Language-Oriented Modularity through AWESOME DSALs *

Summary of Invited Talk

David H. Lorenz

Open University of Israel
Dept. of Mathematics and Computer Science,
1 University Rd., P.O.Box 808, Raanana 43107 Israel.
lorenz@openu.ac.il

Abstract

In this talk we coin the term *language-oriented modularity* to refer to the process of constructing and composing DSALs to better support aspect-oriented modularity. Language-oriented modularity strives to keep our aspect language as abstract as possible and our software code as modular as possible. While general purpose aspect-oriented languages offer low-level abstractions for modularizing a wide range of crosscutting concerns, they lack the modularity abstractions to tackle all cases of crosscutting. With language-oriented modularity, a solution to unanticipated crosscutting concerns is to construct and combine multiple DSALs to form aspect-oriented modularity of new kinds. We evaluate the AWESOME composition framework in terms of its support for language-oriented modularity.

Categories and Subject Descriptors D.2.6 [*Software Engineering*]: Programming Environments—Programmer workbench; D.3.2 [*Programming Languages*]: Language Classifications—Extensible languages.

General Terms Design, Languages.

Keywords Domain Specific Aspect Languages (DSALs), Language-Oriented Programming (LOP).

1. Language-Oriented Modularity

Language-oriented modularity puts DSALs at the center of the aspect-oriented software development process. DSALs are defined to fulfill a need for aspect-oriented modularity that is specific to the problem or to the domain. These DSALs are then used to implement the software solution, providing better *modularity* [8] in comparison to the use of a general-purpose aspect language. With language-oriented modularity, modularization occurs middle-out. One starts with defining the DSALs, proceeding with modularizing crosscutting concern using these DSALs, which is done in parallel with implementing the DSALs. Language-oriented modularity is a special case of a paradigm generally referred to as *language-oriented programming* [10].

Composition frameworks for language-oriented modularization can be evaluated by how well they adhere to the DSALs' "Bill of Rights" [7]:

DEFINITION: ("freedom of expression") The definition of DSALs should permit syntactic and semantic freedom in forming aspect-oriented modularity that is most suitable for modularizing the crosscutting problem at hand.

IMPLEMENTATION AND USE: ("economic freedom") The implementation and use of DSALs should be practical and cost-effective. Composition frameworks, like AWESOME [4], should reduce the complexity and cost of implementing and composing DSALs. Productivity tools, like debuggers [1], should also make aspect-oriented modularization with these DSALs effective.

INTEROPERABILITY: ("DSALs' freedom of association") The interoperability of DSALs must be supported to enable the composition of multiple DSALs and the concurrent use of multiple forms of aspect-oriented modularity.

AWESOME is a composition framework that enables the definition, implementation, use, and interoperability of multiple DSALs.

2. AWESOME DSALs

An aspect-oriented language (e.g., AspectJ) extends syntactically and semantically a conventional *base language* (e.g., Java). The syntactical extension, called an *aspect extension*, provides the programmer with means of aspect-oriented modularity (e.g., AspectJ provides an aspect construct). The semantical extension, called an *aspect mecha-*

* This research was supported in part by the *Israel Science Foundation (ISF)* under grant No. 926/08.

DSAL'12, March 27, 2012, Potsdam, Germany.
ACM 978-1-4503-1128-1/12/03.

nism, extends the base language semantics, called a *base mechanism*, with concern integration capabilities (e.g., AspectJ extends Java with an advice binding mechanism). A compiler implementation of an aspect mechanism (e.g., ajc) is called an *aspect weaver*.

Let $Base$ be a base language and let $DSAL_1, ..., DSAL_n$ be n domain specific aspect language extensions for $Base$. Let $\mathcal{B}$ denote the base mechanism for $Base$. Let $\mathcal{M}_1, ..., \mathcal{M}_n$ denote the aspect mechanisms for $DSAL_1, ..., DSAL_n$, respectively. Let $\mathcal{A}$ denote a multi-mechanism comprising $\mathcal{B}$ and $\mathcal{M}_1, ..., \mathcal{M}_n$.

AWESOME is a composition framework for constructing a multi-mechanism $\mathcal{A}$ for the multi-DSAL language $Base \times DSAL_1 \times \cdots \times DSAL_n$ by means of third-party composition of the mechanisms $\mathcal{B}$ and $\mathcal{M}_1, ..., \mathcal{M}_n$. We discuss the AWESOME framework in terms of addressing the following problems:

THE DSAL COMPOSITION PROBLEM [2]**:** Construct a multi-DSAL language $Base \times DSAL_1 \times \cdots \times DSAL_n$.

THE COMPOSITION SEMANTICS PROBLEM [2]**:** Define the *meaning* of a program $\langle base, aspect_1, ..., aspect_n \rangle \in Base \times DSAL_1 \times \cdots \times DSAL_n$.

THE MECHANISM ABSTRACTION PROBLEM [3]**:** Specify the semantics of a single-extension aspect language $Base \times DSAL_i$ as a modular composition of the base mechanism $\mathcal{B}$ for $Base$ and an aspect mechanism $\mathcal{M}_i$ for $DSAL_i$.

THE MECHANISM COMPOSITION PROBLEM [2]**:** Enable the assembly of $\mathcal{B}$ and $\mathcal{M}_1, ..., \mathcal{M}_n$ into a multi-mechanism $\mathcal{A}$, where:

- *Units of independent production:* $\mathcal{M}_1, ..., \mathcal{M}_n$ are independently defined. B is defined independently from $\mathcal{M}_1, ..., \mathcal{M}_n$; and $\mathcal{M}_1, ..., \mathcal{M}_n$ rely only on $\mathcal{B}$ and have an explicit context dependency only on $\mathcal{A}$.
- *Units of composition:* $\mathcal{A}$ is constructed by third-party composition [9] of $\mathcal{B}$ and $\mathcal{M}_1, ..., \mathcal{M}_n$.
- *Units of collaboration:* the semantics of $\mathcal{A}$ is the "sum" of the semantics of $\mathcal{B}$ and $\mathcal{M}_1, ..., \mathcal{M}_n$.

THE COMPOSITION SPECIFICATION PROBLEM [5, 6]**:** Identify and resolve the feature interactions in the composition of $\mathcal{M}_1, ..., \mathcal{M}_n$.

THE COMPOSITION IMPLEMENTATION PROBLEM [4]**:** Design a *composition framework* with a plug-in architecture, such that, given n third-party aspect mechanism plugins $\mathcal{M}_1, ..., \mathcal{M}_n$ for $DSAL_1, ..., DSAL_n$, and a composition specification $\mathcal{S}$, plugging them into the framework implements the multi-mechanism $\mathcal{A}$ under the specification $\mathcal{S}$.

3. Speaker Biography

David H. Lorenz is an Associate Professor in the Department of Mathematics and Computer Science at the Open University of Israel. His research interests lie primarily in the areas of aspect-oriented software engineering and language-oriented programming, particularly involving multiple domain-specific languages. Lorenz received his PhD in Computer Science from the Technion–Israel Institute of Technology. He's a member of the ACM and the IEEE. Contact him at lorenz@openu.ac.il.

References

[1] Y. Apter, D. H. Lorenz, and O. Mishali. A debug interface for debugging multiple domain specific aspect languages. In *Proceedings of the 11th International Conference on Aspect-Oriented Software Development (AOSD'12)*, Potsdam, Germany, March 2012. ACM.

[2] S. Kojarski and D. H. Lorenz. Pluggable AOP: Designing aspect mechanisms for third-party composition. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'05)*, pages 247–263, San Diego, CA, USA, October 2005. ACM Press.

[3] S. Kojarski and D. H. Lorenz. Modeling aspect mechanisms: A top-down approach. In *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*, pages 212–221, Shanghai, China, May 2006. ACM Press.

[4] S. Kojarski and D. H. Lorenz. AWESOME: An aspect co-weaving system for composing multiple aspect-oriented extensions. In *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'07)*, pages 515–534, Montreal, Canada, October 2007. ACM Press.

[5] S. Kojarski and D. H. Lorenz. Identifying feature interaction in aspect-oriented frameworks. In *Proceedings of the 29th International Conference on Software Engineering (ICSE'07)*, pages 147–157, Minneapolis, MN, May 2007. IEEE Computer Society.

[6] D. H. Lorenz and O. Mishali. SPECTACKLE: Toward a specification-based DSAL composition process. In *Proceedings of the 7th AOSD Workshop on Domain-Specific Aspects Languages (DSAL'12)*, Potsdam, Germany, March 2012. ACM.

[7] D. H. Lorenz and B. Rosenan. Cedalion: A language for language oriented programming. In *Proceedings of the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'11)*, pages 733–752, Portland, Oregon, USA, October 2011. ACM.

[8] M. Shaw. Modularity for the modern world: summary of invited keynote. In *Proceedings of the 10th International Conference on Aspect-Oriented Software Development (AOSD'11)*, pages 1–6, Porto de Galinhas, Brazil, March 2011. ACM.

[9] C. Szyperski. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley, 2nd edition, 2002. With Dominik Gruntz and Stephan Murer.

[10] M. P. Ward. Language-oriented programming. *Software-Concepts and Tools*, 15(4):147–161, 1994.