Querying Source Code Using a Controlled Natural Language

Oleksandr Panchenko Stephan Müller Hasso Plattner Alexander Zeier
Hasso Plattner Institute for Software Systems Engineering
PO Box 900460, 14440 Potsdam, Germany
Email: {panchenko, stephan.mueller, office-plattner, zeier}@hpi.uni-potsdam.de

***Abstract*—Source code documents are of dual nature: they are in fact texts containing information for developers and they have an explicit structure for compilers and other tools. Several representations for the structured information of source code exist: abstract syntax tree, call graph, data flow graph, and others. Although the questions developers ask about source code seem easy to formulate, the complex code structure requires writing intricate queries. Developers use both, lexical and structured information for queries, though they dislike writing complex queries. Querying source code is an important activity in software development and maintenance. But often it cannot rely on predefined queries alone and requires writing more intricate queries. There is a need for a simple, user-friendly querying interface. This paper discusses an implementation of such a user interface based on a controlled natural language which is an unambiguous subset of the English language. When the developer enters the query, the source code grammar and the actual search results are used to automatically propose possibilities for query refinement and further navigation on the result set. The controlled natural language queries are then transformed to structured queries to retrieve data from a source code repository. The proposed approach provides a better expressiveness compared to simple keyword-based queries and enables consideration of complex structured relations between source code elements.**

***Keywords*-Source code repository; source code query language; development tools; controlled natural language.**

I. Introduction

The availability of efficient software development tools leads to lower development and maintenance costs, better code quality, and better organization of the development process. An important area of activities to be supported by tools is information gathering: searching source code, navigating through code and investigating related entities, performing various code inspections, calculating metrics, and mining code repositories. Source code repositories have been developed to enable code search using structured information contained in it. In general, this representation is based on a graph or a tree, whose vertices represent source code entities and whose edges represent relationships (implicit or explicit) between entities. Depending on the selected granularity level, entities can be single tokens in source code, modules, or entire subsystems. Examples of such structures are abstract syntax trees (ASTs), call graphs, data flow graphs, and module dependency graphs. Since each node contains lexical information, queries against such a data structure should be able to combine both types of data: keyword-based search and structured queries.

Usually, the repositories require some structured query language: e.g., SQL, XPath, XQuery, Relation Manipulation Language [1] or Datalog [2]. Given a complex grammar of a programming language and complex relations between entities in the repository, even simple developers' questions require writing complex queries. Although developers use both, lexical and structured information for queries, they refuse formulating complex queries. For certain kind of tasks (metrics calculation, code inspections, etc.) a complex notation is acceptable, because the queries are written once and the developer can reuse them. Other scenarios, however, require more interaction and the possibility to enter a query manually with an easily understandable syntax and semantic of the query language.

This paper introduces a user-friendly interface for flexible source code querying that allows for queries up to high complexity using a controlled natural language (CNL). Further, we use XPath as the underlying structural query language. XPath expressions are evaluated on tree representations of ASTs. In our previous work we have shown that AST-based representation of source code can leverage querying of syntactical patterns in source code [3]. While we started with a query language that is similar to the surface programming language, in this project we investigate CNL as a query language.

Instead of using keywords or code snippets, the developer can compose queries using a CNL. Queries in CNL are transformed into a suitable query language to retrieve data from a code repository. This paper exemplifies the proposed approach based on a repository which contains detailed structured information in the form of abstract syntax trees and uses XPath as a structured query language.

The following section relates our approach to existing research. An introduction of controlled natural languages is given in Section 3, and Section 4 introduces a simple use case. Section 5 describes our implementation details. We conclude the paper in Section VI and give an overview of possible directions for future work.

II. Related Work

Traditionally, regular expressions have been used to search source code. Although this simple method is useful for many tasks, the relations between source code entities are disregarded. Moreover, the search scope is limited to few files because of performance issues. To overcome performance limitations, Marcus et al. proposed to use an information retrieval approach to store lexical information in an inverted index [4]. However, structured information of the source code was not included in the index.

There is a large body of research on source code repositories and query languages. Bull et al. proposed to combine regular expressions with structured queries [5]. The approach works well for simple structured queries, but has limitations because of restricted expressiveness. A set of query languages (e.g., Relation Manipulation Language [1]), which are based on predicate calculus, have been used to query software artifacts. Hajiyev et al. have proposed to use safe Datalog, a query language based on the use of logic programming. Their tool, CodeQuest, maps safe Datalog queries to a relational database system [2]. JQuery is a tool supporting exploration of source code [6], [7]. The flexibility of the exploration views is achieved by a query-based customization of the content presented in the views. CodeQuest and JQuery aim at supporting high-level navigation and understanding software systems. Since existing source code querying systems store only coarse-grained software artifacts, the complexity of possible queries remains limited. But, as soon as the source code data model gains in complexity, queries become unwieldy.

The idea of providing a natural language interface for developers is not new. Würsch et al. presented a framework based on an OWL ontology to present data extracted by classical software analysis tools [8]. They used knowledge processing technologies from the Semantic Web and a guided-input natural language to answer questions about static source code information. The approach presented in our paper addresses developers' questions of the similar type. However, we focus on smaller syntactic code patterns based on AST representation of code. The major difference to the existing approaches is the capability of automatic generation of refinement proposals.

Further, there is a need for a flexible interface that enables keyword-based search over source code and takes into account its fine-grained and complex structure.

III. Controlled Natural Language

A controlled natural language is an unambiguous subset of a natural language with a restricted grammar and a domain-specific vocabulary. As a subset of the natural English language, CNL can be read and understood by a human user without any training. Although writing requires some training, it can be efficiently used to express formal statements, lowering the entry barrier to formal languages. Instead of learning a new language, the user is simply trained which subset of the ordinary language to apply. The writing process can be further supported by corresponding intelligent authoring tools. Among many available CNL implementations, Attempto Controlled English (ACE) stands out due to high research activity and a wide range of available tools [9] and is thus used in this project. The ACE Editor is an example of a menu-based editor that facilitates the construction of ACE sentences with no need to explicitly know the syntactical restrictions [10].

Being effectively a formal language, ACE is unambiguously translated to first-order logic, that is appropriate for reasoning about the expressed contents by the machine. Since many formal languages use first-order logic as a logical foundation, transformations to those languages are possible. In the ACE implementation, sentences are translated to discourse representation structures (DRS) which is a syntactical variant of first-order logic [11].

A CNL is applicable in a variety of areas. It can be used for software specifications, documentation, ontology authoring, rules and policy formulation as well as an interface to other formal languages. The idea of transforming CNL into a query language is not new. The tool LingoLogic is an implementation that translates a CNL to SQL [12]. The contribution of this paper is the use of the structure of the underlying data model and actual search result to generate refinement proposals and offer these to developers.

Opposed to plain keywords, a sentence written in CNL is capable of carrying more semantics. The words relate to one another and enable the construction of complex sentences with higher expressivity.

IV. Example Use Case

Generally speaking, many of the programmers' queries reported so far [13] can be answered by our approach. This paper illustrates the approach with several simple examples. In the test scenario, a developer wants to find all references to a variable *xyz*, where the variable is assigned a value. The structured query to answer this trivial question is quite complex: all references to the variable should be found; it should be ensured, that no other variable with the same name, but from another namespace, pertains to the result set. Finally, the references should be selected, where the variable is placed in the left side of the assignment operator. Since the query is supposed to be used ad hoc, developers expect the tool to provide this functionality at their fingertips. On the other hand, since there are thousands of such questions, it is not possible to prepare a list of queries for developers to select from. A simple, user-friendly query language would allow a flexible, handy interface to a complex information repository.

The proposed interactive approach guides the developer from a very simple keyword-based query to the required result by refining the query step by step selecting one of the proposed alternative queries. The developer starts with a simple keyword query:

$$xyz \tag{1}$$

In the background, this query is automatically extended to a correct CNL sentence that conforms to the ACE construction rules [9]:

$$Which\ entities\ are\ named\ xyz? \tag{2}$$

Then, the CNL form is translated into an XPath statement, which is executed by the source code repository. The result set is presented with alternatives for refinement, which are proposed by the query generator:

$$Which\ classes\ are\ named\ xyz? \tag{3}$$
$$Which\ entities\ are\ named\ xyz\ and\ are\ methods? \tag{4}$$
$$Which\ variables\ are\ named\ xyz? \tag{5}$$
$$Which\ entities\ are\ named\ xyz\ and\ are\ parameters? \tag{6}$$

At this point it is important to mention that the query generator checks if the proposed queries may return non-empty result set. In our example queries 3 and 4 will be hidden if there is no class or method called *xyz*.

The developer chooses the query 5 and hereupon the query generator proposes a set of further possibilities:

$$Which\ statements\ define\ a\ variable\ that\ is\ named\ xyz \tag{7}$$
$$Which\ statements\ use\ a\ value\ of\ a\ variable\ that\ is\ named\ xyz? \tag{8}$$
$$Which\ statements\ read\ a\ value\ of\ a\ variable\ that\ is\ named\ xyz? \tag{9}$$
$$Which\ statements\ change\ a\ value\ of\ a\ variable\ that\ is\ named\ xyz? \tag{10}$$

By selecting the query 10, the developer gets the intended query and the desired result set. XPath statements created and executed by the source code repository are as follows while query 11 corresponds to 2, 12 to 5, and 13 to 10:

$$//[. =' xyz'] \tag{11}$$
$$//IDENT[. =' xyz'] \tag{12}$$
$$//COMPUTE/RESULT/IDENT[. =' xyz'] \tag{13}$$

This example demonstrates how a query is interactively created with just a few clicks. Figures 1 and 2 illustrate the transformation of the query 10 into query 13. Figure 1 represents the query tree of the original CNL expression generated by the ACE parser. The phrase structure parsing approach decomposes the query into its elements. The query which is effectively a question consists of a noun phrase (np) and a verb phrase (vp), which are each refined recursively. While the noun phrase is made up of a question determiner (qt) and a noun (n), the verb phrase consists of a verb (v) and another noun phrase, decomposing it further.

Figure 2 shows the resulting query tree in which the CNL parse tree was transformed. For simplicity reasons, a XPath query tree, which is the main part of the 13, is shown. The description of each vertex in the query tree includes the type of the vertex (in the Figure 2 it is reflected as the name of the corresponding Java class in the upper box) and the local name of the instance of the class, as shown in the bottom box. Each edge corresponds to a movement along an axis and is labeled by the name of the corresponding axis. Some transformation patterns can be recognized: e.g., the vertex *rel_cl* (Figure 1) is transformed into the vertex *PredicateNodeTest* (Figure 2). Nevertheless, the complete list of transformation rules is still to be designed.

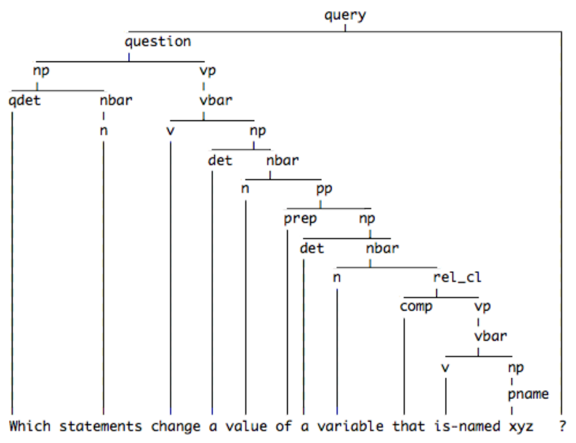

Figure 1. CNL parse tree of query 10

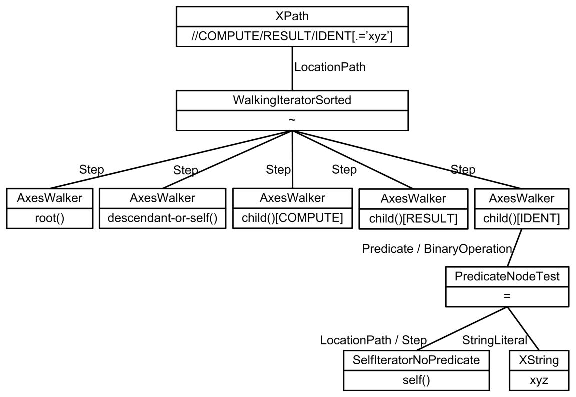

Figure 2. XPath query tree of query 13

V. Proposed Architecture

The proposed architecture is presented in Figure 3. The client part is implemented as an Eclipse plugin that enables

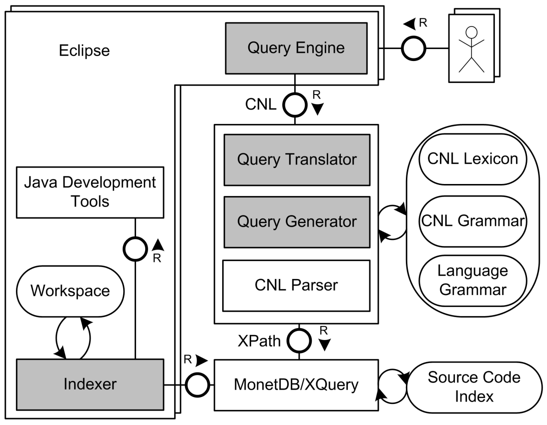

Figure 3. Prototype architecture

indexing of the local workspaces, which are sent to a central repository, and provides a user interface for querying. Eclipse Java development tools are used to parse source code and to construct ASTs. The server side is responsible for receiving developers' requests formulated in CNL, translating these requests into XPath, generating proposals for the query refinements, and returning the result set. As a database of the source code repository, MonetDB/XQuery is used which natively supports XQuery as a method for accessing data.

The central source code repository stores code in the form of ASTs. The indexer parses code into ASTs, annotates the vertices of the trees with meta data (class name, responsible, last editor, data and time of the last modification, etc.), and stores the trees in the index.

The primary function of the query translator is the transformation of CNL statements as provided by the developer, or selected from the proposed ones, into a logical form. The logical form is the DRS and can be regarded as the actual system language for information integration. The parsing is done by the Attempto parsing engine, though CNL grammar and CNL lexicon containing domain-relevant words and their grammatical categories are prepared in advance to describe a certain programming language. The resulting DRS of the query 10 is depicted below:

```
[A, B, C, D, E, F]
object(A, xyz, named, na, eq, 1)-1
query(B, which)-1
object(B, statements, countable, na, geq, 2)-1
object(C, value, countable, na, eq, 1)-1
object(D, variable, countable, na, eq, 1)-1
predicate(E, named, D, A)-1
relation(C, of, D)-1
predicate(F, change, B, C)-1
```

For querying the index, this logical form is transformed into the XPath query. The implementation of corresponding transformations is subject to the ongoing research. The logical form from the CNL parser can serve domain-independently as the integrative platform.

In order to represent the result of the query in a user-friendly way, it has to be structured appropriately and to yield functionality for further activities. For this purpose, once the query has been entered by the developer, the source code grammar and the actual search results are used to automatically propose possibilities for query refinement and further navigation on the result set. In this example to generate queries 7-10, source code grammar should reflect that a variable has a definition, variables are used in statements, and statements can read or change the value of a variable. All this information is already available in language specifications and should be made available to the CNL parser. Thus, a set of CNL queries are generated and proposed to the developer. This is facilitated through the ACE verbalization that takes the logical form, which is the DRS, and generates valid English sentences [14].

VI. Conclusion

The utilization of user-friendly interfaces for flexible information exploration in complex software environments leverages an indispensable contribution to a top-quality development framework, allowing for precise formulation of information needs. This leads to accurate information access, easy-to-use handling, flexibility and extensibility of interface functionality, high reusability in other domains, and significant lower development costs.

This paper discusses the usage of controlled natural language for querying source code. This approach is complementary to the keyword-based search and is a simple parlance that enables expression of complex relations between source code entities. Due to the fact that CNL is a subset of natural language, it can be read without any training.

The syntactic restrictions that have to be considered in the writing process are handled by a smart query authoring tool. Moreover, in most of the cases developers have only to select the query out of few automatically generated proposals.

The AST is not the only information to be indexed. There is a lot of relevant information available that is gathered in the development or maintenance process: test coverage measurements, code convention checks, change frequency, organizational metadata, customer complain messages, hot fixes, etc. This data can be made available as search criteria in the index.

References

[1] D. Beyer, A. Noack, and C. Lewerentz, "Efficient relational calculation for software analysis," *IEEE Transactions on Software Engineering*, vol. 31, no. 2, pp. 137–149, 2005.

[2] E. Hajiyev, M. Verbaere, and O. de Moor, "CodeQuest: Scalable Source Code Queries with Datalog," in *Proceedings of the European Conference on Object-Oriented Programming*, ser. LNCS, vol. 4067. Springer, 2006, pp. 2–27.

[3] O. Panchenko, J. Karstens, H. Plattner, and A. Zeier, “Precise and Scalable Querying of Syntactical Source Code Patterns Using Sample Code Snippets and a Database,” in *Proceedings of the 19th IEEE International Conference on Program Comprehension*, 2011, pp. 41–50.

[4] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, “An Information Retrieval Approach to Concept Location in Source Code,” in *Proceedings of the Working Conference on Reverse Eng.*, 2004, pp. 214–223.

[5] R. I. Bull, A. Trevors, A. J. Malton, and M. W. Godfrey, “Semantic grep: regular expressions + relational abstraction,” in *Proceedings of the 9th Working Conference on Reverse Engineering*, 2002, pp. 267–276.

[6] E. McCormick and K. D. Volder, “JQuery: Finding Your Way through Tangled Code,” in *Proceedings of the Conf. on Object-oriented programming systems, languages, and applications*. ACM, 2004, pp. 9–10.

[7] D. Janzen and K. D. Volder, “Navigating and querying code without getting lost,” in *Proceedings of the international conference on Aspect-oriented software development*. ACM, 2003, pp. 178–187.

[8] M. Würsch, G. Ghezzi, G. Reif, and H. C. Gall, “Supporting Developers with Natural Language Queries,” in *Proceedings of the International Conference on Software Engineering*. IEEE Computer Society, 2010.

[9] N. E. Fuchs, K. Kaljurand, and T. Kuhn, “Attempto Controlled English for Knowledge Representation,” in *Reasoning Web, Fourth International Summer School*, ser. LNCS, C. Baroglio, P. A. Bonatti, J. Małuszyński, M. Marchiori, A. Polleres, and S. Schaffert, Eds., no. 5224. Springer, 2008, pp. 104–124.

[10] T. Kuhn and R. Schwitter, “Writing Support for Controlled Natural Languages,” in *Proceedings of the Australasian Language Technology Association Workshop*, 2008, pp. 46–54.

[11] H. Kamp and U. Reyle, *From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Dordrecht: Kluwer, 1993.

[12] C. W. Thompson, P. Pazandak, and H. R. Tennant, “Talk to Your Semantic Web,” *IEEE Internet Computing*, vol. 9, no. 6, pp. 75–78, 2005.

[13] B. de Alwis and G. C. Murphy, “Answering Conceptual Queries with Ferret,” in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE. New York, NY, USA: ACM, 2008, pp. 21–30.

[14] N. E. Fuchs, “Verbalising Formal Languages in Attempto Controlled English,” University of Zurich, deliverable I2-D5, 2005.