SugarJ: Library-based Syntactic Language Extensibility

Grzegorz Rowiński

Object-oriented programming seminar

MIMUW, May 14th, 2012

Keywords: SugarJ, language extensibility, syntactic sugar, sugar libraries, meta-programming, domain-specific languages (DSL) embedding, language composition, static analysis, semantic analysis, editor libraries, Java, SDF, Startego, Spoofax, Sugarclipse

SugarJ: Library-based Syntactic Language Extensibility

SugarJ

- Is a (Java based) **programming language**
- It allows **meta-programming** by **syntactic language extensibility** with static (compile-time) semantic code analysis
  - by means of *sugar libraries*
- It supports extensions to augment IDE
  - by means of *editor libraries*

SugarJ: Library-based Syntactic Language Extensibility

SugarJ is **Java based** in a couple of means:

- It contains Java (at a language level)

$$\textit{Java-1.5} \subseteq \textit{SugarJ}$$

- It may be seen as a preprocessor outputting pure Java
- The tools provided in the SugarJ implementation are written mostly in Java

SugarJ: Library-based Syntactic Language Extensibility

SugarJ contains *SDF* language

- It's embedded in SugarJ

$$E_{\text{SDF/SugarJ}}(SDF) \subseteq SugarJ$$

- It's used in sugar libraries to describe syntax extensions provided by a library

SDF (syntax definition formalism) – a modular, declarative language for describing context-free grammars, oriented for SGLR parsing method (SGLR = Scannerless Generalized LR)

SugarJ: Library-based Syntactic Language Extensibility

SugarJ contains *Stratego* language

- It's embedded in SugarJ

$$E_{\text{Stratego/SugarJ}}(\textit{Stratego}) \subseteq \textit{SugarJ}$$

- It's used in sugar libraries to describe program transformations provided by a library

Stratego – a functional (more precisely: strategic term rewriting) programming language designed for expressing program transformations, operates on AST (abstract syntax tree) of transformed program

Stratego enables usage of a concrete syntax of transformed program's language, as well!

SugarJ: Sugar Libraries

Sugar library stipulates an augmentation of the base language, by:

- extending the syntax

  Sugar library may add new productions to the *current grammar*

- providing de-sugaring

  Sugar library provides transformation rules (or strategies), which enable the SugarJ compiler to *de-sugar* a new syntax i.e. transform it to the syntax allowed in embedding environment and perform contextual static checking

  Subsequent application of de-sugaring by SugarJ compiler results in pure Java syntax being emitted.

SugarJ: Sugar Libraries

Ideally (this is ***not exactly*** the case in SugarJ):

Let $v \in SugarJ$ be a sugar library stipulating an extended language $L(v)$.

Then

$$vw \in SugarJ \text{ for every } w \in L(v),$$

i.e. SugarJ is **closed** under extensions it may express.

Self-applicability: vw above may be a sugar library.

Composability:

$$v_1 v_2 w \in SugarJ \text{ for every } w \in L(v_1) \oplus L(v_2)$$

SugarJ: Sugar Libraries

Why the previous slide is not quite correct?

GLR parsers try to manage in ambiguous grammars. The unresolvable case, however, is when there are two or more distinct parse trees possible for program text.

The composition of languages (⊕) is a language generated by grammar being the sum of grammars (union of production sets).

This may lead to composed grammars with unresolvable ambiguity.

SugarJ: Sugar Libraries

SugarJ rejects a program when parser finds such ambiguity.

Most likely it may happen in composed languages, but even with single sugar library in use, as the library is composed with the base language.

So, despite $v \in SugarJ$ *and* $w \in L(v)$,

$vw \in SugarJ$ may not hold.

This is not considered really harmful by SugarJ authors – such cases are easily fixable by programmer.

Very simple example: pairs

Let's extend SugarJ (well, Java) with pair syntax, so make the following code

```
(String, Integer) p = ("Answer", 42);
```

to be equivalent to

```
pair.Pair<String, Integer> p = pair.Pair.create("Answer", 42);
```

SugarJ: Sugar Libraries | Very simple example: pairs

pair/concrete/Test.sugj

```
package pair.concrete;
import pair.concrete.Syntax;
import pair.concrete.Desugar;
public class Test {
  public static void main(String[] args) {
    (String, Integer) p = ("Answer", 42);
    System.out.println(p);
  }
}
```

SugarJ: Sugar Libraries | Very simple example: pairs

pair/concrete/Syntax.sugj

```
package pair.concrete;
import org.sugarj.languages.Java;
public sugar Syntax {
  context-free syntax
    "(" JavaExpr "," JavaExpr ")" ->
      JavaExpr {cons("PExpr")}
    "(" JavaType "," JavaType ")" ->
      JavaType {cons("PType")}
}
```

SugarJ: Sugar Libraries | Very simple example: pairs

pair/concrete/Desugar.sugj

```
package pair.concrete;
import concretesyntax.Java;
import pair.concrete.Syntax;
public sugar Desugar {
  desugarings
    pair2expr
    pair2type
  rules
    pair2expr : |[ (~expr:e1, ~expr:e2) ]| ->
       |[ pair.Pair.create(~e1, ~e2) ]|
    pair2type : |[ (~type:t1, ~type:t2) ]| ->
       |[ pair.Pair<~t1, ~t2> ]|
}
```

SugarJ: Sugar Libraries

Sugar libraries may be used to extend a language with general-purpose language features – pairs or tuples for example.

Other examples of general-purpose features, to name a few: regular expressions, closures (functional style in Java), JavaBeans accessors and other automatic boiler-plate code generation.

SugarJ authors emphasize usage of sugar libraries as a novel method for *domain-specific language (DSL)* embedding. This makes them address SugerJ as a *language-oriented programming* language.

SugarJ: Domain-specific languages

Traditional methods of DSL embedding:

- string encoding
- class library embedding
- dedicated preprocessors

SugarJ: Domain-specific languages

String DSL encoding, e.g.:

- Java regular expressions
  - Pattern p = Pattern.compile("a*b");
- SQL in JDBC
- JPQL in JPA
- XML as strings
  - StringBuffer sb = new StringBuffer();
  - sb.append(“<item>\n”);
  - ...

SugarJ: Domain-specific languages

Class library DSL embedding, e.g.:

- XML with JDOM
- JPA QueryBuilder

Dedicated DSL preprocessors, e.g.:

- Oracle Pro*C – C embedded PL/SQL

SugarJ: Domain-specific languages

Comparison of DSL embedding methods

	advantages	disadvantages
String encoded DSL	• simplicity • dynamically constructed • original syntax	• no static checking at all • escaping required • no editor support (usually)
Class library embedded DSL	• partial static checking • partial editor support	• syntax deviated (in general) • static checking is partial
Dedicated embedded DSL preprocessing	• original syntax (in the majority of cases) • full static checking (as the domain allows)	• hardly composable • non-uniform: tool-chain dependencies exist • may not support dynamic constructions

SugarJ: Domain-specific languages

SugarJ works in a way similar to a preprocessor method while it allows composability and makes the processing uniform (possibly including the editor support).

This is expected to result in higher programmer convenience, simpler build system and less tool-chain dependencies.

SugarJ: Domain-specific languages

XML sugar library usage example:

```
ContentHandler ch = new Test();
String title = "Sweetness and Power";
ch.<book title="{new String(title)}">
          <author name="Sidney W. Mintz" />
        </book>;
```

Regexp sugar library usage example:

```
boolean b = args[0].matches(/Sugar\S[A-Z]*/);
```

SugarJ: Editor Libraries

IDE / type-time support seems to be required feature for any production (or claiming) programming language / environment nowadays. Language extensibility makes this requirement harder to fulfill.

Editor libraries may accompany SugarJ's sugar libraries and enable IDE support for language extensions.

Sugarclipse is currently the only tool which respects SugarJ's editor libraries. Sugarclipse is Eclipse plug-in based on *Spoofax* (which is also based on SDF and Stratego).

SugarJ: Editor Libraries

There are eight *editor services*, which may be augmented by (declarative, domain-specific) language of editor libraries:

- Syntax coloring
- Code folding
- Outlining
- Content completion
- Reference resolving
- Hover help
- Refactoring (or projection)
- Parentheses matching

SugarJ: Limitations

Limitations / future work in SugarJ:

- SugarJ does not deal deeply with ambiguity in compositions of de-sugaring rules and editor libraries
- Debugging is only possible on emitted Java code
- Sugarclipse is not production-stable yet

SugarJ: Stability

Sugarclipse installed as described in (very brief) guide was unable to compile the pair example.

Quick hacking was necessary to make it work.

What I did was:

```
cd ${PROJECT_DIRECTORY}
rm -fr .sugarjcache
ln -s /tmp .sugarjcache
```

It helped, a bit...

SugarJ: Stability

Sugarclipse signalizes random NullPointerException at compile-time – it's not repeatable – finally it compiles the pair example.

Sugarclipse seems slow. There's a risk that at the current stage it's too slow for interactive/IDE usage.

I have not tried command-line compiler `sugarjc` – I just believe it's more stable.

SugarJ: Library-based Syntactic Language Extensibility

SugarJ home page:

http://sugarj.org/

Spoofax, Stratego and SDF home pages:

http://strategoxt.org/Spoofax/

http://strategoxt.org/Sdf

http://strategoxt.org/Stratego/StrategoLanguage

SugarJ: Library-based Syntactic Language Extensibility

[1] Sebastian Erdweg, Tillmann Rendel, Christian Kästner and Klaus Ostermann. **SugarJ: Library-based Syntactic Language Extensibility**. In *Proceedings of Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 391–406. ACM, 2011.

[2] Sebastian Erdweg and Lennart C. L. Kats and Tillmann Rendel and Christian Kästner and Klaus Ostermann and Eelco Visser. **Growing a Language Environment with Editor Libraries**. In *Proceedings of Conference on Generative Programming and Component Engineering (GPCE)*, pages 167–176. ACM, 2011

[3] Stefan Fehrenbach. **Retrofitting Language-oriented Design with SugarJ**. Bacherlor thesis, University of Marburg, November 2011. Co-supervised with Klaus Ostermann
http://www.informatik.uni-marburg.de/~seba/publications/thesis-fehrenbach.pdf

Q&A + Discussion + Feedback, pls

Thank you!