.QL: Object-Oriented Queries Made Easy

Oege de Moor, Damien Sereni, Mathieu Verbaere, Elnar Hajiyev,
Pavel Avgustinov, Torbjörn Ekman, Neil Ongkingco, and Julian Tibble

Semmle Limited

Abstract. These notes are an introduction to .QL, an object-oriented query language for any type of structured data. We illustrate the use of .QL in assessing software quality, namely to find bugs, to compute metrics and to enforce coding conventions. The class mechanism of .QL is discussed in depth, and we demonstrate how it can be used to build libraries of reusable queries.

1 Introduction

Software quality can be assessed and improved by computing metrics, finding common bugs, checking style rules and enforcing coding conventions that are specific to an API. Many tools for these tasks are however awkward to apply in practice: they often detract from the main task in hand. Above all, it is tough to customise metrics and rules to one's own codebase, and yet that is where the greatest benefit lies.

These lectures present a new approach, where all these tasks related to software quality are phrased as *queries* over a relational representation of the code base. Furthermore, the language for expressing these queries is object-oriented, encouraging re-use of queries, and making it easy to tailor them to a specific framework or project. While code queries have been considered before (both in industry and academia), the object-oriented query language (named .QL) is unique, and the key to creating an agile tool for assessing software quality.

As an advance preview of .QL, let us briefly consider a rule that is specific to the *Polyglot* compiler framework [46]. Every AST node class that has children must implement a method named "visitChildren". In .QL, that requirement is checked by the query:

```
class ASTNode extends RefType {
  ASTNode() { this.getASupertype+().
                hasQualifiedName("polyglot.ast","Node") }
  Field getAChild() {
     result = this.getAField() and
     result.getType() instanceof ASTNode
  }
}
from ASTNode n
where not(n.declaresMethod("visitChildren"))
select n, n.getAChild()
```

R. Lämmel, J. Visser, and J. Saraiva (Eds.): GTTSE 2007, LNCS 5235, pp. 78–133, 2008.

Of course this may appear rather complex to the reader for now, but the example still serves to illustrate a couple of important points. First, this is a very useful query: in our own research compiler for *AspectJ* called *abc* [4], we found no less than 18 violations of the rule. Second, the query is concise and in a syntax resembling mainstream languages like SQL and Java. Third, as we shall see later, the class definition for *ASTNode* is reusable in other queries.

.QL has been implemented as part of an Eclipse plugin named *SemmleCode*. SemmleCode can be used to query any Java project. It provides an industrial-strength editor for .QL, with syntax highlighting, autocompletion and so on (as shown in Figure 1). Furthermore, the .QL implementation itself is quite efficient. Java projects are stored in relational form in a standard database system. That database system can be a cheap-and-cheerful pure Java implementation such as H2 (which is distributed with SemmleCode), or a dedicated system such as PostgreSQL or SQL Server. With a proper database system in place, SemmleCode can easily query projects that consist of millions of lines of code. That scalability is another unique characteristic that sets SemmleCode apart from other code querying systems.

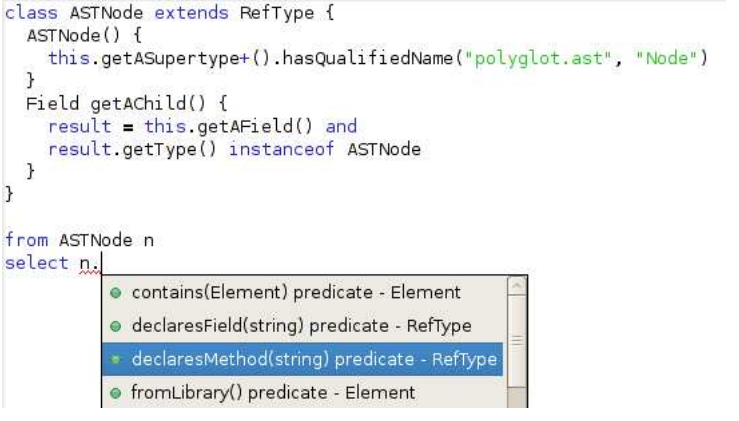

Fig. 1. The SemmleCode editor

.QL is in fact a general query language, and could be thought of as a replacement for SQL — the application to software quality in these notes is just an example of its power. Like SQL, it has a simple and intuitive syntax that is easy to learn for novices. In SQL, however, that simple syntax does not carry over to complex constructs like aggregates, while in .QL it does. Furthermore, recursion is natural in .QL, and efficiently implemented. Compared to the direct use of recursion in SQL Server and DB2, it can be orders of magnitude faster. Finally, its object-oriented features offer unrivalled flexibility for the creation of libraries of reusable queries.