SugarJ: Library-based Language Extensibility

Sebastian Erdweg* Lennart C. L. Kats† Tillmann Rendel*
Christian Kästner* Klaus Ostermann* Eelco Visser†

* University of Marburg
† Delft University of Technology

Abstract

SugarJ is a Java-based programming language that provides extensible surface syntax, static analyses, and IDE support. SugarJ extensions are organized as libraries; conventional import statements suffice to activate and compose language extensions. We illustrate how programmers can use SugarJ to modularly extend Java's syntax, semantic analyses and IDE support.

Categories and Subject Descriptors D.3.2 [*Language Classifications*]: Extensible languages; D.2.13 [*Reusable Software*]

General Terms Languages

Keywords language extensibility, library, DSL embedding, language workbench

1. Introduction

With embedded domain-specific languages (DSLs) and language-oriented programming [1, 6], two core requirements arise: Languages have to be extensible and language extensions need to compose easily. Programmers require language extensibility to break up the ties to a single (typically general-purpose) programming language and to benefit from all aspects of embedded DSLs (for instance, domain-specific syntax or IDE support). Furthermore, since software projects touch upon multiple domains, it is essential to support composing DSLs for the common case of conflict-free language composition. For example, it should be possible to extend Java with SQL, XML or regular expressions with regard to their concrete syntax, IDE support (e.g., code completion), static analyses (e.g., XML Schema validation), and so forth. It should be simple for programmers to use any combination of such language extensions within a single source file.

To address these goals, we propose to organize and implement language extensions as libraries in the object language itself. In contrast to conventional libraries, *language libraries* do not export functionality and data structures but rather stipulate an augmentation of the object language. Due to our library-based design, a programmer can easily activate and compose language extensions by simply importing the corresponding language libraries; no external configuration or reasoning is necessary to understand a given source file. Furthermore, programmers can readily implement a language extension themselves by writing a language library; no additional tools but the object language compiler are required. Lastly, language libraries inherit the self-applicability property from conventional libraries, that is, language extensions can be used for developing language extensions: domain syntax, IDE support and static analyses for the definition of syntactic extensions, IDE extensions, static analyses, and so forth.

We have developed an extension of Java—called *SugarJ*—which demonstrates the feasibility of our library-based approach for extending a language [4]. SugarJ supports the definition of syntactic sugar within libraries, where each syntactic sugar extends the grammar of the object language and specifies a transformation—called desugaring—from the extended syntax into the base syntax. Programmers can activate and compose (domain-specific) syntax extensions through simple import statements that bring the corresponding libraries into scope. Technically, we support library-based syntax extensions through an incremental parsing process that parses a file one top-level entry at a time and adapts its own grammar as it goes along. The finally resulting abstract syntax tree is desugared using all desugarings in scope.

For example, consider the illustration of a SugarJ source file in Figure 1, where we extended the base language with syntax for XML through an import of the `xml.Sugar` library. In our embedding, we compose the grammar of XML with SugarJ's base grammar, so that SugarJ parses XML documents as part of the surrounding Java syntax. Furthermore, the `xml.Sugar` library declares a desugaring of XML to Java, which SugarJ applies after parsing. Programmers can easily compose the XML embedding with other syntactic extensions such as SQL or regular expressions by adding more import statements.

As the screenshot furthermore highlights, we generalize our library-based extensibility mechanism towards IDEs [2]. Accordingly, we promote to organize and implement IDE extensions within libraries of the object language, so that simple import statements suffice to activate and compose editor services of several DSLs. In the example above, we import the `xml.Editor` library and the `Book` schema to bring syntax coloring and code completion for XML into scope. Such editor services compose with editor services for Java because each one only affects those fragments of the syntax tree that correspond to Java *or* XML, respectively. We have implemented a prototypical extensible IDE—called *Sugarclipse*—based on the Spoofax language workbench [5] and its support for the declarative configuration and dynamic reloading of editors. Sugarclipse provides editor services on a file-by-file basis, according to the libraries in scope.

In summary, SugarJ is a lightweight and scalable alternative to model-driven language workbenches: it is lightweight because it is purely textual and aligns with the host language's module system; it is scalable because DSLs and their code generators can easily be combined and composed.

Copyright is held by the author/owner(s).
SPLASH'11 Companion, October 22–27, 2011, Portland, Oregon, USA.
ACM 978-1-4503-0940-0/11/10.

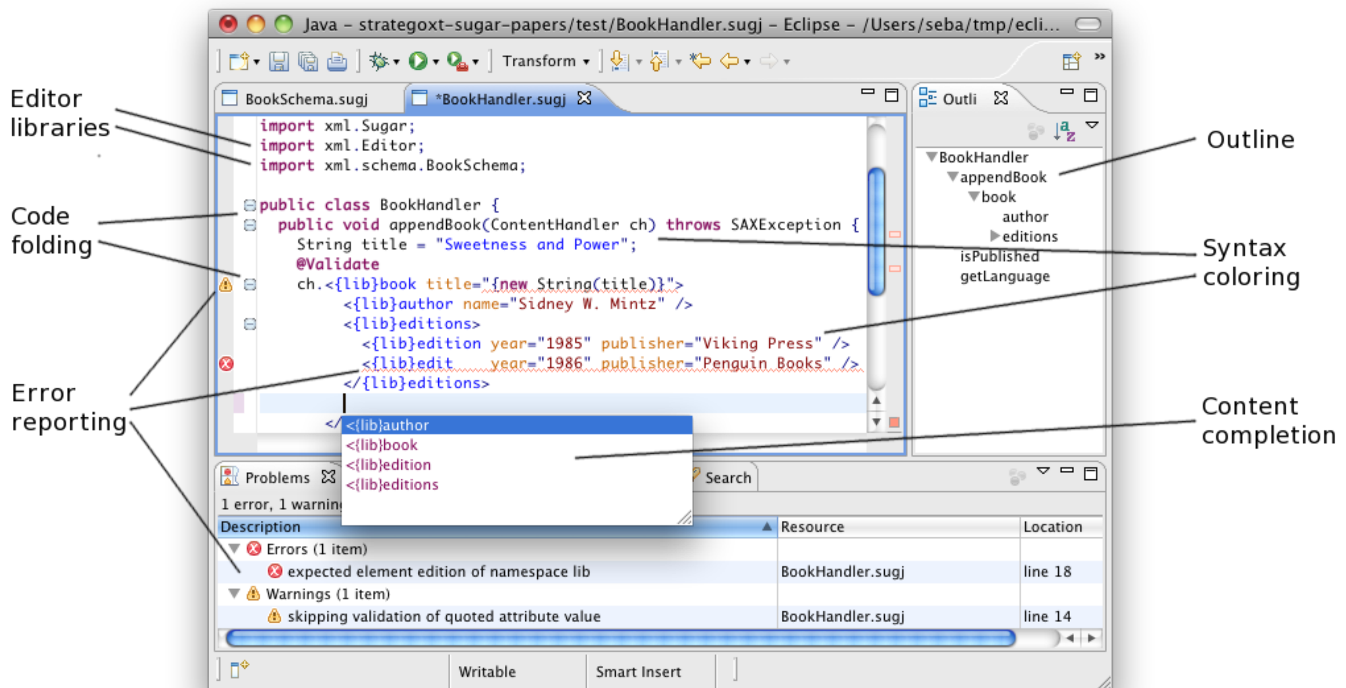

Figure 1. SugarJ extended with support for XML processing: The library xml.Sugar provides an integration of XML syntax, xml.Editor provides XML IDE support (e.g., code coloring, folding and outlining), and xml.schema.BookSchema integrates XML validation and auto-completion rules derived from an XML schema.

SugarJ is an open source project; its compiler, Eclipse plugin and case studies are publicly available at `http://sugarj.org`.

A slightly altered version of this text has been published as a demonstration proposal at the same conference [3].

Acknowledgments

This work is supported in part by the European Research Council, grant No. 203099, and NWO/EW Open Competition project 612.063.512, *TFA: Transformations for Abstractions*.

References

[1] S. Dmitriev. Language oriented programming: The next programming paradigm. Available at `http://www.jetbrains.com/mps/docs/Language_Oriented_Programming.pdf`, 2004.

[2] S. Erdweg, L. C. L. Kats, T. Rendel, C. Kästner, K. Ostermann, and E. Visser. Growing a language environment with editor libraries. In *Proceedings of Conference on Generative Programming and Component Engineering (GPCE)*. ACM, 2011.

[3] S. Erdweg, L. C. L. Kats, T. Rendel, C. Kästner, K. Ostermann, and E. Visser. Library-based model-driven software development with SugarJ. In *Companion to Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. ACM, 2011.

[4] S. Erdweg, T. Rendel, C. Kästner, and K. Ostermann. SugarJ: Library-based syntactic language extensibility. In *Proceedings of Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. ACM, 2011.

[5] L. C. L. Kats and E. Visser. The Spoofax language workbench: Rules for declarative specification of languages and IDEs. In *Proceedings of Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 444–463. ACM, 2010.

[6] M. P. Ward. Language-oriented programming. *Software – Concepts and Tools*, 15:147–161, 1995.