

QualiXML - an XML-based Static Code Analysis Framework

Dimitri Durieux, Christophe Ponsard, and Jean-Christophe Deprez
CETIC Research Centre
Charleroi, Belgium
email: {ddu,cp,jcd}@cetic.be

Abstract—QualiXML is (yet another) static code analysis framework, initially developed to answer a industrial request of quick and flexible code quality checks. It relies on XML to represent the Abstract Syntax Tree and enables the use of standard XML tools to compute various quality metrics. Initially limited to C++ and standards metrics, the tool evolved towards a flexible framework with multi-language support and the ability to handle technology specific processing (like mobile framework specificities). The framework is designed to ease the integration of new languages following the object-oriented paradigm through the addition of ANTRL parsing to the XML proposed format. The author's intend is to release QualiXML under an Open Source Software license.

I. MOTIVATION

There exist many good code analysis frameworks both in the Open Source [1] and commercial world [2]. Some of these frameworks cover many mainstream languages, provide rich analysis capabilities and support a variety of reports to present analysis results. However a number of practical use cases require the use of standalone code analyser for instance, to perform pre-commit checks or specific type of analysis. Many of the existing solutions proved inadequate to address these specific situations because of their closed nature or monolithic structure. Other frameworks combining various tools also lack homogeneity or would require significant adaptation effort to fit the need reusing the same analyser across different languages.

A possible option is the use of a meta-engineering framework like RASCAL, which aims at rapidly constructing analysis, transformation, generation or visualization of source code written in different programming languages [3]. However, the practical request from a customer is to rely on XML technology to represent the abstract syntax tree (AST) so that XML technology (XSLT, XPATH, DOM...) could then be used to carry out analyses. Furthermore, the customer request is to replace the SrcML tool [4] which proved not reliable enough.

Although the first QualiXML implementation was limited to C++, relying on ANTRL technology for the parsing helped to extend fairly easily to other languages like Java [5]. Furthermore QualiXML also integrates with SonarQube for the presentation of results.

II. FRAMEWORK ARCHITECTURE

The overall dataflow architecture of QualiXML is depicted in Figure 1 and is composed of the following steps.

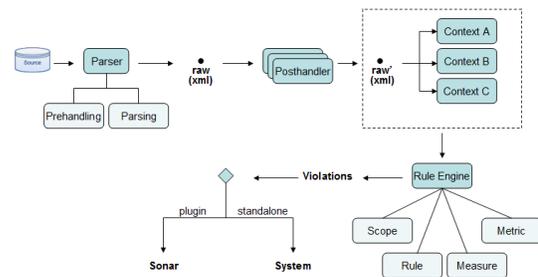


Fig. 1. Framework Architecture

- **Parsing.** QualiXML gets a representation of the source code as a XML tree after a parsing step. The parsing can accommodate different parsers but we preferably rely on ANTRL technology for which many grammars are already available. Visitors are applied to get a XML output complying to a single grammar for different object-oriented languages and to enable the application of the same rules across different languages hence ease the alignment of quality models on different lanaguages.
- **Post-handlers.** A post-handler mechanism is available to decorate the raw XML with attributes to capture typing information, data flow information among others. Attribute values will then ease the computation of rules. A number of standard handlers are available for type inference or filtering, e.g. for class level analysis.
- **Context definition.** A context is a more recent QualiXML feature that provides an abstract view on families of languages or technology. Context information is used to facilitate the alignment of quality models, for example standard OO languages metrics can be defined on an OO context. Specific profiles can also be defined, e.g for relational databases.
- **Rule Engine.** Predefined rules are applied on the XML AST to generate a violation report. A general configuration ensures proper post-handling. Rules can have dependencies on each other (low level metrics used by higher level rules). QualiXML uses smart propagation mechanisms across its network of rules to optimise the

evaluation. Internally, a violation report is represented in an XML format describing the violations as a set of tuples (one per violation) composed by the violated rule, the violating artefact, the line where the violation is located and a human readable message giving information about the violation.

- **Result Exploitation.** Violation reports are used either in the scope of a specific run-time system (e.g. pre-commit checks) or inside a code quality dashboard. SVN hooks and an integration plugin for SonarQube are available.

QualiXML currently supports C++, Java and partly PHP. It is implemented in Java and relies on standard XML libraries like XSL1.0 transformation, dom4J, XPATH 1.0.

III. SOME USAGE SCENARIOS

A. Pre-commit Checks

QualiXML can be used as a pre-commit hook to validate a portion or the whole source code. This validation ensures that the code complies to a set of predefined rules. The commit is rejected if at least one violation is detected. The installation as a pre-commit hook consists of a simple script calling QualiXML standalone deployment.

The set of predefined rules are constructed manually using profile templates. Profiles can be managed internally or externally to QualiXML. The internal QualiXML profile manager simply consists of a specific XML file gathering the profiles. A profile is a set of language-dependant rules that can be activated depending on the analysis context. External profile management is done through the use of an external tool that defines a quality model. We currently only support the SonarQube profile's management. This allows us to manage all the QualiXML profiles in a central SonarQube web platform and to generate dynamically the XML configuration at analysis time.

B. Cross-Technology Analysis

Most projects consist of a mix of code in different languages potentially based on specific technologies like relational databases. A typical scenario is to check, the consistency of a database model and the entities declared in a Java code. Three QualiXML features ease this kind of analysis.

- **dynamic profile composition:** during the analysis, QualiXML dynamically chooses the profile for each technology detected in the code and creates a "composite profile" based on the detected technologies. Rules are then applied consistently on each supported technology.
- **unique and centralized representation:** QualiXML merges the different parsing results into a single XML AST file.
- **use of contexts:** The XML AST structure provides an abstraction level thus source code mixing different OO technologies can be analysed using the same set of rules. This will also reduce the number of technology interactions to consider after parsing.

IV. PRO AND CONS

On the positive side, the solution has a simple design and the use of XML facilitates adoption because tuning rules is a matter of writing XPath expressions. XPath is well known in Industry. Larger evolution effort for instance, to integrate a new language proved quite reasonable given the level of reuse that can be achieved inside the platform. Relying on an homogeneous set of rules across technologies is an advantage to enable the comparison of quality practices across languages.

A main drawback is the performance both in time and memory. We performed comparison benchmarks between QualiXML and PMD on large projects (e.g. Apache Xalan, Tomcat codebases). Although the tool can scale, there is a time overhead factor between 4 and 7 (lower for larger projects) and a memory overhead where QualiXML can use up to twice the memory space used by PMD. In order to address those issues, we first introduced a filtering mechanisms that only produce the XML AST at the required level of granularity. A second enhancement is a mechanism for processing the tree representation in-memory. Note that not all usage scenarios requires to process a large number of files. For example, in pre-commit checking, usually small number of files are committed at a time.

QualiXML has shown useful in specific industry cases thanks to its simple structure and ease of tuning. It can also support R&D activities, especially in the fields of software quality and software evolution. That is why we decided to engage in the process of releasing it under an Open Source licence in early 2016 through some popular forge.

V. ROADMAP

Our short term roadmap is to keep enriching the rule library and support other languages. We presently work on supporting the new SWIFT language developed by Apple, with a focus on security rules. We are also working on enhancing the performance overhead and preparing the code for an Open Source release through a thorough cleansing and documentation. More contexts are also being defined.

ACKNOWLEDGMENT

This work was funded by the ASCETiC project (nr 610874). Many thanks to Felipe Fabio (U. Valparaison, Chili) and Dang-Minh Nguyen (TU. Troyes, France) for contributing.

REFERENCES

- [1] G. A. Campbell and P. P. Papapetrou, *SonarQube in Action*, 1st ed. Greenwich, CT, USA: Manning Publications Co., 2013.
- [2] CAST Software, "Code Analysis tools."
- [3] P. Klint, T. v. d. Storm, and J. Vinju, "Rascal: A domain specific language for source code analysis and manipulation," in *Proc. of the Ninth IEEE Int. Working Conf. on Source Code Analysis and Manipulation*, 2009.
- [4] M. L. Collard, M. J. Decker, and J. I. Maletic, "Lightweight transformation and fact extraction with the srcml toolkit," in *11th IEEE Working Conference on Source Code Analysis and Manipulation*, 2011.
- [5] T. Parr, *The Definitive ANTLR 4 Reference*, 2nd ed. Pragmatic Bookshelf, 2013.