ANTLR4EMF
Project proposal

Harald A. Weiner
JKU
Linz, Austria
Email: harald.weiner@jku.at

I. Introduction

Software developers and engineers often have to deal with modernization of legacy code as part of the maintenance life cycle phase for software products. MDE (Model-driven engineering) promises to support software builders with approaches, models, tools and processes to simplify this task by using automation. The scientific term for migration of software with the help of MDE is called "Model-Driven Software Modernization" (MDSM) [1]. When the source of an application should be converted from one programming language to another, e.g. from Delphi to Java, it is first necessary to import the existing code from text files into a model which has a meta-model of the input programming language. This first step is called Text-to-Model (T2M). There exist domain-specific language (DSL) tools which can be used or extended for simpler general-purpose programming languages. But when it comes to dealing with complex grammars, these tools reach their limits. In [2], the authors discuss the existing problems, evaluate various existing approaches and propose a new language to bridge the gap between grammar-ware and Model-Driven Development (MDD).

But why invent another language. Is it really necessary? With the advent of compiler-generators, Yacc/Bison [3], JavaCC, Coco/R [4] and ANTLR [5] for example, it has gotten easier then ever to describe grammars in an explicit, programmatic form. Collections of grammar files for different programming languages, like [6] and [7], have arisen. Wouldn't it be nice if they could just be used, as they are, for T2M projects?

II. Existing T2M software - the good, the bad and the ugly

Before exploring the requirements of this project, it will be a good idea to take a deeper look on the existing software with its advantages and problems. This will make the project scope and the requirements of this proposal much clearer. As the Eclipse Modeling Framework (EMF) with its Ecore meta-model is a de-facto standard, only Eclipse plug-in projects will be described in this section.

A. EMFText

"EMFText is a tool for defining textual syntax for Ecore-based metamodels" [8]. It can be used for creating DSLs or parsing of GPLs, like Java in the Jamopp project [9] [10]. EMFText uses ANTLR in version three under the hood. Although it is built on top of it, ANTLR grammars can not be used one-to-one. Instead, a dedicated DSL is used for the mapping of grammar elements to Ecore elements. It does not generate any meta-model on itself but requires the users to provide it with one by themselves.

EMFText does not support ANTLR version 4 yet [11].

B. Xtext

Xtext [12] is another project built on top of ANTLR version three. It uses its own DSL for parsing of grammar elements and mapping them to meta-model elements. One advantage of Xtext compared to EMFText is that it can generate an Ecore model automatically based on these DSL description files. But it is also possible to re-use an existing meta-model and just specify the parsing structure and the mapping. Xtext uses a dialect of ANTLR 3 but does not support all ANTLR3 language features (e.g., semantic predicates, which are needed by C/C++ in ANTLR3, are not available).

Xtext does not support ANTLR version 4 yet [13] and it is questionable if it will ever support version four. Xtext also does not use the latest release of the version three branch, ANTLR 3.5, either.

C. Summary

To summarize, both tools do not support ANTLR version 4. The previous branch API version three is now not maintained anymore and does not receive any updates upstream. Version 4 has the advantage that it uses another parsing algorithm and, therefore, does not require syntactic or semantic predicates anymore [14] [15] [16] (which are necessary for parsing of GPLs like C/C++ otherwise). ANTLRv4 has the advantages that, first, there exists an awesome collection of grammar specifications for this version already at [7] on GitHub which seems to have an active user and developer community. And, second, ANTLRv4 does provide a better separation of concerns between grammar specifications and processing of parse-trees by automatically generating visitor and listeners classes.

III. Requirements

All requirements for the "ANTLR4EMF" project are described in this section. Each of them is rated based on the estimated effort and on the importance.

A. Basics

ANTLR4EMF should be implemented as an Eclipse plug-in which uses the Eclipse Modeling framework and exactly one stable library version of ANTLR version 4 (the latest stable release?). It should be possible to use any ANTLR version 4 grammar file and provide this as the only required input to ANTLR4EMF. ANTLR4EMF will then generate a meta-model, a gen-model, a lexer/parser and a visitor/listener for mapping parsed grammar elements to meta-model elements automatically. So it should be pretty much like Xtext, but without using a dedicated DSL file.

1) Inputs:

- .g4 - ANTLRv4 grammar specification file

2) Expected outputs:

- .xmi - Ecore meta-model
- .genmodel - Ecore gen-model for Ecore meta-model
- A Java-program that expects a file, which confirms to the ANTLRv4 grammar specification, as its input and writes an EMF model (.xmi file - file name as another argument?) to a file which confirms to the Ecore meta-model.

3) WP meta-model generation:

- For each Non-Terminal-Node a corresponding EClass should be generated.
- For each TerminalNode a corresponding EString attribute in an EClass should be generated.
- Cardinality conversion should be easy:
  - (no cardinality) turns into lower-bound 1 and upper-bound 1
  - ? turns into 0 ... 1
  - + turns into 1 ... -1
  - * turns into 0 ... -1
- There are several pitfalls/problems with this task:
  - The visitor/listener which is generated by ANTLR automatically does not store any information about the index of a TerminalNode. E.g., when we have *A: KW_PUBLIC KW_STATIC — KW_STATIC KW_PUBLIC* we can not distinguish between these two versions without going down to the tokens layer (and compare the node positions in the TokenStream).
  - The visitor/listener which is generated by ANTLR automatically does not store any information about indexes of combined nodes. E.g., when we have *A: ((B C)+ — (B D)+)+* and we have a sequence like *BC BD BC* we only get lists of Bs, lists of Cs and lists of Ds but we can not re-construct in which order the Cs and Ds occurred without going down to the tokens layer (and compare the node positions in the TokenStream).

4) Effort: The effort may vary for the different work-packages. Some might be easier and some more difficult. But in summary, it should be average programming work with a little bit thinking required.

5) Importance: This requirement with its corresponding work-packages is a must-have. Without it the whole project is useless.

B. Recursion-to-Non-recursion conversion

It might be interesting to be able to convert statements of the form *A: A B — A C* into *A: (B — C)+* automatically. If this is a conversion of the .g4 files itself or happens at a later point during parsing does not matter.

1) Inputs:

- .g4 - ANTLRv4 grammar specification file (which might contain recursive rules)

2) Expected outputs:

- .g4 - ANTLRv4 grammar specification file (without recursive rules)

3) Effort: This might be tricky to develop an algorithm but should be easy to implement once such an algorithm is found. Maybe there exists already an implementation / a solution.

4) Importance: This is considered optional. But of course it would be nice for real-world examples, as most grammars are written in a recursive way.

C. Model-walker

As grammar rules for a programming language are represented as a tree, it is possible to walk through these elements. It would be nice if ANTLR4EMF also could generate a walker for the EMF model (which is based on the generated Ecore meta-model).

1) Inputs:

- .g4 - ANTLRv4 grammar specification file

2) Expected outputs:

- Tree-walker and visitor/listener template interfaces and classes for visiting EMF meta-model elements.

3) Effort: This might be easy to implement.

4) Importance: This is considered optional. But of course it would be nice for real-world examples, as it improves a developers life a lot.

REFERENCES

[1] K. Kowalczyk and A. Kwiecinska, "Model-driven software modernization," Master's thesis, , School of Computing, 2009.

[2] J. L. C. Izquierdo and J. G. Molina, "Extracting models from source code in software modernization," *Software & Systems Modeling*, vol. 13, no. 2, pp. 713–734, 2014.

[3] "Gnu bison website," https://www.gnu.org/software/bison/, last visited: 2016-05-09. [Online]. Available: https://www.gnu.org/software/bison/

[4] "Coco/r," http://ssw.jku.at/Research/Projects/Coco/, last visited: 2016-05-09. [Online]. Available: http://ssw.jku.at/Research/Projects/Coco/

[5] T. Parr, "Antlr website," http://www.antlr.org/, 2016, last visited: 2016-05-11. [Online]. Available: http://www.antlr.org/

[6] "Grammar zoo," https://github.com/grammarware/slps/tree/master/topics/grammars, last visited: 2016-05-09. [Online]. Available: https://github.com/grammarware/slps/tree/master/topics/grammars

[7] T. Parr, "antlr/grammars-v4," https://github.com/antlr/grammars-v4/, 2015. [Online]. Available: https://github.com/antlr/grammars-v4/

[8] F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende, "Model-based language engineering with emftext," in *International Summer School on Generative and Transformational Techniques in Software Engineering*. Springer, 2011, pp. 322–345.

[9] F. Heidenreich, J. Johannes, M. Seifert, and C. Wende, "Closing the gap between modelling and java," in *International Conference on Software Language Engineering*. Springer, 2009, pp. 374–383.

[10] F. Heidenreich, J. Johannes, J. Reimann, M. Seifert, C. Wende, C. Werner, C. Wilke, and U. Assmann, "Model-driven modernisation of java programs with jamopp," in *Joint Proceedings of the First International Workshop on Model-Driven Software Migration, MDSM*, 2011, pp. 8–11.

[11] H. A. W. Lars Schtze, Mirko Seifert, "Use antlr 4.0 - issue 17," https://github.com/DevBoost/EMFText/issues/17, 2016, last visited: 2016-05-11. [Online]. Available: https://github.com/DevBoost/EMFText/issues/17

[12] L. Bettini, *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing Ltd, 2013.

[13] H. A. W. Oleg Bolshakov Sven Efftinge, "Xtext and antlr4?(still no plans to use newer antlr?)," https://www.eclipse.org/forums/index.php/t/640630/, 2016, last visited: 2016-05-11. [Online]. Available: https://www.eclipse.org/forums/index.php/t/640630/

[14] T. Parr, *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013.

[15] T. J. Parr and R. W. Quong, "Adding semantic and syntactic predicates to ll (k): pred-ll (k)," in *Compiler Construction*. Springer, 1994, pp. 263–277.

[16] T. Parr and K. Fisher, "Ll (*): the foundation of the antlr parser generator," *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 425–436, 2011.