Demonstration of a Domain-Specific Language Debugging Framework

Hui Wu [1] Jeff Gray [2]

Department of Computer and Information Sciences
The University of Alabama at Birmingham
Birmingham AL, USA 35294-1170

Marjan Mernik [3]

Faculty of Electrical Engineering and Computer Science
University of Maribor
2000 Maribor, Slovenia

Abstract

This demonstration presents a debugging framework that targets domain-specific languages (DSLs). The DSL Debugging Framework (DDF) generates debuggers automatically from DSL grammar specifications. The debugging concern is weaved into a base grammar in an aspect-oriented style. DDF was used to generate debuggers for three different types of DSLs (imperative, declarative, and hybrid) that are translated into a General Purpose Language (GPL), such as Java. The demonstration will show how debugging concerns are weaved into the base grammar of a hybrid DSL to generate a debugger. The generated debugger will be demonstrated on a DSL for describing GUIs.

Key words: AOP, Debugging, DSL, Grammars

1 Introduction

A DSL is a small language that is more concise and closer to a specific problem domain than a traditional programming language. A key benefit of using a DSL is the isolation of accidental complexities typically required in the implementation phase (i.e., the solution space) such that a programmer can focus on the key abstractions of the problem space. The development of debuggers for each individual DSL is time consuming and error-prone.

[1] Email:wuh@cis.uab.edu
[2] Email:gray@cis.uab.edu
[3] Email:marjan.mernik@uni-mb.si

Creating a pre-processor that translates the DSL source into a GPL target is a common DSL implementation pattern [3]. However, this implementation approach results in a mismatch of abstraction levels between the DSL and generated GPL, which forces programmers to deal with the translated GPL code, rather than a higher-level description contained in the DSL. To bridge the language abstraction gap we developed DDF, which is based on the Eclipse SDK debug platform.

In this demonstration, we will show an aspect-oriented approach that weaves debugging concerns into a DSL grammar. The generated debugger will be used to step through the execution of a DSL for building a GUI that is embedded within a Java program.

2 Domain-Specific Language Debugging Framework

A DSL debugger is a useful testing tool to assist domain experts in finding software errors in DSL programs. However, it is usually difficult to develop a debugger for every DSL from scratch because each language debugger depends heavily on the underlying platform [5]. A key technique of the DDF is a mapping process that records the correspondence between the DSL and the generated GPL. The DDF uses a generative approach that automates the reuse of a GPL debugger as a base machine that is translated back into the virtual layer of the DSL.

The DDF is an integrated debugging environment that is composed of a DSL grammar weaver and a set of Eclipse plug-ins that provide DSL debugging support using in a traditional debugger user interface (e.g., step into/over). In order to generate the debugger, additional debugging mapping information is needed, which depends on both the source language (DSL) and the target language (GPL). The debugging mapping aspects are weaved into the DSL grammar to generate the mapping information used to integrate with the host GPL debugger (e.g., the stand-alone command line Java debugger - jdb, and the GNU project debugger - gdb). The generated mapping code re-interprets the DSL program and the program's debugger states into a sequence of debugging commands that query the GPL debugger server. The debugging responses from the GPL debugger server are mapped back into the DSL debugging perspective [6], which is an extension of the Eclipse debugging platform plug-in. Thus, the end-user performs debugging actions at the level of abstraction specified by the DSL, not at the lower-level abstraction provided by the GPL. Furthermore, the values displayed in the DDF variable view are presented in the context of the DSL, not the GPL.

3 Case Study

The demonstration will use the Swing User Interface language (SWUL) as a case study (http://www.program-transformation.org/Stratego/Java-Swul).

SWUL is a DSL that is embedded in a Java program to assist in constructing Swing user interfaces in a more comprehensive and structured manner. Ideally, programmers should be able to debug the SWUL code between two different language notations (e.g., Java and SWUL).

3.1 A Debugging Aspect

The first half of the demonstration will show the process of weaving a debugging concern into the SWUL grammar, which is written in ANTLR [4]. The demonstration will introduce AspectG, which is our aspect-oriented language for grammars. The demonstration will illustrate changes to the SWUL grammar that add the debugging concern. AspectG is written on top of the Design Maintenance System (DMS) [1], which is a program transformation system.

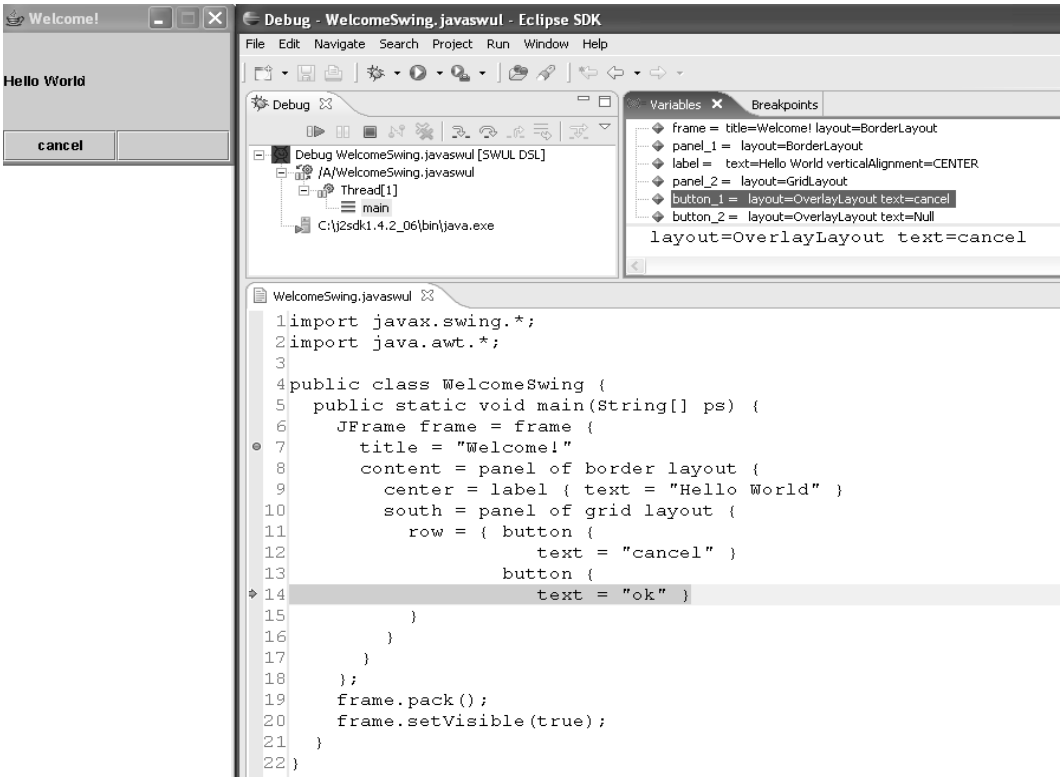

Fig. 1. Screen-shot of SWUL debugger

3.2 Debugging a SWUL Program

The second half of the demonstration will use a sample SWUL program as the debugging target. Figure 1 is a screen-shot during a debugging session of a SWUL program, which implements a small JFrame window with two buttons (e.g., “cancel” and “ok”). The lower-right part of Figure 1 is the editor, where line 6 to line 18 of the SWUL code is surrounded by Java code. A break point is set at line 7 and the current program execution pointer

is at line 14. The upper-left corner of Figure 1 is the GUI that is created throughout the debugging session, which incrementally changes its appearance corresponding to each execution step. The upper-right corner of Figure 1 contains the variable view, which displays the current SWUL variable values that are translated from the underlying Java variable values.

4 Conclusion

This demonstration will present the details of an aspect-oriented approach that automatically generates a debugger from a DSL grammar. The demonstration will showcase the capabilities of the DDF to provide a debugger that can alternate between a DSL (SWUL) and a GPL that serves as host (Java). This research makes a contribution toward language extension through grammar transformations in an aspect-oriented manner. Additional technical details about the DDF, including publications and video demonstrations, are available at the DDF project page (http://www.cis.uab.edu/wuh/ddf/). As related work, the TIDE project offers a flexible interactive debugging framework using the ASF+SDF Meta-Environment [2].

References

[1] I. Baxter, C. Pidgeon, and M. Mehlich, "DMS: Program Transformation for Practical Scalable Software Evolution," *International Conference on Software Engineering*, Edinburgh, Scotland, pp. 625-634, May 2004.

[2] M. van den Brand, B. Cornelissen, P. Olivier, and J. Vinju, "TIDE: A Generic Debugging Framework," *Tool Demonstration at Fifth Workshop on Language Descriptions, Tools, and Applications*, Edinburgh, Scotland, June 2005.

[3] M. Mernik, J. Heering, and A. Sloane, "When and How to Develop Domain-Specific Languages," *ACM Computing Surveys*, vol. 37, no. 4, pp. 316-344, December 2005.

[4] T. Parr, *The Definitive ANTLR Reference Building Domain-Specific Languages*, Pragmatic Bookshelf, 2007.

[5] J. B. Rosenberg, *How Debuggers Work-Algorithms, Data Structures, and Architecture*, John Wiley and Sons, 1996.

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