The Language Evolver Lever — Tool Demonstration —

Elmar Juergens [1,2] Markus Pizka [1]

Institut für Informatik, Technische Universität München
Boltzmanstr. 3, 85748 Garching, Germany

Abstract

Since many domains are constantly evolving, the associated domain specific languages (DSL) inevitably have to evolve too, to retain their value. But the evolution of a DSL can be very expensive, since existing words of the language (i. e. programs) and tools have to be adapted according to the changes of the DSL itself. In such cases, these costs seriously limit the adoption of DSLs.

This paper presents Lever, a tool for the evolutionary development of DSLs. Lever aims at making evolutionary changes to a DSL much cheaper by automating the adaptation of the DSL parser as well as existing words and providing additional support for the correct adaptation of existing tools (e. g. program generators). This way, Lever simplifies DSL maintenance and paves the ground for bottom-up DSL development.

Key words: domain specific languages, bottom- up language development, language evolution, coupled transformation

1 Introduction

Just as other software artifacts, languages need to evolve as the environments in which they are employed change.

This is especially apparent for domain specific languages (DSLs), since they are usually tightly bound to a domain.

Whenever its domain evolves, a DSL must be adapted in order to reflect these changes. Evolving a DSL requires three main steps:

- evolution of the language syntax
- migration of existing words (i. e. programs) to conform to the new grammar

[1] Email: {juergens, pizka}@in.tum.de

[2] Thanks to Jurgen Vinju for plentiful advice on SDF/SGLR, Ralf Lämmel for valuable discussion on Grammar Adaptation and Jonathan Streit for helpful comments on this paper.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

- adaptation of language processing tools (i. e. parser, generator)

In traditional approaches to implement DSLs, the evolution capabilities are limited, since all three evolution steps usually have to be performed manually. Transformers that migrate existing words must be written and parsers and generators must be adapted by hand. Costs for doing this in an ad-hoc manner every time a DSL evolves are high and thus seriously inhibit evolution.

This paper explains our solution to the evolutionary development of DSLs which we call Lever (Language Evolver). Lever provides itself a domain specific language for DSL creation and evolution. It automates the adaptation of a DSL's syntax, parser and existing words. Furthermore, it supports the manual adaptation of the DSL generator by indicating those generator parts that are affected by the language evolution operations performed.

Related Work

A tool having a strong relation to Lever is TransformGen [1,5]. It simplifies the migration of existing words, but provides only limited support for coupled evolution and does not support the adaptation of parsers or generators.

The grammar evolution part of Lever was inspired by the work of Ralf Lämmel on Grammar Adaptation [3,4] but heads into a different direction by considering coupled evolution operations on words and their grammars.

2 Overview of Language Evolution with Lever

2.1 Grammar Evolution

Lever uses labeled context free grammars for the specification of the syntax of a language. A labeled context free grammar extends canonical context free grammars with unique labels for productions and production symbols. These labels are later on used in path expressions that navigate through labeled context free grammars to select grammar elements.

Grammars in Lever are mutable. Lever provides a Grammar Evolution Language that is used to create and modify grammar elements. The Grammar Evolution Language comprises a set of evolution operations that is complete in the sense that every grammar can be turned into any other grammar by applying a sequence of Grammar Evolution Language statements.

2.2 Word Evolution

Lever internally represents words as labeled derivation trees: The production labels name the nodes and the production symbol labels name the edges in the tree. Thus, the same path expressions that select grammar elements from the labeled context free grammar can be used to select corresponding nodes from the labeled derivation tree. This turns path expressions into a uniform

querying mechanism for both labeled context free grammars and derivation trees.

Labeled derivation trees are also mutable. Dual to the Grammar Evolution Language, Lever provides a Word Evolution Language that is used to perform evolution operations on the derivation trees. The Word Evolution Language comprises a set of evolution operations that is complete in the sense that every derivation tree can be turned into any other derivation tree by applying a sequence of Word Evolution Language statements.

2.3 Coupled Evolution of Grammar and Words

While the grammar and word evolution languages are expressive, their level of abstraction is still relatively low, since they target grammar and word evolution separately.

Various frequently used evolution operations can be done more comfortably using higher level coupled evolution operations that are automatically mapped onto corresponding grammar and word evolution operations. Examples for such higher level commands are renaming of terminals or the introduction of new nonterminals with a default value. Lever provides an integrated Language Evolution Language to facilitate such coupled evolution operations. It builds on the Grammar- and Word Evolution Languages to implement these coupled evolution commands. When working with Lever, users mainly employ the Language Evolution Language. Only in cases it does not cover, elementary grammar and word evolution operations are used.

The Language Evolution Language is extensible, allowing users to add their own coupled evolution commands. This way we hope to gradually grow it until it provides all commonly encountered language evolution operations.

It is interesting to notice that the Language Evolution Language itself is a DSL that is being developed in a bottom-up, stepwise manner and could thus be implemented using Lever. However, the Language Evolution Language is currently realized as an internal DSL, since the Grammar- and Word Evolution Languages are still evolving, as our understanding of grammar and tree transformations changes. It is planned to implement the Language Evolution Language using Lever, as soon as the Grammar- and Word Evolution Languages reach a sufficient level of stability.

2.4 Adaptation of Language Processing Tools

Lever can automatically produce parsers for its languages. It generates SDF grammars [2] from labeled context free grammars and uses the SGLR parser [6] to instantiate labeled derivation trees from words of the language. Adaptation of the parser is thus completely automated. [3]

[3] Note that Lever does not depend on GLR parsing techniques. If their use is not desired, they can be replaced by hand-written parsers. However, parser adaptation then cannot be

Language processing tools (i. e. generators) use path expressions to access nodes in labeled derivation trees. Lever does not automate the adaptation of these path expressions after language evolution, yet. But path expressions are grammar- aware: Lever validates path expressions statically against the grammar to detect those expressions that would fail or only produce empty result sets when evaluated on labeled derivation trees. This static checking detects all path expressions that broke during language evolution.

3 Demonstration

We demonstrate an exemplary evolution step to illustrate the stepwise development of a simple DSL to generate data structures. The initial grammar is displayed textually [4] and visually [5] in Figures 1 and 2a. It contains two parameters that influence code generation: The *type* describes allowed data objects and when *unique* is present, there may be no two equal objects contained in store instances. Figures 3, and 2b show words for the initial grammar (both textually and visually). [6]

```
"Store" "[" lbl:"type=" type:"[a-z]+" "unique"? "]" -> Datastructure {Store}
```

Fig. 1. Initial Grammar in textual form

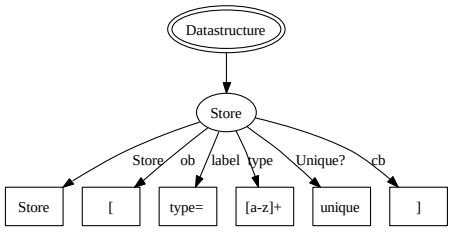
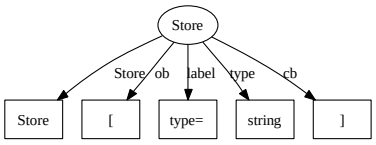


Fig. 2. a) Initial Grammar



b) Initial Word

```
Store [type=string]
Store [type=object unique]
```

```
Bag [type=string]
Set [type=object]
```

Fig. 3. Words before and after evolution

As our understanding of the domain of data structures grows, we decide to replace the *unique* keyword with the terms *Set* and *Bag*. To reflect our changed understanding of the domain, we evolve our DSL accordingly:

- All unique Stores are to be converted to Sets, all other instances to Bags.
- Both data structures contain the type parameter. To avoid duplication in the resulting grammar, we encapsulate it into a production of its own.

automated anymore.

[4] In this example, some of the labels of literal symbols have been omitted for brevity.

[5] Key: double ellipses are sorts, single ellipses are productions and boxes are terminals.

[6] Key: ellipses are nodes corresponding to productions, boxes are leafs with word fragments.

- The Store production is now unused and gets removed from the grammar.

Figures 3, 4, 5a and 5b display grammar and words after evolution.

```
"Set" ob:"[" Type cb:"]" -> Datastructure {Set}
"Bag" ob:"[" Type cb:"]" -> Datastructure {Bag}
lbl:"type=" type:"[a-z]+" -> Type {Type}
```

Fig. 4. Textual representation of the evolved grammar

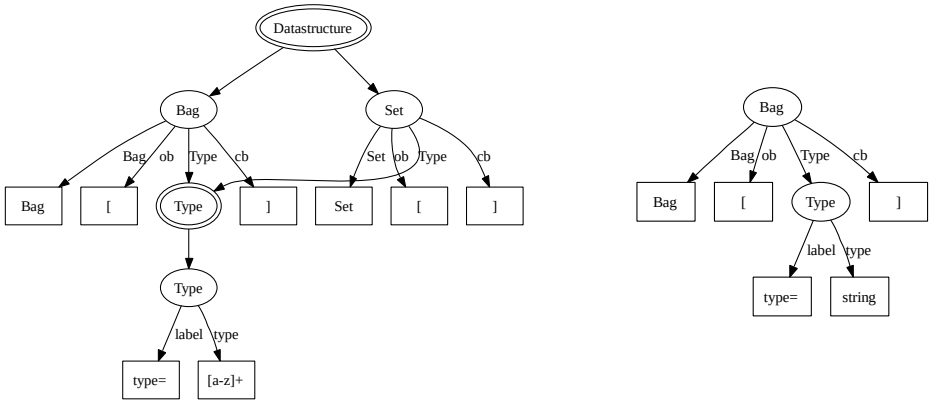


Fig. 5. a) Evolved grammar b) Evolved word

Figure 6 shows the Language Evolution Language commands for these evolution operations. [7]

```
encapsulate "lbl", "type" into Type in Store

create production "Set" ob:"[" Type cb"]"       ->      Datastructure {Set}
create production "Bag" ob:"[" Type cb"]"       ->      Datastructure {Bag}

for store in Stores:
       remove literal "Store"
       if contains literal unique:
             set production to "Datastructures.Set"
             append leaf "Set"
       else:
             set production to "Datastructures.Bag"
             append leaf "Bag"

delete production "Store"
```

Fig. 6. Language Evolution Language statements

[7] The syntax of the statements has been simplified to increase readability.

4 Conclusion

Lever provides several DSLs for different levels of language evolution: the Grammar Evolution Language for grammars, the Word Evolution Language for words (i. e. programs) and the Language Evolution Language for the coupled evolution of grammar and words. Evolution operations formulated using these DSLs allow Lever to automate the adaptation of existing words and parsers. Furthermore, Lever can point out areas that need manual adaptation in tools that do not get adapted automatically (e. g. generators). Compared to ad hoc approaches to DSL evolution, Lever thus significantly decreases evolution costs.

Future work includes the application of Lever to the development of real world DSLs to grow the Language Evolution Language and thus increase its expressiveness. Additionally, we plan to automatically adapt path expressions for those Language Evolution Language commands that merely refactor a language (i. e. renaming of nonterminals, encapsulating or inlining nonterminals,...).

Lever is currently being implemented and tested and will be made available in the first half of 2006.

References

[1] Garlan, D., C. W. Krueger and B. S. Lerner, *Transformgen: automating the maintenance of structure-oriented environments*, ACM Trans. Program. Lang. Syst. **16** (1994), pp. 727–774.

[2] Heering, J., P. R. H. Hendriks, P. Klint and J. Rekers, *The syntax definition formalism sdf reference manual*, SIGPLAN Not. **24** (1989), pp. 43–75.

[3] Lämmel, R., *Grammar Adaptation*, in: *Proc. Formal Methods Europe (FME) 2001*, LNCS **2021** (2001), pp. 550–570.

[4] Lämmel, R. and G. Wachsmuth, *Transformation of SDF syntax definitions in the ASF+SDF Meta-Environment*, in: M. van den Brand and D. Parigot, editors, *Proceedings of the First Workshop on Language Descriptions, Tools and Applications (LDTA'01), Genova, Italy, April 7, 2001, Satellite event of ETAPS'2001*, ENTCS **44** (2001).

[5] Staudt, B. J., C. W. Krueger and D. Garlan, *A structural approach to the maintenance of structure-oriented environments*, in: *SDE 2: Proceedings of the second ACM SIGSOFT/SIGPLAN software engineering symposium on Practical software development environments* (1987), pp. 160–170.

[6] van den Brand, M. G. J., J. Scheerder, J. J. Vinju and E. Visser, *Disambiguation filters for scannerless generalized lr parsers*, in: *CC '02: Proceedings of the 11th International Conference on Compiler Construction* (2002), pp. 143–158.