First-Class Domain Specific Aspect Languages

Arik Hadas[1] and David H. Lorenz[1,2]
[1]Open University of Israel
[2]Technion-Israel Institute of Technology

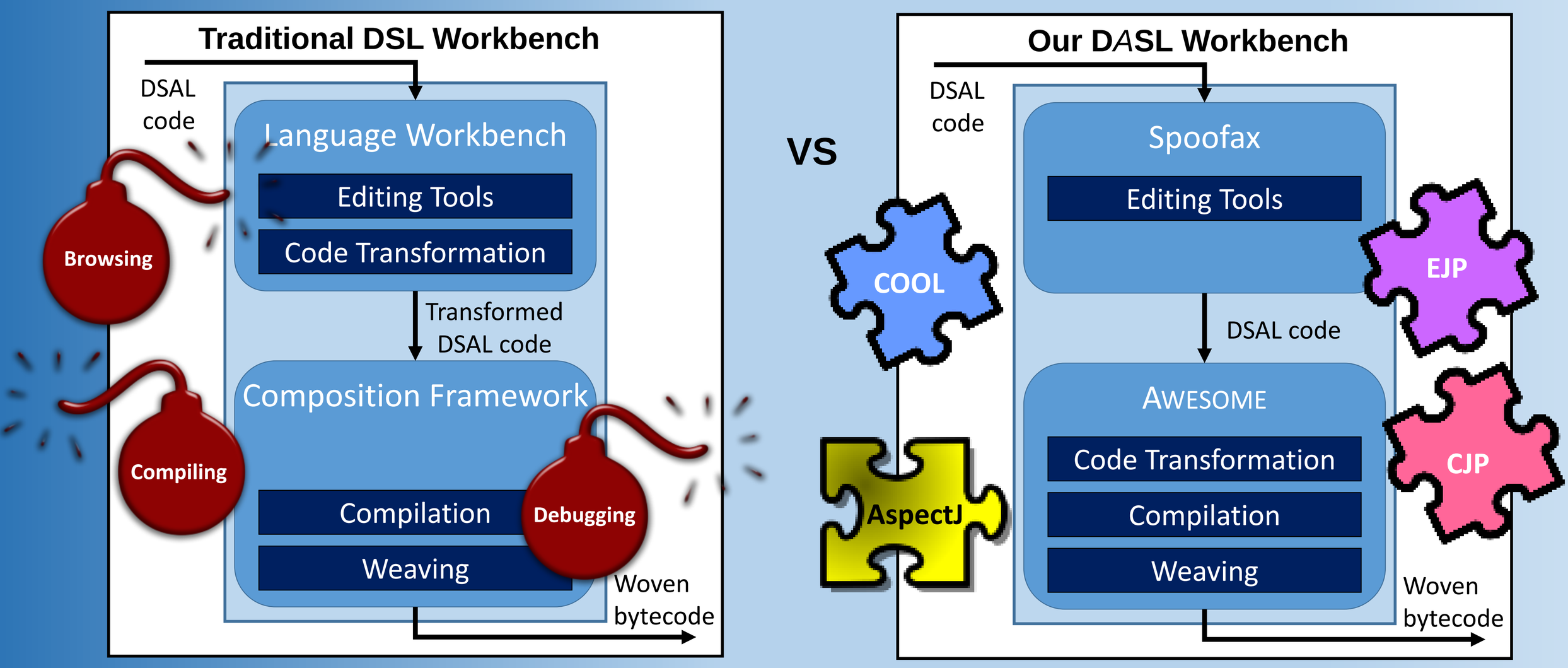
Programming in a domain specific aspect language (DSAL) typically involves some language workbench and some AOP composition framework.

However, DSAL development remains *second-class* in two respects:

- Language workbenches do not support definition of the weaving semantics needed for DSALs
- DSAL source code is incompatible with existing AOP tools due to the additional pre-processing phase

We present a *DSAL workbench* solution in which DSALs are first-class DSLs as well as first-class AOP languages. We illustrate the approach by integrating Spoofax and AWESOME into such a workbench.

DSAL Development	Language Workbench (LW)	Composition Framework (CF)	LW + CF	**DSAL Workbench**
Language Definition	✓	✗	✓	✓
Weaving Semantics	✗	✓	✓	✓
First-class Tooling	✗	✗	✗	✓



Presentation:

Arik Hadas and David H. Lorenz:
Demanding First-Class Equality for Domain Specific Aspect Languages

Thursday, 3:45 PM
Location: LSC 312

Demonstration:

Arik Hadas and David H. Lorenz:
A Language Workbench for Implementing Your Favorite Extension to AspectJ

Thursday, 10:50 AM
Location: LSC 300

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

AJDT out of the box for the COOL language

BoundedStack.java

```
package base;

public class BoundedStack implements Stack {

    protected Object[] buffer;

    private int usedSlots = 0;

    public BoundedStack(int capacity) {
        this.buffer = new Object[capacity];
    }

    public Object pop() {
        Object result = buffer[usedSlots - 1];
        usedSlots--;
        buffer[usedSlots] = null;
        return result;
    }

    public void push(Object obj) {
```

Multiple markers at this line
- implements base.Stack.push
- advised by injar aspect: BoundedStackCoord.cool

```
}
```

BoundedStackCoord.cool

```
package base;

coordinator base.BoundedStack {

  selfex {push(java.lang.Object), pop()};
  mutex {push(java.lang.Object), pop()};

  condition full = false, empty = true;
  int top = 0;

  push(java.lang.Object):
      requires (!full);
      on_entry {top = top + 1;}
      on_exit  {
        empty = false;
        if (top == buffer.length) full = true;
      }

  pop():
      requires (!empty);
      on_entry {top = top - 1;}
      on_exit  {
        full = false;
        if (top == 0) empty = true;
      }
}
```