

Evaluation of code query technologies for industrial use

Tiago L. Alves

University of Minho, Portugal and
Software Improvement Group, The Netherlands
t.alves@sig.nl

Peter Rademaker

University of Utrecht, The Netherlands and
Software Improvement Group, The Netherlands
p.rademaker@sig.nl

Abstract

We have investigated the possibility of using emerging code query technologies in industry. For that purpose we evaluated three alternatives: Crocopat, Rscript, and SemmlCode. We made a comparison with respect to eight criteria focussing on language features and tool integration issues. Although the available solutions are promising we found that none of them fully satisfies our requirements. In particular, we found that the combination of good abstraction and extension facilities lacks in all languages, and that an API is missing from all tools.

We recognize a need for a solution that offers the language abstraction facilities offered by Rscript and the extendability of SemmlCode combined with an API to achieve a smooth integration with existing technologies. We expect that meeting these challenges will expedite industrial adoption.

1. Introduction

Query technologies are a more advanced alternative to traditional ways of querying source code. Instead of imperatively programming visitors, query technologies offer abstraction over implementation details by providing a domain-specific language in which code queries can be implemented in a more abstract but yet more powerful way. Several solutions already exist, but are these solutions mature enough to be embraced by industry?

In this paper we compare and evaluate three tools: Crocopat [1, 2], Rscript [7], and SemmlCode [3]. We use criteria covering language features, licensing, user interface, API availability and other technical issues.

Section 2 explains the context in which the evaluation was made. In Section 3 we compare and evaluate code query tools and end stating our position. Finally, in Section 4 we conclude and pose challenges to be met.

2. Background

The Software Improvement Group (SIG) offers three main services, to help companies manage their systems: software monitoring [8, 9], software risk assessments [11], and automatic documentation generation [4].

At the core of all these services lays a software analysis framework that helps consultants to identify problems in the systems. This framework was built with three requirements in mind:

- cope with incomplete code;
- analyze very large systems (> 1M LOC);
- support multiple programming languages.

The framework extracts facts and relations from these systems which are then represented as a graph and persisted in a database. To answer the questions of SIG consultants and clients, this information is then processed using visitors and presented in a web-based interface.

Whenever new questions about the software systems must be answered, this is done by implementing new visitors and redeploying the framework.

This approach, however, suffers of three main shortcomings: (1) it is not very flexible (new queries must be programmed against the framework requiring deployment of a new version of the framework); (2) reuse of the queries for different programming languages is difficult; and (3) queries are defined in an imperative and verbose way.

As part of the SIG's effort to continuously improve its own tooling we started to investigate code query technologies to overcome the identified shortcomings.

With the use of query technologies we would expect to minimize the work necessary for implementing specific queries, maximize the sharing between queries for different programming languages gaining consistency and be able to formulate queries in a more concise and declarative way.

3 Comparison of existing solutions

In this section we compare three software querying tools.

3.1 Query technologies

Croccopat [1, 2] is an interpreter for programs written in the RML language. RML is an imperative relational programming language, the relational expressions in RML are based on predicate logic. Besides relational expressions in the form of predicate logic, the language includes control structures, numerical expressions, and some basic input and output facilities.

Rscript [7] is a small scripting language based on the relational calculus. The language has scalar types (Boolean, integer, string, location) and composite types (set and relation). Expressions can be constructed from comprehensions, function invocations and operators. Rscript is developed as part of a framework for language development, source code analysis and source code transformation, but is currently also available as standalone tool.

SemmlCode [3] is a commercial tool offered by Semml Ltd. It differs from the other evaluated tools in that it uses a relational database to store and query relations. SemmlCode is offered as an Eclipse plugin that allows you to query the source code in a project using .QL, an object-oriented query language similar to SQL with an added transitive closure operation. A .QL query is translated to SQL and executed by the RDBMS.

3.2 Comparison approach

To objectively compare the expressiveness, abstraction, and extendability of the languages, we have implemented four queries: lifting of a call graph [5], detection of the degenerate inheritance pattern [1], computation of the package instability metric [10], and forward graph slicing [12]. Together, these queries cover a large part of the spectrum of software analysis queries.

We compared the tools with respect to eight criteria: style/paradigm, abstraction, extendability, licensing, user interface, API availability, interchange format, and extractor support.

We have omitted performance from the criteria since it is not our highest priority due to the fact that SIG analysis are meant to run on a weekly basis. Nonetheless we acknowledge the importance of well-performing query tools, in particular when one would like to query large systems interactively.

3.3. Comparison

Table 1 summarizes the comparison of the three solutions.

1. **Style/paradigm:** We compared the languages by specifying the four queries in each tool. In every language

we were able to express these queries, however, differences can be found in how the queries are expressed. We have included one of these queries, call graph lifting, in Figure 1. This example illustrates some of the differences in styles.

- Croccopat offers an imperative language based on first order logic and therefore required us to specify the lifting query in a point-wise style using existential quantifiers.
 - Rscript allowed us to specify a query either using point-free binary relational calculus, or in a point-wise style using comprehensions.
 - SemmlCode’s .QL language required us to specify a query in a point-wise, SQL-like style with added object orientation.
2. **Abstraction:** We evaluated whether the languages allow abstraction over the model or application used, in particular *abstraction by parametrization* (e.g. a slicing query parameterized with its starting criteria), and *abstraction over the type of the relation* or parametric polymorphism (e.g. a generic lifting function). In Rscript relations are first-class citizens which allows a relation to be passed as parameter to functions, assigned to variables, and returned as result of a function. In addition, Rscript supports parametric polymorphism. Besides Rscript, no other tool offers abstraction facilities in its language.
 3. **Extendability:** We evaluated the possibility of extending the language either with new operators or with new libraries. Only SemmlCode supports adding user-defined libraries. However, SemmlCode lacks of proper import mechanism: only one library file can be loaded, obliging user-defined extensions be stored in the standard library file. The other two tools, on the other hand, offered no support for this feature. But since these tools are open-source, extending the language by adding operators or other language constructs can, in principle, be achieved by changing the parser and the compiler.
 4. **Licensing:** We investigated which license each technology is licensed under since that can be an issue for adopting the technology in industry. The license of SemmlCode is a proprietary license while the other two come with open-source licenses. Rscript is licensed under the BSD license, which allows it to be incorporated into proprietary commercial products. Croccopat is licensed under GNU LGPL which also allows incorporation into proprietary software, but with additional limitations on redistribution.
 5. **User interface:** We evaluated the user interfaces offered by each of the tools. Both Croccopat and Rscript offer a command-line interface (CLI) which is fairly

	Crocopat	Rscript	SemmlCode
Style/paradigm	Imperative + FO logic	Relational + comprehensions	SQL-like + object orientation
Abstraction	no	yes	no
Extendability	no	no	yes
Licensing	GNU LGPL	BSD	Proprietary
User interface	CLI	CLI + IDE	Eclipse plugin
API availability	no	no	no
Interchange format	RSF	Rstore	no
Extractor availability	no	no	Java + XML

Table 1. Summary of comparison results

easy to use. Additionally, Rscript offers an IDE. SemmlCode is available as an Eclipse plugin with user friendly features such as syntax completion, but is not shipped with other integration options.

6. **API availability:** We investigated whether any of the tools offer an API to enable the integration of query technologies with existent tooling. None of the tools offer an API, and integration for Crocopat and Rscript is only possible via tool execution. SemmlCode can only be executed interactively from within Eclipse, an API is not available.
7. **Interchange format:** We investigated the support for an interchange format, both for reading facts and for saving results. Only Crocopat and Rscript support interchange formats. While Crocopat supports Rigi Standard Format (RSF), Rscript supports a generalized version of RSF, called Rstore. SemmlCode does not support any interchange format: it provides its own fact extractors for importing relations into a database, and supports only text export.
8. **Extractor availability:** We investigated the availability of extractors for each of the tools. For Crocopat or Rscript, we are not aware of the existence of any existing extractor. However, the existence of an interchange format enables us to create our own extractor. In the Rscript documentation [7], you can find detailed instructions on how to build an extractor from an SDF grammar [6]. SemmlCode, on the other hand, has an integrated extractor for the Java programming language and XML format. The lack of an interchange format precludes connections to other (pre-existent) solutions.

3.4 Evaluation

Crocopat’s imperative style and use of existential and universal quantifiers lacks declarativeness and ease of expression. For integration with other components it does not offer the convenience of an API, but note that the CLI and

support for the RSF format also allow a (limited) form of interoperability.

In RScript relations are first-class citizens which makes it the only solution that makes it possible to define parameterized queries. In addition to parametrization it offers parametric polymorphism which allows the expression of generic queries. The fact that Rscript is a language based on relational calculus allows declarative and concise definition of queries. It comes with the BSD license which is the most permissive license of the three. Regarding integration issues Rscript is comparable to Crocopat.

Because of its similarity to SQL, SemmlCode’s .QL language is a familiar and therefore easy to learn language for a wide range of users. A disadvantage is that it lacks the abstraction facilities that Rscript offers. SemmlCode is the only tool that offers extendability of the language through libraries, with the limitation that a proper import mechanism to allow the use of user-defined libraries is currently missing. SemmlCode comes with two integrated extractors but it does not allow the addition of new extractors. It comes with a closed license which might pose problems for adoption in industry.

After evaluating the three solutions we conclude that none of them fully satisfies all our requirements. We recognize a need for a solution that offers the language abstraction facilities of Rscript and the extendability of SemmlCode (with a proper import mechanism) combined with an API to achieve a smooth integration with existent technologies and workflows.

4. Conclusions

We have compared and evaluated several code querying tools: Crocopat, Rscript and SemmlCode. The general purpose of this study was to assess whether emerging code querying technologies are mature enough for industrial use.

Although each one of them offers a rich set of features none of the solutions totally fulfills our requirements: none of the solutions provides both language abstraction facilities

and extendability, and none of them offers API access.

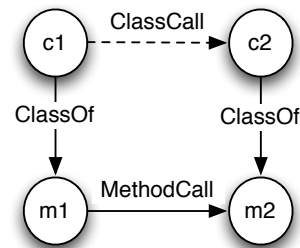
We pose the challenge to create a solution that offers extendability as well as abstraction on the language level, and on the tool level provides an API for optimal integration possibilities.

Acknowledgements Thanks to Joost Visser for useful discussions and his comments on this paper. The first author is supported by the *Fundação para a Ciência e a Tecnologia*, grant SFRH/BD/30215/2006.

References

- [1] D. Beyer. Relational programming with crocopat. In *International conference on Software engineering (Tool Demo)*, pages 807–810, Shanghai, China, 2006. IEEE.
- [2] D. Beyer and A. Noack. Crocopat 2.1 introduction and reference manual. Technical Report UCB/CSD-04-1338, EECS Department, University of California, Berkeley, Jul 2004.
- [3] O. de Moor, M. Verbaere, E. Hajiyev, P. Avgustinov, T. Ekmann, N. Ongkingco, D. Sereni, and J. Tibble. Keynote address: .ql for source code analysis. In *Proc. of the Seventh IEEE Int. Working Conf. on Source Code Analysis and Manipulation (SCAM 2007)*, pages 3–16, Washington, DC, USA, 2007. IEEE Computer Society.
- [4] A. v. Deursen and T. Kuipers. Building documentation generators. In *Proc. Int. Conf. on Software Maintenance*, pages 40–49. IEEE Computer Society, 1999.
- [5] L. Feijs, R. Krikhaar, and R. V. Ommerring. A relational approach to support software architecture analysis. *Softw. Pract. Exper.*, 28(4):371–400, 1998.
- [6] J. Heering, P. R. H. Hendriks, P. Klint, and J. Rekers. The syntax definition formalism SDF — Reference manual. *SIG-PLAN Notices*, 24(11):43–75, 1989.
- [7] P. Klint. *A tutorial introduction to Rscript*. CWI, May 2005.
- [8] T. Kuipers and J. Visser. A tool-based methodology for software portfolio monitoring. In M. Piattini and M. Serrano, editors, *Proc. 1st Int. Workshop on Software Audit and Metrics, (SAM 2004)*, pages 118–128. INSTICC Press, 2004.
- [9] T. Kuipers, J. Visser, and G. de Vries. Monitoring the quality of outsourced software. In J. van Hillegerberg et al., editors, *Proc. Int. Workshop on Tools for Managing Globally Distributed Software Development (TOMAG 2007)*. Center for Telematics and Information Technology (CTIT), The Netherlands, 2007.
- [10] R. C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall PTR, USA, 2003.
- [11] A. van Deursen and T. Kuipers. Source-based software risk assessment. In *ICSM '03: Proc. Int. Conference on Software Maintenance*, page 385. IEEE Computer Society, 2003.
- [12] M. Weiser. Program slicing. In *Proceedings of the 5th International Conference on Software Engineering*, pages 439–449. IEEE Computer Society Press, 1981.

When analyzing large software systems, views on different levels of abstraction are needed. The canonical example is that of a call graph on method level that is lifted to a higher level. The following diagram depicts the particular case where the method calls are lifted to the class level.



This query can be expressed in Crocopat, Rscript, and SemmleCode as follows:

Crocopat

```
ClassCall(c1, c2) :=
  EX(m1, EX(m2, ClassOf(c1, m1) & ClassOf(c2, m2)
    & MethodCall(m1, m2));
```

EX denotes the existential quantifier.

Rscript - relational calculus

```
ClassOf o MethodCall o inv(ClassOf)
```

The \circ operator denotes relational composition, the $\text{inv}()$ method inverses a relation.

Rscript - comprehension

```
{<C1, C2> | <method M1, method M2> : MethodCall,
  <class C1, class C2> : ClassOf[M1] x ClassOf[M2]
}
```

Where $\text{ClassOf}[M1]$ is the right image of the ClassOf relation with respect to $M1$, \times denotes cartesian product. Both `method` and `class` are type synonyms for the string type.

SemmleCode

```
from Class c1, Class c2
where c1.getACallable().calls(c2.getACallable())
select c1, c2
```

Class is the type of the `c1` and `c2` objects. `getACallable()` denotes a method invocation, returning an object of the type `Callable`.

Figure 1. Call graph lifting using Crocopat, Rscript, and SemmleCode