# Adding Domain-Specific and General Purpose Language Features to Java with the Java Language Extender [*]

Eric Van Wyk  Lijesh Krishnan  Derek Bodin  Eric Johnson

Department of Computer Science and Engineering
University of Minnesota
evw,krishnan,bodin,johnson@cs.umn.edu

## Abstract

The Java Language Extender is a compiler-generator tool that allows programmers to create new domain-adapted languages by importing a set of domain-specific language extensions into an extensible specification of Java 1.4. Language extensions define the syntax, semantic analysis, and optimizations of new language constructs. Java and the language extensions are specified as attribute grammar fragments written in Silver, an attribute grammar language supporting forwarding and higher-order attributes. Programmers need no implementation-level knowledge of the language extensions and the Silver tools automatically compose the programmer-selected extensions and the Java host language specification. We demonstrate several language extensions. One embeds the SQL database query language into Java and statically checks for syntax and type errors in SQL queries. Other extensions for the domain of computational geometry provide transformations that simplify the writing of efficient and robust geometric algorithms. General purpose extensions include Java 1.5 features such as the for-each loop and auto-boxing and unboxing and features from Pizza such as pattern matching.

***Categories and Subject Descriptors*** D.3.4 [*Programming Languages*]: Processors

***General Terms*** Compilers, Programming Languages, Domain-Specific Languages

***Keywords*** Extensible Languages, Attribute Grammars, Forwarding

## 1. Introduction

Software development is difficult. It is an error-prone and time-consuming process. This is at least partially because of the the wide semantic gap between the programmer's high-level understanding of a problem's solution and the relatively low-level language in which the solution must be encoded. General purpose language features such as classes and generics in object-oriented programming or higher-order functions in functional programming are useful for specifying abstractions for a given problem or specific domain, but they only provide the functionality of the desired abstractions. Domain-specific languages (DSLs) can provide this functionality as well as language constructs (new syntax) for the domain-specific abstractions. These raise the level of abstraction of the language to a specific domain and thus reduce the semantic gap. DSLs also provide domain-specific optimizations and analyses that are either impossible or quite difficult to specify for programs written in general purpose languages. But problems often cross multiple domains and no language will contain all of the general-purpose and domain-specific features needed to address all of the problem's aspects, thus the fundamental problem remains. Thus, programmers cannot "say what they mean" but must encode their ideas as programming idioms at a lower level of abstraction.

The Java Language Extender (JLE) is a language processing tool that addresses this fundamental problem. It makes it possible for programmers to import a set of domain-specific language extensions into an extensible specification of Java in order to create new domain-adapted languages. An extended language defined by this process has features that raise the level of abstraction to that of a particular problem. These features may be new language constructs, semantic analyses, or optimizing program transformations, and are packaged as *modular language extensions*. Language extensions can be as simple as a the Java 1.5 *for-each* loop or the more sophisticated set of SQL language constructs that statically check for syntax and type errors in SQL queries.

JLE makes an important distinction between two activities: $(i)$ implementing a language extension, which is performed by a domain-expert feature designer and $(ii)$ selecting the language extensions that will be imported into an extensible language specification in order to create an extended language. This second activity is performed by a programmer. This is similar to the distinction between library writers and library users. This distinction and the way that extensible languages and language extensions are used in our framework is diagrammed in Figure 1 in which a programmer selects the SQL and geometric (CG) language extensions that they want to use in writing a program with both database query and geometric aspects to it. An important characteristic of language extensions in JLE is that the programmer does not need any implementation-level knowledge of the extensions to import them into the host language. The extensions are also modular and composable so that several can be simultaneously composed with the Java host language. The specifications for the selected language extensions and the host language are provided to the Silver extensible compiler tools that generate a customized compiler. Thus, there is an initial "compiler generation" step that the tools, at the direction of the programmer, must perform. Language extensions are not loaded into the compiler during compilation.

[*] This work is partially funded by the National Science Foundation under NSF CAREER Award #0347860 and NSF CCF Award #0429640 and by the McKnight Foundation.

*OOPSLA'06* October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

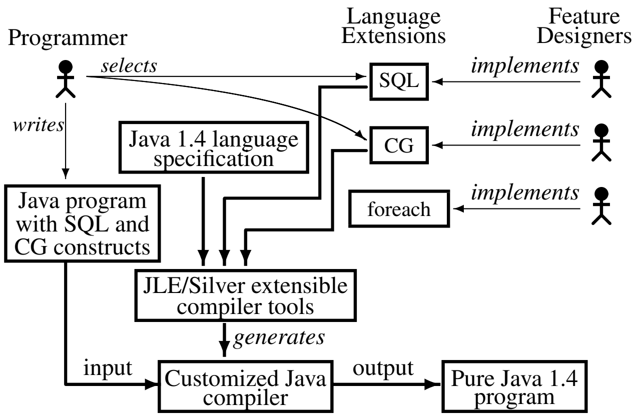

**Figure 1.** Using JLE and Language Extensions.

The Java 1.4 "host" language and the language extensions described below are specified as attribute grammars and implemented in the Silver, an attribute grammar specification language that supports higher-order attributes [10] and forwarding [8]. Language extensions specify new productions to define new language constructs and attribute definitions to define static analysis on extension or host language productions. Extension productions also specify their translation to pure Java code and "forward" attribute queries for attribute they do not explicitly define to this semantically equivalent construct [8]. This enables the high degree of modularity in language specification that we seek. JLE does not generate byte-codes, instead, forwarding is employed to extract a programs translation to pure Java 1.4 code that a traditional Java compiler then converts to byte-codes for execution.

## 2. Language Extensions

We have specified a number of modular language extensions in Silver that extend the host Java 1.4 language specification in order to highlight the static analysis and optimization capabilities of JLE.

One domain-specific extension embeds SQL into Java so that queries can be written directly, as opposed to creating character strings containing the SQL queries as is done in library-based approaches. This extension also defines attributes on the SQL productions that check at compiler-time that no syntax errors and no type errors have been made in SQL queries. When this extension is used, the extended compiler would detect the type error in the following syntactically valid program fragment (where `c` is a `Connection` to the database server and `first_name` is a String-valued field):

```
ResultSet rs = using c query { SELECT first_name
        FROM person_table WHERE first_name > 18 } ;
```

A second domain-specific extension [9] introduces static analysis and optimizing program transformations for exact-precision numeric types and symbolic-perturbation transformations for coping with data degeneracies that are specific to the domain of computational geometry. These greatly simplify the writing of efficient and robust geometric programs.

We have also created a number of modular language extensions that add new general purpose features to Java 1.4. One such extension adds algebraic data types and pattern matching similar to those found in ML and Pizza [7]. Another extension adds the foreach loop and auto-boxing and unboxing features of Java 1.5 to the Java 1.4 specification as language extensions. These extensions support our on-going development of a Silver attribute grammar specification of Java 1.5.

## 3. Related Work

Many language processing tools have been developed for building extensions to languages. For example, MetaBorg[4] allows one to extend a host language by adding concrete syntax for objects. This system uses strategies and term-rewriting to process programs. Specifying semantic analyses, like the error checking, is less straight forward that it is in JLE using attributes. The Polyglot extensible Java compiler [6] allows Java to be extended with powerful abstractions, but it requires one to write Java code to incorporate new extensions into Java host language. Macro based systems such as JSE (Java Syntax Extender) [1], Maya [2], and JTS [3] have also been proposed. Maya and JTS provide specific error checking facilities but they lack the ability to specify more general static analysis. These approaches do not support the automatic composition of the full-featured language extensions expressible in JLE.

The attribute grammar community has also addressed issues of modular language design. Of particular interest are the rewritable reference attribute grammars [5] in the JastAddII system. Language extension constructs are translated to host language constructs by destructive rewrites on the syntax tree. Thus *all* analysis on an extension construct must be completed before any analysis on its translation to the host language. This is more restrictive than forwarding and hinders the automatic composition of language features that perform semantic analysis.

## 4. Software Availability

The Java Language Extender, including Silver and the Java 1.4 and language extension attribute grammar specifications are freely available at `http://www.melt.cs.umn.edu`.

## Acknowledgments

We thank Phil Russel, Paul Huntington, August Schwerdfeger, and Jimin Gao for their help in developing extensions to Java.

## References

[1] J. Bachrach and K. Playford. The Java syntactic extender (JSE). In *Proc. of OOPSLA '01 Conf.*, pages 31–42. ACM Press, 2001.

[2] J. Baker and W. Hsieh. Maya: Multiple-dispatch syntax extension in java. In *Proc. of ACM PLDI Conf.*, pages 270–281. ACM, 2002.

[3] D. Batory, D. Lofaso, and Y. Smaragdakis. JTS: tools for implementing domain-specific languages. In *Proceedings Fifth International Conference on Software Reuse*, pages 143–53. IEEE, 2–5 1998.

[4] M. Bravenboer and E. Visser. Concrete syntax for objects: domain-specific language embedding and assimilation without restrictions. In *Proc. of OOPSLA '04 Conf.*, pages 365–383, 2004.

[5] T. Ekman and G. Hedin. Rewritable reference attributed grammars. In *Proc. of ECOOP '04 Conf.*, pages 144–169, 2004.

[6] N. Nystrom, M. R. Clarkson, and A. C. Myer. Polyglot: An extensible compiler framework for java. In *Proc. 12th International Conf. on Compiler Construction*, volume 2622 of *LNCS*, pages 138–152. Springer-Verlag, 2003.

[7] M. Odersky and P. Wadler. Pizza into Java: translating theory into practice. In *Proc. of ACM POPL Conf.*, pages 146–159, 1997.

[8] E. Van Wyk, O. de Moor, K. Backhouse, and P. Kwiatkowski. Forwarding in attribute grammars for modular language design. In *Proc. 11th Intl. Conf. on Compiler Construction*, volume 2304 of *LNCS*, pages 128–142. Springer-Verlag, 2002.

[9] E. Van Wyk and E. Johnson. Composable language extensions for computational geometry: a case study. In *Proc. 40th Hawaii International Conf. on System Sciences*, 2007.

[10] H. Vogt, S. D. Swierstra, and M. F. Kuiper. Higher-order attribute grammars. In *ACM PLDI Conf.*, pages 131–145, 1990.