Static analysis of textual models

Iván Ruiz Rube, Tatiana Person, and Juan Manuel Dodero

Universidad de Cadiz,
Avenida de la Universidad de Cádiz, 10, 11510 Puerto Real (Cádiz), España
ivan.ruiz@uca.es,tatiana.personmontero@alum.uca.es,juanma.dodero@uca.es
http://www.uca.es

Abstract. Domain specific languages (DSLs) based on textual notations are useful to describe the semantics of a given problem. Software frameworks, such as Xtext, enable to easily design and develop textual DSLs. The use of interactive quality platforms for analysing source code such as SonarQube is increasing. For evaluating the quality of a program written with an Xtext-designed DSL, all the artifacts required by SonarQube to parse and query the source code must be developed, which becomes time-consuming and error-prone. A transformation tool and its application to a DSL are presented to bridge the gap between Xtext and SonarQube grammar formats by following a model-driven interoperability strategy.

Keywords: DSL, Xtext, SonarQube, Model-driven, Interoperability

1 Introduction

Domain specific languages (DSLs) are computer languages which are targeted to particular kinds of problems, rather than general purpose languages that are aimed at any kind of software problems [3]. In the recent years, various tools have arisen to easily develop DSLs. In this sense, Eclipse Modeling Project gathers most of the libraries, frameworks and tools to deal with the design and development of external DSLs. These tools fall into the Model Driven Software Engineering approach, which promotes model design, development, transformation and use to conduct the software process lifecycle [2]. Xtext [1] is the Eclipse Modeling Project framework used to develop programming languages and DSLs by means of a dedicated grammar language.

The use of interactive quality platforms for analysing source code such as SonarQube is increasing. SonarQube provides support for more than 20 different languages and allows users to add their own rules and advanced metrics. Providing Xtext-designed DSLs with features for evaluating the quality of the programs written with these languages could be a additional factor to contribute to the success of their adoption [4].

2 Bridging grammar formats

SonarQube includes an extension mechanism for recognising new languages and including new rules to check programs written with some of the built-in lan-

guages or the new ones. In order to evaluate the quality of programs written with Xtext-designed DSLs or to compute some metrics, we would have to develop all the artifacts required by SonarQube to parse and query the Abstract Syntax Tree (AST) of the source code parts. This task is time-consuming and error-prone, especially when it comes to maintain the consistency of the grammars while evolving the language. Afterwards, several AST visitors should be implemented to analyse quality rules or compute measures. Xtext grammars are designed by using its own specific language to describe the concrete syntax of the new language and how it is mapped to an in-memory semantic model. Xtext uses the well-known ANTLR parser [5], which implements an LL top-down parse algorithm for a subset of context-free languages. The grammar language uses Extended Backus-Naur Form (EBNF) expressions. However, the development of language recognisers in SonarQube is quite diferent. Instead of writing a grammar by using EBNF expressions, SonarQube parsing is implemented as Java classes that use a specific library called SonarSource Language Recogniser[1] (SSLR). SSLR is a lightweight Java library that provides everything required to create lexers and parsers for analysing a piece of source code.

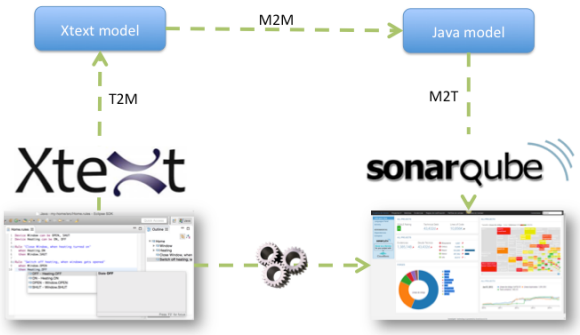

Fig. 1: Interoperability strategy

Is it possible to bridge the gap between the formats of Xtext and SonarQube grammars? This kind of problem can be tackled by adopting a model-driven interoperability approach. SonarQube describes grammars in a declarative form, whereas Xtext requires grammars written in imperative Java code. Figure 1 shows the strategy for bridging the grammar formats of both tools. This strategy consists of three phases. First, a Text to Model (T2M) transformation is carried out to obtain a Xtext model from the grammar syntax of a language written with Xtext. Second, a Model to Model (M2M) process to transform the Xtext grammar model elements into those of a Java model, according to the Sonar grammar. Finally, a Model To Text (M2T) process serializes the model from the

[1] `http://docs.sonarqube.org/display/DEV/SSLR`

previous step as the syntax required by SonarQube. In Figure 2, we can observe the abstract mapping between the elements of the grammars in both tools.

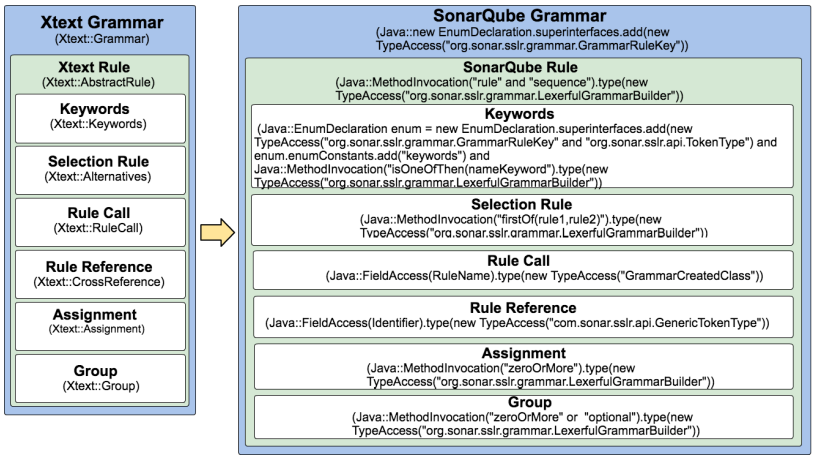

Fig. 2: Interoperability strategy

Regarding to the implementation details, the first T2M process is not necessary to be implemented from scratch, because it is already provided by Xtext. The user has to configure a specific property in the Modeling Workflow Engine (.mew) file that comes with every Xtext project. This way, when the user runs the process (represented by that workflow), an XMI version of the grammar description is automatically generated. For the sake of simplicity, the second and third transformation processes have been implemented as a single step. This prevents the need for using a model transformation engine, such as ATL, and a subsequent Java serialiser. In this vein, an Acceleo module and a set of code templates were developed to generate the Java source code artifacts required by SonarQube to recognise new languages. In addition, all the boilerplate code required by SonarQube is automatically generated. From the user's perspective, two Eclipse IDE plug-ins have been developed and made available at `https://github.com/TatyPerson/Xtext2Sonar`. These plug-ins extend the options available in the contextual menu associated to the .xtext grammar files, by adding an option to generate the source code of a Java project. The Java project has to be later imported into an existing SonarQube installation in order to support the new language.

The proposal described in this work can be also used for general purpose languages, as long as they have been developed with Xtext. In this way, the strategy was applied in Vary[2], an environment for writing and running pseudocode algorithms. A snippet of the Xtext grammar of the Vary language and its equivalent

[2] `http://tatyperson.github.io/Vary/`

for SonarQube, along with their relationships, is shown in Figure 3. Analyzing pseudocode algorithms is possible in this way with SonarQube, which provides with the basic metrics (e.g. lines of code, percentage of commented code, etc.) and quality checks according to several guideline rules, such as the abuse of global variables, the excessive number of lines of code, or whether the program subroutines are well-documented, among others.

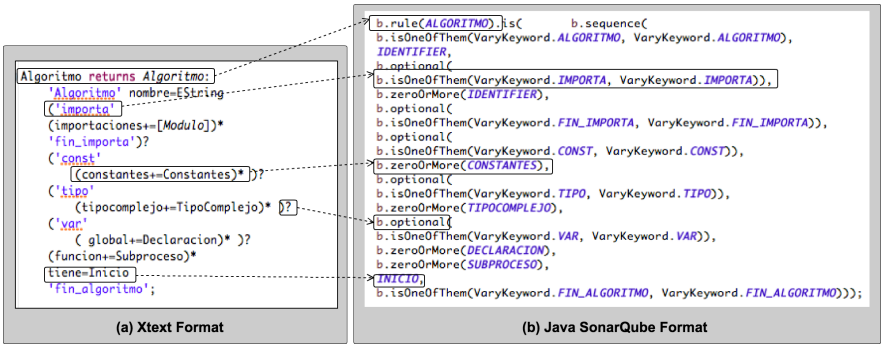

Fig. 3: Grammar code snippets

3 Conclusions

This research intends to improve the usability of the textual DSLs, by offering a tool to automatically build language recognisers for source code quality analysis platforms. The development of the tool is based on a model driven interoperability strategy that transforms Xtext grammar files into Java plug-ins for SonarQube. As a future research, a wider evaluation of this proposal on existing DSLs will be performed.

References

1. Bettini, L.: Implementing Domain-Specific Languages with Xtext and Xtend. Packt Publishing Ltd (2013)
2. Brambilla, M., Cabot, J., Wimmer, M.: Model-driven software engineering in practice. Synthesis Lectures on Software Engineering 1(1), 1–182 (2012)
3. Fowler, M.: Domain-specific languages. Pearson Education (2010)
4. Hermans, F., Pinzger, M., Deursen, A.: Domain-specific languages in practice: A user study on the success factors. In: Model Driven Engineering Languages and Systems: 12th International Conference. pp. 423–437. Springer, Berlin (2009)
5. Parr, T., Fisher, K.: LL (*): the foundation of the antlr parser generator. In: ACM SIGPLAN Notices. vol. 46, pp. 425–436. ACM (2011)