is that SPARQL is designed for the domain of Semantic Web, whereas Cypher is a general purpose graph query language. The development of Cypher started out from a need to have easier syntax than existing general purpose graph query languages, such as Gremlin[2].

```
START  x=node:indexName(idxPropName='propValue')
MATCH         x-[:relationName]->y
WHERE         (x.property_1 = 'value1'  OR
              x.property_1 = 'value2') AND
              y.property_2 = 'value3'
RETURN        x, y
```

Figure 2. Example Cypher query

Figure 2 presents an example Cypher query. The query selects a starting point by doing index lookup from index *indexName*, fetching nodes that have property *idxPropName* with value *propValue*. Then the query goes through the fetched nodes trying to match the relationship and the rest of the criteria. Finally, the query returns all the nodes *x* and *y* that match the pattern and satisfy the criteria.

B. *Apache Lucene indexing*

Neo4j does not provide own indexing solution, and instead as a default uses Apache Lucene[3] search and indexing library. In Neo4j, Lucene provides means for indexing nodes and relationships. Due to Lucene indexes, finding Neo4j nodes by node properties is extremely fast.

In test setup, the queries were executed in around 10 milliseconds on average. However, if the indexes were not utilized, the queries ran more than one magnitude slower. Thus, to gain maximum query performance, Lucene indexes should be fully utilized.

C. *Interfaces and operation modes*

Neo4j supports three operating modes: *embedded*, *server* and *embedded server*. In embedded mode, the database can only be accessed by the application into which it is embedded. In server mode the database can be accessed anywhere through the REST API. In embedded server mode the application has direct access to the database, and remote parties have access through REST API. In the experiment setup, embedded server mode was chosen because it provides fast database access for the application, and possibility to access the database through a web interface.

IV. CDL TOOLKIT

CDL Toolkit was developed to present the chosen approach in practice, and to facilitate future research efforts on CDL. The toolkit contains a query builder, a parser, Neo4j integration and a search engine.

A. *Neo4j integration*

At this stage, the Neo4j integration allows to insert data and execute queries. However, it is also possible to access and modify the data using the web interface provided by Neo4j. The toolkit uses Neo4j in embedded server mode, which means that the application is able to access the database directly, whereas remote users have access through the REST[4] API.

B. *Search engine*

The toolkit allows one to write queries with a slightly modified CDL syntax. E.g. natural language question "What did Jim buy?" can be represented as shown in Figure 3. Based on data shown in Figure 1, the search engine would be able to answer the query by returning *car(icl>vehicle)*.

```
{#
        <A0:Jim>
        <A1:buy(agt>person,obj>thing).@past>
        <A2:?x>
        [A1 agt A0]
        [A1 obj A2]
}
```

Figure 3. Example CDL query

[1] http://docs.neo4j.org/chunked/stable/cypher-query-lang.html
[2] https://github.com/tinkerpop/gremlin/wiki
[3] http://lucene.apache.org/core/
[4] http://en.wikipedia.org/wiki/Representational_state_transfer

Because Neo4j cannot comprehend CDL, the query must be converted into Cypher before it can be executed on the database. The conversion is performed automatically by the search engine. Figure 4 shows how the CDL query is converted into Cypher.

```
START       xA0=node:concepts(uw='Jim')
MATCH       xA1-[:agt]->xA0,
            xA1-[:obj]->xA2
WHERE       xA1.uw! = 'buy(agt>person,obj>thing)'
RETURN      xA2
```

Figure 4. CDL converted to Cypher

V. SEMANTIC DATA IN A GRAPH DATABASE

This section describes how CDL data can be represented as a semantic graph in Neo4j.

A. Representing CDL data and UNL Ontology as graphs

Figure 5 illustrates the structure of CDL data inside the database. In the experiment setup, all documents are directly connected to the root node. UNL ontologies are connected to the root node through *uw* node, which is the root node of the ontology graph. Each document can contain multiple statements, and each statement can contain concepts or other statements. Concepts are connected to the corresponding ontology concepts. The attributes of a concept are stored as a string array. The 325 hidden nodes shown in the figure represent the sentences directly below the document.

Due to CDL's nature as a conceptual graph, adding CDL documents into a graph database is an easy process. Figure 6 illustrates how the ontology can be stored in Neo4j. Neo4j does not require the graph to be connected, but in the experiment, all nodes were connected. Nodes can be accessed either by their id or through indexes. The 38 hidden nodes under the root node are the CDL documents stored in the database at the time. CONTAINS relation is not a semantic relation, but it is used only to denote the sub-entities.

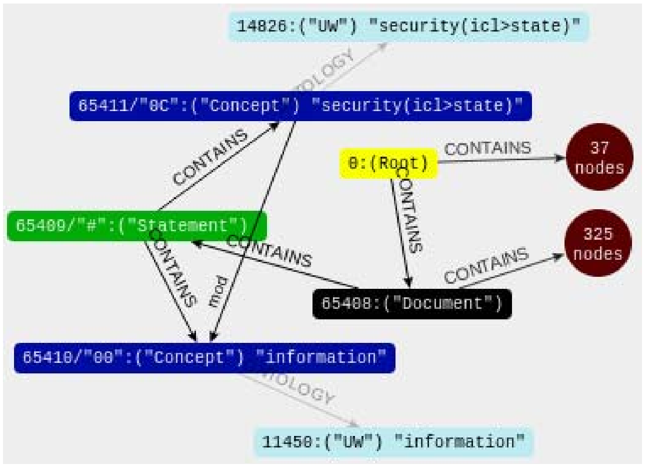
14826:("UW") "security(icl>state)"
ONTOLOGY
65411/"0C":("Concept") "security(icl>state)"
CONTAINS
0:(Root)
CONTAINS
37 nodes
65409/"#":("Statement")
CONTAINS
CONTAINS
CONTAINS
325 nodes
CONTAINS
mod
65408:("Document")
65410/"00":("Concept") "information"
ONTOLOGY
11450:("UW") "information"

Figure 5. CDL data stored in Neo4j (Neo4j web interface)

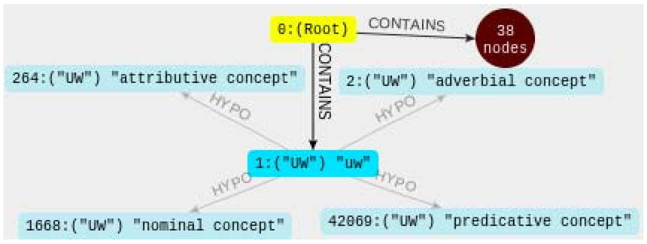
0:(Root)
CONTAINS
38 nodes
264:("UW") "attributive concept"
CONTAINS
2:("UW") "adverbial concept"
HYPO
HYPO
1:("UW") "uw"
HYPO
HYPO
1668:("UW") "nominal concept"
42069:("UW") "predicative concept"

Figure 6. Top level UWs of UNL Ontology stored in Neo4j (Neo4j web interface)

B. Querying

There are multiple ways for querying Neo4j but CDL Toolkit relies on Cypher and its advanced features. Figure 7 gives an example on how Cypher can be used for returning all nodes on a specific path inside the ontology.

```
START       x=node:uws(uw='uw')
MATCH       p1=x-[HYPO*0..20]->y
RETURN      y, length(p1)
```

Figure 7. Advanced queries in Cypher

The query does a lookup on index *uws*, fetching all nodes that are indexed by property *uw* with value *uw*. In *MATCH* part the path is put into optional variable *p1*. *x-[HYPO*0..20]->y* matches all nodes starting from *uw* that have 0 to 20 *HYPO* relations between them. It is also possible to access data directly through indexes, assuming that the nodes have been added to an index, as Neo4j does not index nodes automatically unless configured to do so.

C. Performance

There are several things affecting the performance of querying. One should try to minimize the size of node set used in the pattern matching. This is possible by selecting the concept that has the lowest document frequency as a starting point. UNL System contains document frequencies for some concepts, but not to the extent that they would provide significant benefit for index lookup optimization.

The query expansion (see VI.B) should also be set to reasonable limits. For high ontology level concepts, the hyponym tree used for the query expansion can contain hundreds or even thousands of UWs. One way is to allow the user to select the level of expansion. If the user does not find answer on the first try, he can select more aggressive expansion, which naturally means also slower quer

VI. QUERY IMPROVEMENT

This section describes the approach chosen for improving semantic queries. The query improvement consists of fetching all the concepts that have meaningful relation to the concepts in the query, and then augmenting the original query with the fetched concepts.

A. UNL Ontology

The search improvement technique is based on utilizing semantic relations in UNL Ontology. The ontology contains only the hierarchical relations, represented as *icl* (is a kind of) and *iof* (is an instance of). *icl* is equivalent to hypernymy, e.g. a UW *mammal(icl>animal)* implies that there is an *icl* relation from mammal to animal. UNL Ontology is available in a simple tree structure[5]. [3]

In CDL Toolkit, hyponymy is implemented through *HYPO* relation. The *HYPO* relation is equivalent to reverse *icl* relation. E.g., if the ontology contains a UW *mammal(icl>animal)*, it would mean that there exists a *HYPO* relation from *animal* to *mammal(icl>animal)*. Because Neo4j relations can be traversed in both ways, there is no need for another relation for modeling hypernymy.

B. Query expansion

The query expansion is performed in three steps:

1. Find the ontology concepts for the data concepts
2. Fetch all data concepts that are meaningfully related to the ontology concepts
3. Add the hyponyms to the original Cypher query

Ontology concept is a UW stored in the ontology graph. Data concepts are the concepts in the CDL documents. The first and second steps can be performed in one Cypher query. In the third step, the query is augmented by all found hyponyms.

Figure 8 shows a modified version of the query shown in Figure 3. The query in Figure 3 can be directly answered from the data shown in Figure 1. However, the query in Figure 8 uses verb *get* instead of *buy*. Verb *get* does not show in the data, and hence the search engine cannot directly answer the query. The only solution to get an accurate answer is to find a suitable semantic relation between *get* and *buy*. In this case, entailment would provide the suitable relation connecting these two concepts.

```
{#
        <A0:Jim>
        <A1:get(icl>obtain(agt>person,obj>thing)).@past>
        <A2:?x>
        [A1 agt A0]
        [A1 obj A2]
}

START     xA0=node:concepts(uw='Jim')
MATCH     xA1-[:agt]->xA0, xA1-[:obj]->xA2
WHERE     (xA1.uw = 'get(icl>obtain(agt>person,obj>thing))')
          OR (xA1.uw = 'buy(agt>person,obj>thing)')
RETURN    xA2
```

Figure 8. Example of indirect CDL query and corresponding Cypher query when expanded

[5] http://www.undl.org/unlsys/uw/UNLOntology.html

VII. Conclusions and Future Work

This paper described a technique of utilizing a graph database for storing semantic data represented in CDL, and performing query improvement for semantic queries. The query improvement was performed by utilizing semantic relationships between the UWs in UNL Ontology.

The main problem was the small size of the current publicly available version of UNL Ontology. As less than 20% of the concepts in the sample documents had a representation in the ontology, it was not possible to perform query improvement in the extent that was hoped for.

To further improve the queries, more complex semantic relations, such as entailment, holo-, mero- and antonymy, should be provided. The existence of these relations in the ontology would dramatically increase the reasoning capability of the search engine. Finally, to enable natural language searches, a parser capable of parsing and converting natural language questions into CDL queries should be developed.

Acknowledgment

Special thanks to Mamdouh Farouk and Francisco Tacoa, who helped by providing sample data and useful information about the core semantic technologies employed in the experiment.

References

[1] T. Yokoi, H. Yasuhara, H. Uchida, M. Zhu, K. Hasida, "CDL (Concept Description Language): A Common Language for Semantic Computing," WWW2005 Workshop on the Semantic Computing Initiative (SeC2005), Makuhari, Japan, 2005, http://www.instsec.org/tr/CDL.pdf

[2] H. Uchida, T. Yokoi, M. Zhu, N. Saito, V. Avetisyan, "Common Web Language", W3C Incubator Group Report, Mar. 2008, http://www.w3.org/2005/Incubator/cwl/XGR-cwl/

[3] UNL Center of UNDL Foundation, "Universal Networking Language (UNL) Specifications Version 2005 Edition 2006", Aug. 2006, http://www.undl.org/unlsys/unl/unl2005-e2006/

[4] M. Jayaratne, I. Haththotuwa, C.D. Arachchi, S. Perera, D. Fernando, S. Weerakoon, 2012, iSeS: intelligent semantic search framework, EATIS '12 , ACM, pp. 215-222

[5] D. Damljanovic, M. Agatonovic, H. Cunningham, "FREyA: an Interactive Way of Querying Linked Data using Natural Language", 'Proceedings of 1st Workshop on Question Answering over Linked Data (QALD-1), Jun. 2011

[6] A. Freitas, J.G. Oliveira, E. Curry, S. O'Riain, J.C.P. da Silva, "Treo: Combining Entity-Search, Spreading Activation and Semantic Relatedness for Querying Linked Data", Proceedings of 1st Workshop on Question Answering over Linked Data (QALD-1), Jun. 2011