B. Updating the Debugger Extension

The *MUnit* debugger extension tries to lift for each `Testcase` a stack frame whose name is prefixed with *test_*, followed by the name of the respective `Testcase` (see Section V-D). However, due to our generator modification, this frame is not present and therefore the whole call stack construction fails with an error. To solve this problem, we update the name used for matching the stack frame name:

```
String frameName = "testcase_" + this.name;
contribute frame mapping for frames.select(name=frameName);
```

Other aspects, such as stepping, breakpoints or watches are not affected by the generator modification and hence do not need to be changed. Therefore, after fixing the call stack lifting for `Testcase` our debugger tests pass again.

XI. Related Work

Wu et al. describe a unit testing framework for DSLs [10] with focus on testing the semantics of the language. However, from our perspective, it is necessary to have testing DSLs for all aspects of the language definition, e. g., editor (concrete syntax), type system, scoping, transformation rules, and finally the debugger.[1] mbeddr contains tests for the editor, type system, scoping and transformation rules, our work contributes the language for testing the debugger aspect.

The Low Level Virtual Machine (LLVM) project [11] comes with a C debugger named Low Level Debugger (LLDB). Test cases for this debugger are written in Python and the unit test framework of Python. While those tests verify the command line interface and the scripting Application Programming Interface (API) of the debugger, they also test other functionality, such as using the help menu or changing the debugger settings. Further, some of the LLDB tests verify the debugging behavior on different platforms, such as Darwin or Linux. In contrast, we only concentrate on testing the debugging behavior, but also support writing tests for specific platforms. Our approach for testing the debugging behavior is derived from the LLDB project: write a program in the source-language (mbeddr), compile it to an executable and debug it through test cases, which verify the debugging behavior.

The GDB debugger takes a similar approach as the LLDB: tests cover different aspects of the debugger functionality and are written in a scripting language [9]. Contrarily, to our approach of testing debugging for one extensible language, the GDB project tests debugging behavior for all of its supported languages, such as C, C++, Java, Ada etc. Further, those tests run on different platforms and target configurations. Our work supports writing tests against different platforms, but does not allow users to change the target configuration via the DSL.

XII. Summary and Future Work

The mbeddr extensible language comes with an extensible debugger. To test this debugger, we have introduced in this paper a generic and extensible testing DSL. The language is implemented in MPS with focus on mbeddr, but the underlying approach is applicable for testing any imperative language debugger. Further, we have shown in this paper (1) the implementation of a language extension, (2) how debugging support is build for it and (3) how the debugger is tested with use of our DSL. The language is designed for extensibility, so others can contribute their own context-specific validation rules. In addition, we concentrated on reuse, so test data, test structures and validation rules can be shared among tests.

In the future, we plan to investigate ways for integrating the debugger specification DSL with the DSL for testing the debugger extension. From this integration we expect to (1) gain advances in validating debugger test cases and (2) the possibility to automatically generate test cases from formal debugger specifications (based on work from [12], [13] and [14]). In addition, we will continue researching on languages for testing non-functional aspects, such as testing the performance of stepping commands and lifting of program state.

[1]Specific language workbenches might require testing of additional aspects

References

[1] M. Voelter, “Language and IDE Development, Modularization and Composition with MPS,” in *Generative and Transformational Techniques in Software Engineering*, ser. Lecture Notes in Computer Science, 2011.

[2] M. Voelter, D. Ratiu, B. Schaetz, and B. Kolb, “Mbeddr: An extensible c-based programming language and ide for embedded systems,” in *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity*, ser. SPLASH ’12. New York, NY, USA: ACM, 2012, pp. 121–140.

[3] JetBrains, “Meta Programming System,” 2015. [Online]. Available: http://www.jetbrains.com/mps

[4] D. Pavletic, M. Voelter, S. A. Raza, B. Kolb, and T. Kehrer, “Extensible debugger framework for extensible languages,” in *Reliable Software Technologies - Ada-Europe 2015 - 20th Ada-Europe International Conference on Reliable Software Technologies, Madrid Spain, June 22-26, 2015, Proceedings*, ser. Lecture Notes in Computer Science, J. A. de la Puente and T. Vardanega, Eds., vol. 9111. Springer, 2015, pp. 33–49.

[5] A. Chis, T. Gîrba, and O. Nierstrasz, “The moldable debugger: A framework for developing domain-specific debuggers,” in *Software Language Engineering - 7th International Conference, SLE 2014, Västerås, Sweden, September 15-16, 2014. Proceedings*, 2014, pp. 102–121.

[6] H. Wu, “Grammar-driven Generation of Domain-specific Language Testing Tools,” in *20th Annual ACM Special Interest Group on Programming Languages (SIGPLAN) Conference on Object-oriented Programming, Systems, Languages, and Applications*. San Diego, CA, USA: ACM, 2005, pp. 210–211.

[7] D. Pavletic and S. A. Raza, “Multi-Level Debugging for Extensible Languages,” *Softwaretechnik-Trends*, vol. 35, no. 1, 2015.

[8] B. Kolb, M. Voelter, D. Ratiu, D. Pavletic, Z. Molotnikov, K. Dummann, N. Stotz, S. Lisson, S. Eberle, T. Szabo, A. Shatalin, K. Miyamoto, and S. Kaufmann, “mbeddr.core - An extensible C,” https://github.com/mbeddr/mbeddr.core, GitHub repository, 2015.

[9] Free Software Foundation, “The GNU Project Debugger,” 2015. [Online]. Available: https://www.gnu.org/software/gdb/

[10] H. Wu, J. G. Gray, and M. Mernik, “Unit testing for domain-specific languages,” in *Domain-Specific Languages, IFIP TC 2 Working Conference, DSL 2009, Oxford, UK, July 15-17, 2009, Proceedings*, ser. Lecture Notes in Computer Science, W. M. Taha, Ed., vol. 5658. Springer, 2009, pp. 125–147.

[11] LLVM Compiler Infrastructure, “The LLDB Debugger,” 2015. [Online]. Available: http://lldb.llvm.org

[12] H. Wu and J. Gray, “Automated generation of testing tools for domain-specific languages.” in *ASE*, D. F. Redmiles, T. Ellman, and A. Zisman, Eds. ACM, 2005, pp. 436–439.

[13] P. R. Henriques, M. J. V. Pereira, M. Mernik, M. Lenic, J. Gray, and H. Wu, “Automatic generation of language-based tools using the LISA system,” *Software, IEE Proceedings* -, vol. 152, no. 2, pp. 54–69, 2005.

[14] H. Wu, J. Gray, and M. Mernik, “Grammar-driven generation of domain-specific language debuggers.” *Software: Practice and Experience*, vol. 38, no. 10, pp. 1073–1103, 2008.