Li et al. [15] present a methodology that views crosscutting features as independent modules and verifies them against CTL properties as open systems. Features consists of state machines and composition is done by connecting them via transitions specified through interfaces. The work supports the detection of undesirable feature interactions. Support for traceability and debugging is not described.

In contrast, AspectLTL aspects are defined in a symbolic and declarative manner. Our method is fundamentally different: it not only solves the possible conflicts or interferences between the specified aspects (if indeed a solution to these conflicts exists) but also produces an executable correct-by-construction implementation. If a solution does not exist, we generate a counter-implementation, annotated with the traceability information that uncovers reasons for unrealizability.

9. Conclusion and Future Work

We presented two-way traceability and conflict debugging techniques for AspectLTL and demonstrated them on a running example. To support two-way traceability, we use symbolic operations that check for intersections between the transitions that can or cannot be taken and the formulas defined in the LTL aspects. To support debugging of unrealizable specifications we reverse the roles of the system and the environment in the synthesis game, and use the winning strategy of the environment to produce a counter-implementation, that is, an interactive program, whose runs show exactly how any generated system can be forced by an (adverse) environment to violate the specifications. We combine traceability and debugging to point at the aspects to blame. The techniques provide important support for the development of systems using AspectLTL, making its use more accessible and informative. The ideas are implemented in the AspectLTL plug-in, available from [2].

One future work direction deals with the computation of an unrealizable core. Given an unrealizable AspectLTL specification, an unrealizable core is a minimal unrealizable subset of the specification. An unrealizable core is useful in debugging, as it better identifies and isolates the causes of failures and enables the generation of smaller counter-implementations. Some recent works have considered the computation of unrealizable cores in the context of LTL (GR(1)) synthesis (see, e.g., [5]). However, computing an unrealizable core for AspectLTL specifications is particularly challenging due to the non-monotonic nature of the language: each aspect may not only restrict the possible behaviors (in its `LTLSPEC` sections) but also add new behaviors (in its `TRANS` sections). Another future work is to investigate how our approach to traceability and debugging can be applied to other aspect languages, e.g., AspectJ, using abstractions similar to the ones of [7, 14], or more generally, to other feature composition frameworks (e.g., [8, 9, 15]), where, we believe, similar two-way traceability and conflict debugging support could be very useful.

References

[1] AspectJ Development Tools. http://www.eclipse.org/ajdt/.

[2] AspectLTL website. http://aspectltl.ysaar.net/.

[3] W. D. Borger, B. Lagaisse, and W. Joosen. A generic and reflective debugging architecture to support runtime visibility and traceability of aspects. In *AOSD*, pages 173–184, 2009.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Inf. Comput.*, 98(2):142–170, 1992.

[5] A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev. Diagnostic information for realizability. In *VMCAI*, 2008.

[6] A. P. Felty and K. S. Namjoshi. Feature specification and automated conflict detection. *ACM Trans. Softw. Eng. Methodol.*, 12(1):3–27, 2003.

[7] M. Goldman, E. Katz, and S. Katz. MAVEN: modular aspect verification and interference analysis. *Formal Methods in System Design*, 37(1):61–92, 2010.

[8] J. D. Hay and J. M. Atlee. Composing features and resolving interactions. In *SIGSOFT FSE*, pages 110–119, 2000.

[9] M. Jackson and P. Zave. Distributed feature composition: A virtual architecture for telecommunications services. *IEEE Trans. Software Eng.*, 24(10):831–847, 1998.

[10] E. Katz and S. Katz. Incremental analysis of interference among aspects. In *FOAL*, pages 29–38, 2008.

[11] S. Katz. Aspect categories and classes of temporal properties. In *T. Aspect-Oriented Softw. Dev. I*, pages 106–134, 2006.

[12] Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Inf. Comput.*, 163:203–243, 2000.

[13] R. Könighofer, G. Hofferek, and R. Bloem. Debugging formal specifications using simple counterstrategies. In *FMCAD*, pages 152–159, 2009.

[14] S. Krishnamurthi and K. Fisler. Foundations of incremental aspect model-checking. *ACM Trans. Softw. Eng. Methodol.*, 16(2), 2007.

[15] H. C. Li, S. Krishnamurthi, and K. Fisler. Verifying cross-cutting features as open systems. In *SIGSOFT FSE*, pages 89–98, 2002.

[16] Z. Manna and A. Pnueli. *The temporal logic of concurrent and reactive systems: specification*. 1992.

[17] S. Maoz and Y. Sa'ar. AspectLTL: An aspect langauge for LTL specifications. In *AOSD*, pages 19–30, 2011.

[18] N. Piterman and A. Pnueli. Faster solutions of rabin and streett games. In *LICS*, pages 275–284, 2006.

[19] N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of Reactive(1) Designs. In *VMCAI*, pages 364–380, 2006.

[20] A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57, 1977.

[21] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.

[22] A. Pnueli, Y. Sa'ar, and L. Zuck. JTLV: A framework for developing verification algorithms. In *CAV*, 2010.

[23] SMV model checker. http://www.cs.cmu.edu/~modelcheck/smv.html.