2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)

From natural language to graph queries

Gaétan J. D. R. Hains*, Youry Khmelevsky‡, Thibaut Tachon*†
*CSI, Paris Research Lab, Huawei Technologies
Boulogne Billancourt, France
Email: {thibaut.tachon,gaetan.hains}@huawei.com
†Univ. Orléans, INSA Centre, Val de Loire, LIFO EA 4022, Orléans, France
‡Computer Science Department
Okanagan College, Kelowna (BC) Canada
Email: ykhmelevsky@okanagan.bc.ca

***Abstract*—Automatic code generation can drastically improve software (SW) engineering and SW development projects. In the last decade we have been conducting research which has been advancing the field of code generators for small and mid-size Web-based DBMS systems [4], [5], [7]. We developed a number of tool prototypes for automatic source code debugging by the source-to-source code transformation for real C and C++ applications [8]. Additionally we investigated Natural Language Processing (NLP) for software code generation and application of it to Graph databases. Graph databases are becoming more and more popular for their applications in Artificial Intelligence (AI) systems, social analytics and many other fields. Query languages like Cypher allow users to search them without direct programming. But even queries of modest complexity like "relatives in a family & friends graph" require some skill to write. In this paper we describe the use of natural language as a more intuitive interface for untrained users and demonstrate 3 use-cases, where translation of typical English phrases to OpenCypher and/or specialized graph engines like Huawei EYWA.**

Keywords – graph databases, natural-language input, queries, Cypher query language, Neo4j, grammar.

I. Introduction

In the last decade our research team created a new approach for Model Driven Software Engineering and code generation. This approach was translated into the development of a prototype to demonstrate a proof of concept [8]. We with the University Paris-Est (France) achieved performance optimization by adaptive code generation from customized UML models. The novelty of the prototype was that it can be modified by end-users on the UML model level and then used with automatic Java code generation [3]–[7]. Furthermore, an advanced coding environment was developed that provides a visual and declarative approach to trading algorithms development. This may directly generate a portable bitcode on the Low-Level Virtual Machine (LLVM) from financial specifications of trading strategies.

In 1995 I. Androutsopoulos (Dept.of Artificial Intelligence, Univ.of Edinburgh) et al. in [1] suggested natural language interfaces to databases (NLIDBs). They discussed advantages and disadvantages of NLIDBs, comparing NLIDBs to formal query languages, form-based interfaces, and graphical interfaces. They described some of the linguistic problems NLIDBs have to confront and suggested NLIDB architectures, discussed portability issues, restricted natural language input systems (including menu-based NLIDBs), and NLIDBs with reasoning capabilities.

In 2013 P.A. Dhomne et al. in [2] proposed an approach for accessing the database easily without knowing English. Their system was capable of handling simple queries with standard join conditions. Because not all forms of SQL queries are supported, further development would be required before the system can be used within NLIDB. Alternatives for integrating a database NLP component into the NLIDB were considered and assessed. The system translates English language to SQL.

In [10] authors discuss how to formulate a query in such way that the computer will understand and produce the desired output from relational databases to retrieve information from a database, one needs. They believe that few people who have knowledge of database structure and formal database language (such as Structured Query Language (SQL) can retrieve the desired information from database. In their paper they suggested to improve human computer interface to allow people interacting with the database in their natural language using Natural Language Processing (NLP). They used Intermediate Representation Technique which is a combination of syntactic based system and semantic based system.

Authors in [9] in 2008 already tried to use NLP for none professional users transfers their questions and requirements to computer in natural language and derives his desired data by natural language processing. They suggested a method for building a "natural languages interfaces to data bases" (NLIDB) system. In their work it first parses the input sentences, and then the natural language expressions are transformed to SQL language.

II. Project Goals and System Design

In 2017 and 2018 three student teams in SW Engineering capstone project courses at Okanagan College (OC) and at University of British Columbia (UBCO, Okanagan Campus) in Canada investigated how to employ NLP technic to generate queries to Neo4J database systems.

The goals of the projects were the following:

- Train Computer Science students by real international research projects by collaborating with the Huawei Research Centre in Paris
- Solve a real problem: NLP-to-SQL was done before but not NLP-to-Cypher.
- Produce a prototype with clean design and some limited but realistic NLP i.e. many input sentences that look natural for the same Cypher query. Processing steps of the prototype produced are detailed in the activity diagram Fig 1.

978-1-7281-0319-8/19/$31.00 ©2019 IEEE

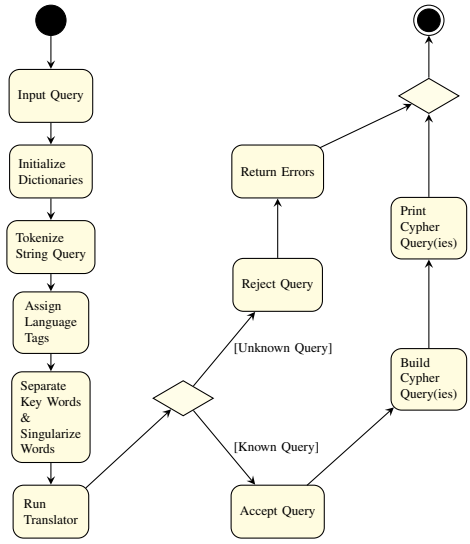

Fig. 1. Activity Diagram. Shows program processing steps.

A. Sampling

In order to match words of the input natural language sentence to properties, node labels and edge types of the graph, sampling the graph is unavoidable. Thus, as preprocessing, the graph is sampled with the following four cypher commands.

1) Find nodes labels :

```
MATCH (n)
UNWIND labels(n) AS lab
RETURN DISTINCT lab
```

2) Find nodes properties :

```
MATCH (n)
UNWIND keys(n) AS key
RETURN DISTINCT key
```

3) Find edges types :

```
MATCH ()-[r]-()
RETURN DISTINCT type(r)
```

4) Find edges properties :

```
MATCH ()-[r]-()
UNWIND keys(n) AS key
RETURN DISTINCT key
```

B. Natural language queries

Number of queries were tested with variation in both semantic and form. A few are written in this section to highlight some of the difficulties encountered. The translation of those queries is provided in section II-C.

a) Case oblivious: one of the property which comes first. Thus, the following queries ought to result in the same output (cypher query number II-C.1).

1) what are the names of all the people?
2) What are the names of all the people?
3) WHAT ARE THE NAMES OF ALL THE PEOPLE?

b) Stop words: the words whose meaning is not relevant for NLP as "of", "the" and "a" in the second natural language query below.

1) How many names start with J ?
2) How many of the names start with a J

Both above queries outputs the cypher query II-C.5.

c) Synonyms and paraphrase: to be treated as such. The following natural language queries should all output the cypher query II-C.2.

1) List the names of the outlaws.
2) List each outlaws.
3) Give me a list of every outlaws.
4) Return a list of the outlaw's names.
5) Who are all the outlaws ?
6) What's the name of all the outlaws ?

The query may be interrogative or imperative.

C. Cypher queries

This section shows a number of cypher features through queries. Students were asked to include as many features as possible in their translation output. This allowed them to progress iteratively. Features 1 to 3 were successfully outputted by all groups. Features 4 to 7 were successfully outputted by at least one group. Features 8 and above were not outputted by any group.

1) Node matching filtered on a label

```
MATCH (n :Person)
RETURN n.name
```

2) Node matching filtered on several labels.

```
MATCH (n :Person :Outlaw)
RETURN n.name
```

When several node labels are mentioned in pattern matching, it's a conjunction : the node matched must possess all labels. Thus the result is a strict subset of query 1.

3) Node matching filtered on properties

```
MATCH (n {species : 'dog'})
RETURN n
```

An equivalent query could be
`MATCH (n)WHERE n.species ='dog'RETURN n`

4) Filtering by quantification on properties

```
MATCH (n)
WHERE n.bounty > 3000
RETURN COUNT (n.name)
```

Operators include (<, >, <>, =, >=, <=).

5) String matching filtering

```
MATCH (n)
WHERE n.name STARTS WITH "J"
RETURN COUNT (n.name)
```

For constraining strings, **STARTS WITH**, **ENDS WITH** and **CONTAINS** are possible.

6) Filtering by property existence checking and aggregation of results

```
MATCH (n)
WHERE n.species IS NOT NULL
RETURN COUNT (n.name)
```

To know whether a node has a property, **IS NULL**, **IS NOT NULL** may be used.

7) Relationship matching with edge types filtering

```
MATCH () -[:PARENTS] -> (n)
RETURN n.name ORDER BY (n.size)
```

8) Path matching on successive identical edge types

```
MATCH ({name : 'Joe'})
-[:BROTHER *]-> (m)
RETURN DISTINCT m.name
```

The star means an undefined number of hop with relationships matched (here, all of them must have type BROTHER).

9) Sub query

```
MATCH ({name : 'Ma Dalton'})-
[:PARENTS]-> (child)
WITH child // the tallest
ORDER BY child.size DESC LIMIT 1
MATCH (child) -[:LIKES]-> (res)
RETURN res
```

In natural language, the translation of this query could be "What does the tallest child of Ma Dalton likes ?" **WITH** is used when the graph exploration depends on an aggregation result.

III. System Development

Student teams developed the following:

- split input to a sequence of words.
- matched it with patterns of their own design, several patterns for each type of query.
- extracted the labels, variables from that then generate Cypher.
- run the query with Neo4j. The whole things was embedded in a client-server system.

All steps are shown in the activity diagram, Fig. 1.

Additionally a data generator was developed. This is a small Java application used to generate data to test the web API and to populate our database. It generates all of the fields that the WTFast's client would supply in the real system, but is configurable so we can test with large or small volumes of data from many virtual users.

In COSC 470/471 SW Engineering project course at OC students used Python and Flask framework, Neo4J database, Javascript, HTML, and CSS for the frontend. In the project SW development cycle Jira, Slack for team communication, GIT SCM managed by GitLab server (integrated with Jira), and Docker were used. The prototyping and code refactoring were used to prove the concept. For the custom query building they used Query object dictionary, fail-early edge creating and filter predicates on objects.

The following NLP solutions were achieved:

- Extracting information that is useful for generating database query language.
- The cypher query API was developed.
- Token to Cypher glue code was developed.
- More advanced systems can tag words with linguistic data (e.g. is this word a noun, verb, or adjective?).
- Using data directly in the database being queried.
- Implemented Auto complete functionality for English query.

In the two research project prototypes at UBCO English Queries in NL2CQ prototype English sentences in section II-B parsed successfully and the following features were implemented:

- NL2CQ accurately translated simple English language queries, into complicated Cyper graph database queries. The resulting query can be entered into a Neo4j database, and the logical results for the query will be returned.
- Over 50 unit tests were implemented to test accuracy of the resulting cyper queries.
- The ability to compare the similarities between words was implemented. It helped avoiding superfluous words interfering with the input, and increasing the accuracy of the queries.
- Since the students were *not* using a parser generator like yacc, they applied ad hoc text analysis to approximate a context free grammar. For example the words "has", "have", and "with" were resulting in different language tags depending on the input sentence. Another parsing issue was solved by scanning the input list for definitive key words that only that kind of query would have, and creating a Boolean table for them.

In the another UBCO project prototype (From NLP to Graph Queries team — "NLP2GQ") the Python NLTK, Neo4J and Cypher querying language were used (Fig. **??** and Section II-C). The team developed 56 unit test for just queries, but not all possible queries can be tested. The team implemented all queries given by the client, using a template system (see the Section II-C).

IV. Future Work

Our future research will be related to utilizing the developed three prototypes into the industrial applications and further NLP to DBMS queries code generation (NLP to SQL, NLP to Cypher Queries/Neof4J, NLP to SPARQL, etc.).

The following issues still should be resolved in the developed NL2CQ prototype:

- All possible ways to state the query may have not been accounted for. So someone might present a query in a way we haven't thought of yet, and our methods may not handle it.
  A possible workaround: Users should enter queries in the most straight forward logical way, to get the best results. It's necessary to continue to work with the methods to handle more casual, loos english queries.
- Certain punctuation, especially apostrophes cause the language tagger to incorrectly tag words, and this results in an incorrect result, or error.
  A possible workaround: At the moment certain queries that should have punctuation, do not. This is a relatively easy fix, but due to time constraints it was not fixed in the project.

The NLP2GQ team found the following issues, which should addressed in the next project developments:

- Every possible query cannot be implemented without investigation and clever programming.
- Where the current implementation might work on a certain graph database, it might not be applicable to another.
- Implementation on higher complexity graphs may not adapt well.
- There comes a point where complex queries cannot be written in English, in a structure that is grammatically correct — is it really natural language at that point?

The prototype should also be tested with untrained users. Their results would provide an insightful evaluation of the system.

V. Conclusion

We have designed, implemented and tested a system for transforming written English-language to Cypher graph queries. The general method is simple but systematic pattern-matching to allow for multiple NL forms for each possible type of query. Many practical issues were considered and can be implemented in the future e.g. automatic pre-processing of the graph database schema. The project shows that a useful interface can be built without deep NLP or ML techniques, taking advantage of the expressive power of a query language like Cypher. Future work in this direction will include the automatic generation of parallel graph queries and automatic recognition of more complex query patterns.

Through this project the Okanagan College students leaned how to work with industrial clients and produced a useful tool for NLP processing and automatic code generation. They also leaned about database systems, new APIs and programming languages. Moreover they could apply a serious software engineering methodology, and systematic team work organization.

Acknowledgments

The described research projects were done under supervision from the CSI, Paris Research Lab, Huawei Technologies, Boulogne Billancourt, France by the undergraduate students at the Okanagan College and at UBC Okanagan campus.

We would like to thank the NODGE team for the R&D project "English to Cypher" conducted by the Computer Science undergraduate students in COSC 470 SW Engineering (2017) and in COSC 471 SW Engineering Project courses (2018) at Okanagan College. And we would like to thank the 3rd year of study undergraduate students at UBC Okanagan campus in COSC 310 Software Engineering, who participated in two prototypes development: "From NLP to Graph Queries" by team "Won" and "Natural Language to Cypher Query (NL2CQ)" by another team of five the students.

References

[1] Ion Androutsopoulos, Graeme D Ritchie, and Peter Thanisch. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(1):29–81, 1995.

[2] Pooja A Dhomne, Sheetal R Gajbhiye, Tejaswini S Warambhe, and Vaishali B Bhagat. Accessing database using nlp. *IJRET Int. J. Res. Eng. Technol. eISSN*, pages 2319–1163, 2013.

[3] R. Grmek, Y. Khmelevsky, and D. Syrotovsky. Automated inventory tracking system prototype in cloud. In *High Performance Computing and Simulation (HPCS), 2011 International Conference on High Performance Computing & Simulation*, pages 435–441, Istanbul, Turkey, July 4-8 2011. In Cooperation with the ACM, IEEE, IFIP, Co-Sponsored by IEEE Turkey, ASIM, EUROSIM, CASS, JSST, LSS, PTSK, TSS, Bahcesehir University.

[4] G. Hains, Chong Li, D. Atkinson, J. Redly, N. Wilkinson, and Y. Khmelevsky. Code generation and parallel code execution from business uml models: A case study for an algorithmic trading system. In *Science and Information Conference (SAI), 2015*, pages 84–93, July 2015.

[5] G. Hains, Chong Li, N. Wilkinson, J. Redly, and Y. Khmelevsky. Performance analysis of the parallel code execution for an algorithmic trading system, generated from uml models by end users. In *Parallel Computing Technologies (PARCOMPTECH), 2015 National Conference on*, pages 1–10, Feb 2015.

[6] Gaétan Hains, Chong Li, Youry Khmelevsky, Brandon Potter, Jesse Gaston, Andrew Jankovic, Sam Boateng, and William Lee. Generating a real-time algorithmic trading system prototype from customized uml models (a case study). 2012.

[7] Youry Khmelevsky, Gaétan Hains, and Chong Li. Automatic code generation within student's software engineering projects. In *Proceedings of the Seventeenth Western Canadian Conference on Computing Education*, WCCCE '12, pages 29–33, New York, NY, USA, 2012. ACM.

[8] Youry Khmelevsky, Martin Rinard, and Stelios Sidiroglou-Douskos. A source-to-source transformation tool for error fixing. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '13, pages 147–160, Riverton, NJ, USA, 2013. IBM Corp.

[9] M Norouzifard, SH Davarpanah, MH Shenassa, et al. Using natural language processing in order to create sql queries. In *Computer and Communication Engineering, 2008. ICCCE 2008. International Conference on*, pages 600–604. IEEE, 2008.

[10] Shaikh Sharmeen Momin Ummiya Tabrez Khan, Shaikh Shagufta. NLP to Create sql Query. *International Journal for Scientific Research and Development*, 3(9):95–98, 01/12/2015 2015.