A "real" query parser and Query Builder Widget

1. Summary

Rewriting Nepomuk's query parser mechanism to provide **1.** Extended syntax support with localized keywords handling **2.** Ability to create fast autocomplete mechanism **3.** Ability to build object based API on top on new query syntax. Also, proposal includes creation of autocomplete widget with integration it into KRunner and Dolphin.

2. Motivation

My proposal's idea is taken from KDE GSoC 2013 ideas list [1]. Currently Nepomuk supports simple queries, it can be seen by looking at [2] where is the whole list of existing keywords. Those queries may be sufficient for most cases at present, however looking to the future growth of Nepomuk, support for more advanced queries will eventually be necessary. A number of improvements may be made into current query syntax. Some of them are mentioned in ideas list, other may get into one's head in the future, especially considering Nepomuk's integration with other software[1]. What is more, it will be a big improvement if Nepomuk's query parser can "understand" query and thus provide an intelligent autocomplete mechanism directly to user.

At present, existing query parser consist of three modules:

- Parser which is responsible for processing queries into a syntax trees (mainly *QueryParser* class),
- A object-oriented syntax tree (*Query* class, related *Term* class and classes inherited from it),
- A compiler module involved into syntax tree classes which translate themselves into SPARQL final queries

[1] For example, putting a keyword "*only:music*" or "*only:movies*" could limit results to music/movie files and entries directly related to them (covers, lyrics, etc), which will may be useful not only for end-users, but also for KDE application programmers.

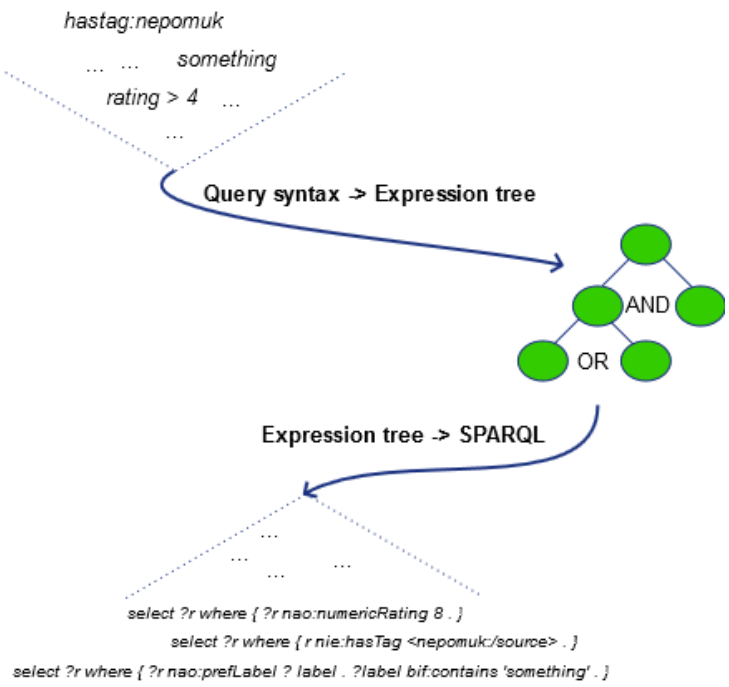

Figure 1. Simple diagram describing current architecture of Nepomuk's query parser.

This approach has a number of positive aspects, but has also one big disadvantage: more advanced query grammar improvements will entail hard to perform syntax tree classes modifications. Currently, it's need for support fully nested queries, but just extending query parser classes won't make the job; further modifications will still require a lot of time. Also, due to fact that query parser is made of regexp comparisons, creation of autocomplete mechanism is pretty complicated. Query parser needs of reimplementation.

3. Implementation Details

My proposition is to change its structure to make all of parsing a little bit simpler, but more powerful. Since there are mature and well tested generators of parsers, I'd like to use one of them. If so, we can focus on creation powerful grammar instead of writing and testing correctness of parsing module, because it'll be generated automatically. By saying generator of parser, I mean Bison (with Flex), which is already utilized in Kate editor and other KDE code, including kdelibs and kdepimlibs libraries.

After writing grammar and grammar-to-SPARQL compiler, code generated in Bison will be then responsible for interpreting queries directly into SPARQL without any additional logical layers[2]. This approach will be very desirable, because:

- It allows to create very detailed information about syntax errors just by design without a lot of effort. In final implementation, parsing bad or uncompleted queries will return detailed information about issues; this will allow to perform translation of valid, but incomplete query into more generic SPARQL request and provide powerful "intelligent" autocomplete mechanism.
- It will be simpler to add new grammatical rules (including recursive graph searching or even more sophisticated

[2] I realize there will be an internal expression tree made by Bison.

things) or modify existing ones.

Rewriting current query parser and providing more powerful query language based on current syntax is a first part of my proposal. By saying more powerful query language, I mainly mean adding support for multiple nested queries (for a reason to be more consistent with SPARQL language), and a lot of syntactic sugar. It will be quite easy to build simple pseudo natural language processor in addition to "normal" special keywords. For example: "*txt file bigger than 10MB created yesterday*" will mean exactly the same as "*filetype:txt size>10MB, created:RRRR-MM-DD*". To provide the best final grammar, I'll get inspiration and ideas by asking KDE users and developers and looking how it's made in other existing similar software, including great Nepoogle project.

After grammar creating and implementing compilation into SPARQL, I'll create an autocomplete mechanism. That mechanism will give propositions basing not only on previous searches and keywords, but also on existing Nepomuk's database. I'll be able to do it using another Bison's based parser. The parser will create more general SPARQL queries from unfinished queries. For example, query "*hastag:*" will generate "*select ?r where { ?r a nao:Tag . }*". Processed output from this query will be used as autocomplete hints. Obviously, more specific queries may be used to provide better results, although final implementation must depend on performance. I will prepare solution not very CPU and disk consuming.

The last part of my proposal is to create simple autocomplete widget and integrate it into existing projects. Most likely, it will be based on QCompleter mechanism with tree model. I will also provide all needed test cases and documentation.

Another and optional task will be implementation of expression tree which will compile not to SPARQL as now, but to new Nepomuk's query syntax. The reason of it is to provide easy to use, object oriented API for developers allowing them to perform complex queries to Nepomuk's db without knowledge of SPARQL.

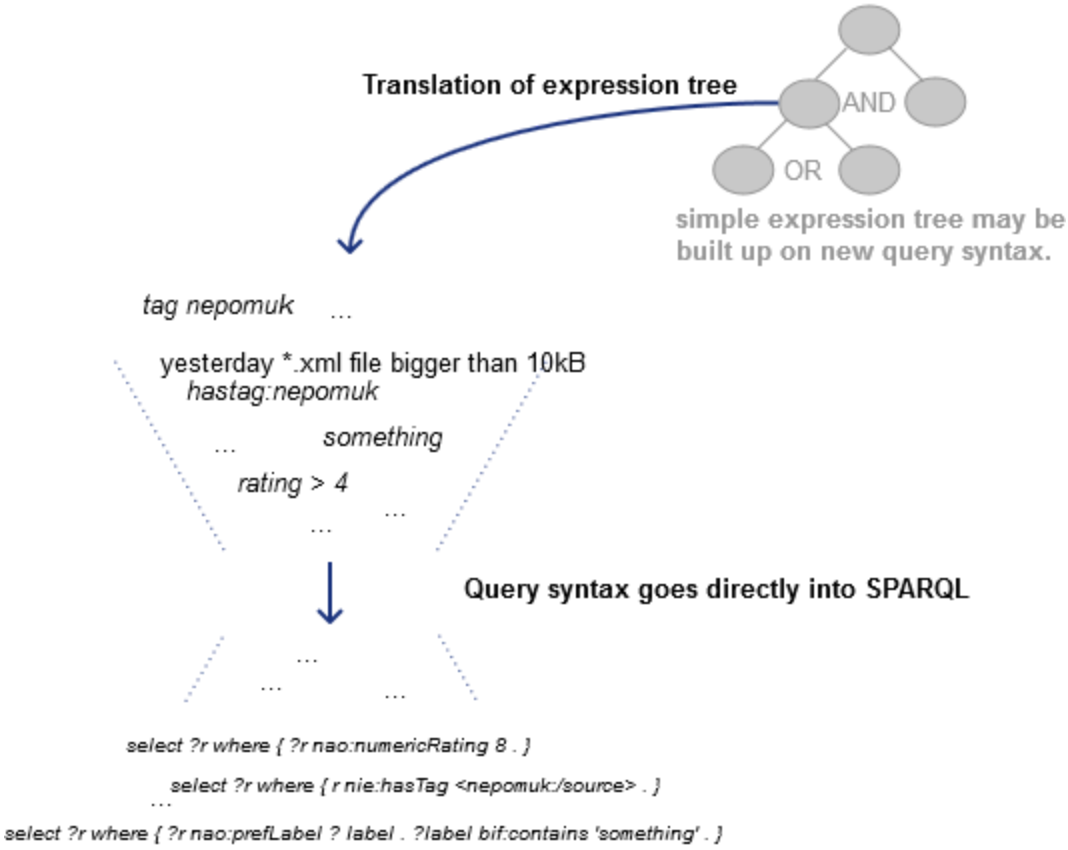

Figure 2. Diagram describing final architecture of Nepomuk's query parser.

I'm not sure whether it's not beyond GSoC time. If so, I'd be happy to create it anyway.

Those things will be done:

- New query parser mechanism built up on Bison and Flex with support for existing syntax, multiple nested queries and some localized fancy words as alternatives to existing keywords; also good error handling,
- Another query parser, also built on Bison and Flex which will be used to create general SPARQL queries based on uncompleted user's input,
- An "intelligent" autocomplete mechanism probably based on QCompleter, with Dolphin and KRunner integration

Those things probably not, but it will depend on time:

- New expression tree as a Nepomuk query API.

4. Tentative Timeline

27. May – 16. June: reading documentation, playing with SPARQL and Nepomuk, thinking about query grammar. Preparing environment, learning deeply Flex and Bison.

17. June – 7. July: Creating existing grammar with support for nested queries and implementing it. During this time I'll do some research and after this period, I'll have final grammatical rules.

8. July – 21. July: Still implementing grammar. Adding new localized words to it. Implementing mechanism which will translate those words into a standard representation. Also implementing a regexp based rules to recognize some kind of words as, for example, filenames, some other kind as tag names etc.

22. July – 4. August: After this time, all of finished query syntax translation into SPARQL should be done. Writing some documentation, testing code, starting implementing of second parser which will be used to provide autocomplete suggestions.

5. August – 19. August: Implementing second parser. Starting implementing autocomplete widget.

26. August – 8. September: Finishing implementing second parser and autocomplete widget. Integrating it into Dolphin and Krunner. Starting writing documentation and all missing tests.

9. September – 22. September: Improving documentation, doing some refactorization, maybe starting writing a proposed syntax tree.

5. About me

I'm a third year student at Computer Engineering Department at Silesian University of Technology in Gliwice, Poland. I've got practical knowledge of C++, Qt and Git. After school, I'm working as a developer in small local company. Also, I'm working on Nepomuk fileindexer optimization right now and that's where I get my experience as being an Open Source developer. I'd like to contribute to KDE not only during GSoC, but after it as well. I'm a hobbyist, I find creating various software as one of the most interesting activities. I know how to use Google, community boards and I'm not afraid to ask on IRC to get help when I am stuck. I'm also comfortable working independently under a mentor who is several thousand miles away.

Apart from coding, I'm a professional musician and it's one of my biggest hobbies. I play on clarinet and also, sometimes I play rock and roll on piano during house parties :) It's very funny and I like it a lot :)

Link for junior job and time availability

My finished junior job is here [3]. Also, I'm in the middle of reviewing new Nepomuk's RegExp cache mechanism in fileindexer which I'm an author. Link to bug: [4], link to related reviewboard page: [5]. During GSoC time, I will work 40 hours a week (from mid-June to mid-September) except one week in August when I'll work on project only for 2-3 hours a day. I can compensate it at weekends if necessary. I also have exams at University late June, but can make it in parallel to full time work on Nepomuk.

6. References

[1] http://community.kde.org/GSoC/2013/Ideas#Project:_A_.22real.22_query_parser_and_Query_Builder_Widget

[2] http://techbase.kde.org/Development/Tutorials/Metadata/Nepomuk/QueryService

[3] https://bugs.kde.org/show_bug.cgi?id=316418

[4] https://bugs.kde.org/show_bug.cgi?id=303654

[5] https://git.reviewboard.kde.org/r/109991/