Grammar-Driven Generation of Domain-Specific Language Testing Tools

Hui Wu, Jeff Gray
University of Alabama at Birmingham
{wuh, gray}@cis.uab.edu

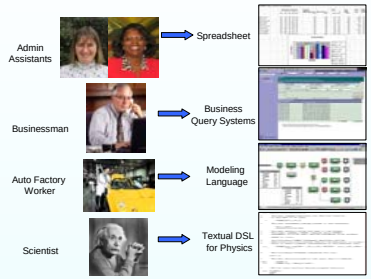
Marjan Mernik
University of Maribor, Slovenia
marjan.mernik@uni-mb.si

This research is supported by an IBM Eclipse Innovation Grant (EIG).

http://www.cis.uab.edu/wuh/DDF

The goal of this research is to build a DSL tool generation framework to automatically generate the testing tools from DSL grammars, which assist in debugging, testing, and profiling a program written in a DSL for end-users.

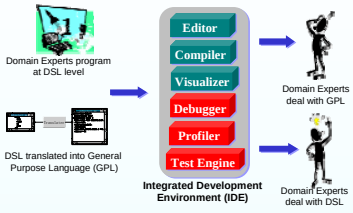
Categories of End-User Programmers



Problem Statements

1. Computer errors cost economy billions of dollars each year: $60 billion per year in US [7]
2. End-user developers (e.g., scientist, accountant, and statistician) are increasing in number: 11 to 55 million End-user programmers compared to 2.75 million professional programmers in US [6]
3. Manual construction of the testing tools for each new DSL can be time-consuming, expensive, and problematic.
4. The mismatch of abstraction levels between DSL and GPL forces the end-user to understand the translated code in the GPL, rather than the higher-level description contained in the DSL [1]; e.g., debugging code from a parser generator.
5. The dangers of end-user programming [3]: End-users often lack knowledge of software development principles; Inadequate testing and debugging processes; No testing tools support for end-users (especially for DSL users)

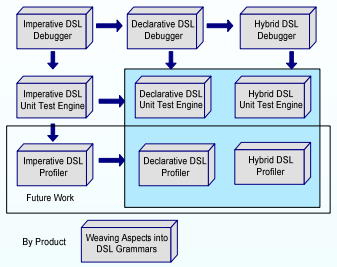
Research Motivation



Although techniques for constructing language tools (e.g., editor and compiler) have been developed over the years, support for debuggers and test engines for DSLs have not been investigated deeply.

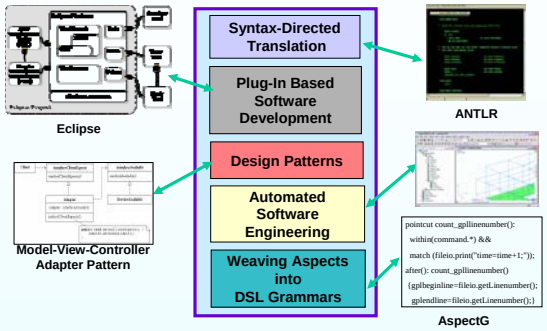
The most popular technique for implementing a DSL is to translate the DSL into a GPL [5].

Research Goal



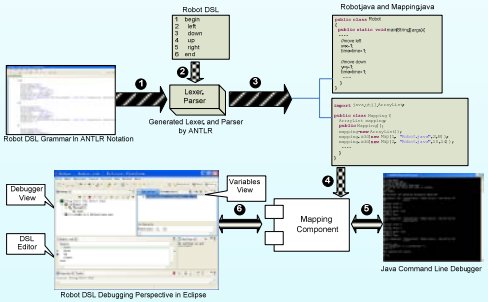
The research goal is to investigate a generalized method that will enable construction of a matrix of DSL inspection tools as a type of software factory [2].

Approaches and Methods used in This Research



Architecture Overview of DSL Debugger Framework and Debugger Generation Processes

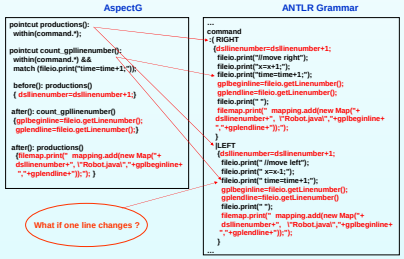
1. A DSL grammar is specified in ANTLR notation and the lexer and parser is generated by ANTLR (step 1)
2. The generated Lexer and Parser takes the Robot DSL as input (step 2).
3. ANTLR not only translates the Robot DSL into the corresponding Robot.java, but also generates the Mapping.java file (step 3).
4. The mapping component interacts and bridges the differences between the Eclipse debugger platform and the command line java debugger (step 4).
5. There are two round-trip mappings and message passing processes involved (step 5 and step 6) between the Robot DSL debugging perspective [8] in Eclipse and jdb.



Mapping and Knowledge Base Methods

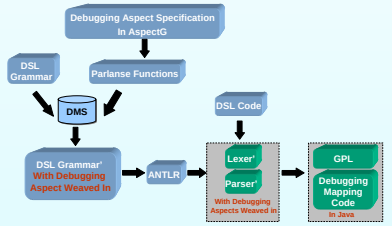
1. Source Code Mapping:
   - One line of DSL code maps to a segment of generated GPL codes.
2. Data Structure Mapping:
   - One Data structure in DSL maps to different Data Structures in GPL
3. Testing and Debugging Methods Mapping Knowledge Base
   - One unit test case in DSL maps to several unit test cases in GPL (Unit Test Mapping Algorithm)
   - One debugging command maps to a series of GPL debugging commands (Debugging Mapping Algorithms)
4. Output messages map back to the DSL level through the wrapper interface

AspectG Pointcut Model



AspectG is an aspect language for grammar specification. The AspectG pointcut model can specify a pointcut at the syntax level (e.g., production) and also at the semantic level (e.g., count_gpllinenumber)

Weaving Aspects into DSL Grammars

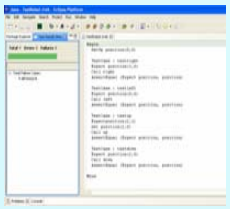


The key contribution of this approach is the transformation of the grammar itself. The specification of the debug mapping is modularized in a single place – the DMS transformation function. The approach has the side benefit of language independence. It does not matter which GPL serves as the generated target language [9].

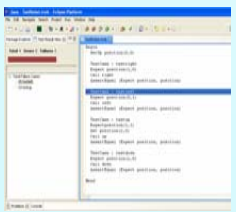
Expected Contributions

1. Provide a DSL language tool generation framework that will synthesize tools (e.g., debugger and unit testing engine) automatically from DSL grammar specifications. For different DSLs, the modification of certain components of the framework may be considered necessary, but the architecture and mapping algorithms of the framework are generic.
2. A new technique for building a set of language tools for DSLs, especially for debuggers and testing engines. Reusing the existing GPL language tools and adapting the interfaces of the new IDE will become a future trend of tool generation. The questions to be answered are: "What features are needed for each language testing tool?"; "What is the minimum information about a DSL that is needed to generate a language tool from the DSL grammar?"
3. Application of the technique for better separation of concerns in Grammarware, which comprises grammars and all grammar-dependent software (e.g., lexer, parser) [4]

Research Results: Generating DSL Unit Test Engine

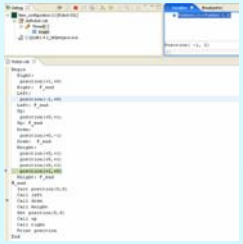


Imperative DSL Unit Test Engine for Robot Language
All four test cases passed

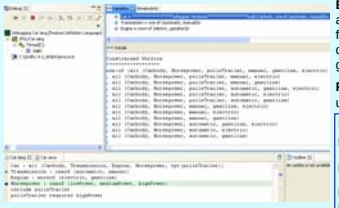


Imperative DSL Unit Test Engine for Robot Language
Two test cased failed (testleft and testup)

Research Results: Generating DSL debuggers



Imperative DSL Debugger for Robot Language



Declarative DSL Debugger for Car Feature Definition Language

Research Summary

Evaluation: Various experimental validation efforts are underway to test this framework's applicability, scalability, and reliability. In order to provide such assessment, DSLs obtained from industry and research collaborators will drive the evaluation. The ability of the generated debugger and test engine to detect errors in the DSL is also an assessment criterion. Complex grammars will serve as test cases to determine the benefits of grammar weaving.

Future Work: A future work will also investigate the concept of profiling, as applied to DSLs, using a framework similar to DDF and DUTF.

Key References:

[1] R. E. Faith, "Debugging Programs After Structure-Changing Transformation," Doctoral Dissertation, Department of Computer Science, University of North Carolina at Chapel Hill, 1998.

[2] J. Greenfield and K. Short, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley Publishing, Inc, Indianapolis, IN, 2004.

[3] W. Harrison, "The Dangers of End-User Programming," *IEEE Software*, vol. 21, no. 4, pp. 5-7, July/August 2005.

[4] P. Klint, R. Lammel, and C. Verhoef, "Towards an Engineering Discipline for Grammarware," *ACM Trans. on Software Engineering and Methodology*, vol. 14, no. 3, pp. 331-380, July 2005.

[5] M. Mernik, J. Heering, and A. M. Sloane, "When and How to Develop Domain-Specific Languages," CWI Technical Report, SEN-E0309, 2003.

[6] C. Scaffidi, M. Shaw, and B. Myers. Estimating the numbers of end users and end user programmers. *In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, Dallas, TX, September 2005.

[7] G. Tassey and RTI, "The Economic Impacts of Inadequate Infrastructure for Software Testing," Final Report, available at www.nist.gov/director/prog-ofc/report02-3.pdf, May 2002.

[8] D. Wright and B. Freeman-Benson, "How to Write an Eclipse Debugger," Eclipse Corner, Fall 2004,

[9] H. Wu, J. Gray, S. Roychoudhury, and M. Mernik, "Weaving a Debugging Aspect into Domain-Specific Language Grammars," ACM Symposium for Applied Computing (SAC) – PSC Track, Santa Fe, NM, pp. 1370-1374, March 2005.