A Language Workbench for Creating Production-ready Extensions to AspectJ

Arik Hadas
Open University of Israel

Research in domain-specific aspect languages suffers from the deterioration of the Aspect Bench Compiler (abc). We present an alternative language-oriented programmer's workbench for developing extensions to AspectJ. Much like the abc, the workbench allows researchers to implement and evaluate new extensions. In contrast to abc, however, it also provides IDE support, compatibility with AspectJ 1.7.4 and Java 7, and support for programming in multiple extensions. For validation we implemented the workbench by integrating Spoofax and AWESOME, and used it to create complex third-party extensions, including COOL, explicit join points (EJP), and closure join points (CJP).

Highlights

- **Workbench, not a compiler**
  - ❖ Editing tools for creating DSALs and for programming with DSALs
  - ❖ Ability for define the weaving semantics required for DSALs
  - ❖ Compatibility with development tools for AOP
  - ❖ Code transformation is a responsibility of the back-end, not the front-end ⟶ standalone compilation of DSALs
  - ❖ Context-aware & source location preserving code transformations
- **Combines the benefits of abc, AWESOME, and Spoofax, avoiding their limitations**

	abc	AWESOME	Spoofax	Workbench
Tools for custom syntax definition	🟥	🟥	🟩	🟩
Extensible Java/AspectJ syntax	🟨	🟥	🟩	🟩
Tools for code transformation	🟥	🟥	🟩	🟩
Editing tools for end-programmers	🟥	🟥	🟩	🟩
Ability to define the weaving semantics required for DSAL	🟨	🟩	🟥	🟩
Works with recent version of AspectJ	🟥	🟩	🟥	🟩
Compliance with AJDT	🟥	🟩	🟥	🟩

Validation

- **Open source implementation**
- **Plugins for well-known third-part extensions**
  - ❖ COOL, EJP, CJP
  - ❖ Including features that were omitted in original prototypes

Example: SDF for CJP

```
module languages/closures/Main

imports
   languages/java-15/Main
   languages/aspectj/ajc/Main

exports
   sorts JoinpointDeclaration
   context-free syntax
      "exhibit" MethodName "(" {FormalParam ","}* ")" Block "("
         {Expr ","}* ")" -> Expr {cons("ClosureJoinpoints")}
      "exhibit" MethodName Block ->
         Expr {cons("ShortClosureJoinpoints")}

      JoinpointDeclaration -> AspectBodyDec
      "joinpoint" ResultType Id "(" {FormalParam ","}* ")" Throws? ";"
         -> JoinpointDeclaration{cons("JoinpointDeclaration")}
      (Anno | MethodMod)* CJPAdviceSpec Throws? Block ->
         AdviceDec {cons("CJPAdvice")}
      "before" Id "(" {FormalParam ","}* ")" ->
         CJPAdviceSpec {cons("CJPBefore")}
      "after" Id "(" {FormalParam ","}* ")" ->
         CJPAdviceSpec {cons("CJPAfter")}
      "after" Id "(" {FormalParam ","}* ")" "returning" CJPSingleParam?
         -> CJPAdviceSpec {cons("CJPAfterReturning")}
      "after" Id "(" {FormalParam ","}* ")" "throwing" CJPSingleParam?
         -> CJPAdviceSpec {cons("CJPAfterThrowing")}
      "(" FormalParam? ")" -> CJPSingleParam {cons("CJPSingleParam")}
         ResultType "around" Id "(" {FormalParam ","}* ")" ->
         CJPAdviceSpec {cons("CJPAround")}

   lexical syntax
      "exhibit" -> Keyword
      "joinpoint" -> PseudoKeyword
```

Example: Implementation of CJP

Language Workbench (Spoofax)

Eclipse Plugin for CJP: error checking, syntax highlighting and auto-completion

```
package research;
public class HelloWorld  {
   public static void main(String[] args) {
     exhibit say(String message) {
       System.out.println("Hello, " + message);
     }("World");
   }
}
```

```
package research;
aspect Impact {
  joinpoint void say(String message);
  after say(String message) {
    System.out.println(
      "It did a " + message + " of good.");
  }
}
```

Composition Framework (AWESOME)

Code Transformation

```
closure-to-java-impl=
  ?ShortClosureJoinpoints(<or(?MethodName(Id(jp_name)), ?MethodName(_,    Id(jp_name)))>, block);
  !Invoke(
   Method(
    NewInstance(
      None()
     , ClassOrInterfaceType(TypeName(Id("JoinpointWrapper")), None())
     , []
     , Some(
       ClassBody(
          [ MethodDec(
             MethodDecHead(
                [MarkerAnno(TypeName(Id("Closure"))), Public()]
              , None(), Void(), Id(jp_name), [], None())
            , block)]))))
    , None()
    , Id(jp_name))
  , [])
```

- **Stratego definition** for transforming a closure-call to calling a method with a @Closure annotation
- **Transformation plugin** in AWESOME for CJP

Transformed Code

```
package research;
import closures.runtime.*;
import org.aspectj.lang.annotation.*;
import org.aspectj.lang.*;
public class HelloWorld  {
 public static void main(String[] args){
   new JoinpointWrapper(){
     @Closure
     @SourceLocation(...)
     public void say(String message){
      System.out.println("Hello, " + message);
     }
   }.say("World");
 }
}
```

Metadata for weaving

```
package research
import closures.runtime.*;
import org.aspectj.lang.annotation.*;
import org.aspectj.lang.*;
aspect Impact {
 private static @Joinpoint void say(String message) {
   throw new UnsupportedOperationException();
 }
 @After("call(@Closure * say(String)) &&args(message)")
 @JoinpointSignature(args = {String.class}, name = "say")
 public void say483096566(String message,
                                    JoinPointthisJoinPoint) {
   System.out.println("It did a " + message + " of good.");
 }
}
```

Compilation and Weaving

```
public void preweave(List<ResolvedType> types) { ... }
List<BcelShadow> around(MultiMechanism mm, LazyClassGen clazz):
   reifyClass(mm, clazz) { ... }
public List<IEffect> match(BcelShadow shadow) { ... }
public List<IEffect> order(BcelShadow shadow, List<IEffect> effects) { ... }
void around(MultiMechanism mm, List effects, BcelShadow shadow):
   execution(void MultiMechanism.mix(List, BcelShadow)) { ... }
```

- **Weaving mechanism** in AWESOME for CJP
- The weaving model was extended with a **preweaving** phase

Running

```
$ java –cp woven.jar:awesome_runtime.jar research.HelloWorld
Hello, World
It did a World of good.
```

Producing **executable woven code**

Advisor: David H. Lorenz
This research was supported in part by the *Israel Science Foundation (ISF)* under grant No. 1440/14.

STUDENT RESEARCH COMPETITION
Association for Computing Machinery
sponsored by Microsoft Research