Library-based Language Extensibility

Sebastian Erdweg
University of Marburg

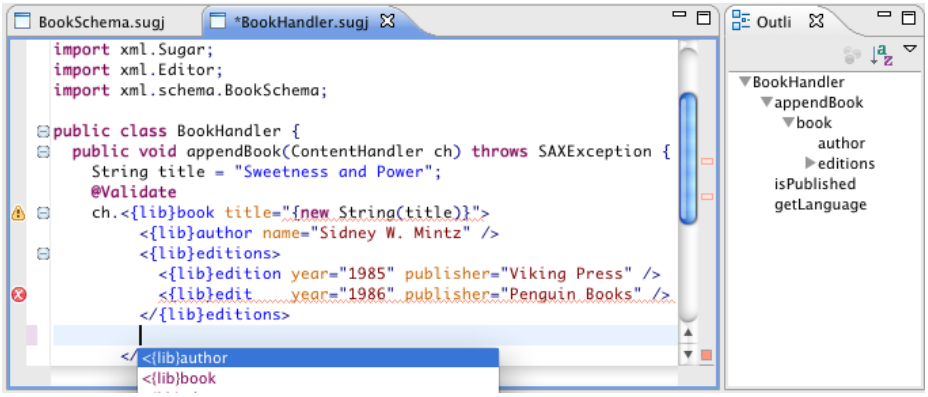
In the modern era of embedded domain-specific languages (DSLs) and language-oriented programming, two core requirements arise: Languages have to be extensible and language extensions need to compose easily. Without language extensibility, programmers are bound to a single (typically general-purpose) programming language and cannot benefit from all aspects of DSLs (for instance, domain-specific syntax or IDE support). Since software projects typically touch upon many domains, it is furthermore essential to support the composition of DSL embeddings 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 an 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.

In prior work [1], we have demonstrated the feasibility of our library-based approach for extending a language's syntax. We have developed an extension of Java—called *SugarJ*—which supports the definition of syntactic sugar within libraries. 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 following illustration of a SugarJ source file, 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.



This screenshot furthermore highlights some of the features of our current work, that is, generalizing our library-based extensibility mechanism towards IDEs. 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 [2] 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 comparison with related work, our library-based approach makes it particularly easy for users to compose language extensions (no external configuration, no user-triggered regeneration of parsers, IDE plug-ins, etc.). In general, though, languages do not always compose naturally and conflicts may arise. In our future work, we plan to investigate *declarative* means for detecting and resolving conflicts between language libraries, as well as to explore library-based extensibility for a language's type system and semantics.

References

1. S. Erdweg, T. Rendel, C. Kästner, and K. Ostermann. Sugarj: Library-based syntactic language extensibility. In *OOPSLA*. ACM, 2011. to appear.
2. L. C. L. Kats and E. Visser. The Spoofax language workbench: Rules for declarative specification of languages and IDEs. In *OOPSLA*, pages 444–463. ACM, 2010.