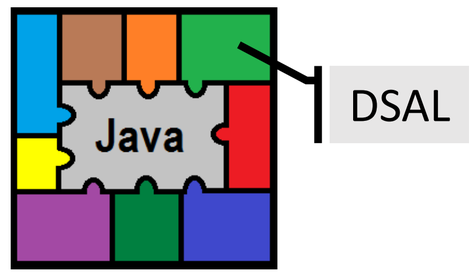
Experiencing with Language Oriented Modularity

Arik Hadas and David H. Lorenz
Open University of Israel

Language Oriented Modularity (LOM), taking after *Language Oriented Programming (LOP*), is a programming methodology that involves the development and use of *Domain Specific Aspect Languages (DSALs)* on-demand during the software modularization process. A DSAL is a programming language that is both domain-specific and aspect-oriented. It provides not only domain-specific abstractions and notations like an ordinary *Domain Specific Language (DSL)* does, but also a modularization mechanism for the separation of domain-specific crosscutting concerns.



Approach: Transforming DSAL code to annotated AspectJ code using a Language Workbench

Hides join points associated with this type to resolve multi-DSAL foreign advising conflicts

Preserves the source location of the advice at the DSAL code to achieve compatibility with AJDT

```
logs for com.mucommander.job.impl.CopyJob:
   case start log COPY_STARTED with
       nbFiles baseSourceFolder baseDestFolder files
   case finish log COPY_FINISHED with
       nbFiles baseSourceFolder baseDestFolder
   case interrupt log COPY_INTERRUPTED with
       baseSourceFolder baseDestFolder
   case pause log COPY_PAUSED with
       baseSourceFolder baseDestFolder nbProcessedFiles
   case resume log COPY_RESUMED with
       baseSourceFolder baseDestFolder;
```

```
@HideType
public privileged aspect Logs {
  @BridgedSourceLocation(line=1,
       file="/mucommander/src/main/java/ ... /jobs.audit",
       module="jobs.audit")
  after(CopyJob job): execution(void start()) && this(job) {
    if (true) {
      audit("start copying {0} files from {1} to {2} ({3})",
          job.nbFiles, job.baseSourceFolder,
          job.baseDestFolder, job.files);
      return;
    }
  }
  //... skipped ...
}
```

Comparison between our implementation of COOL and the one in the AWESOME composition framework

Coordinator for a bounded stack

```
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;
      }
}
```

Our @Hide annotations preserve the join point "fingerprint" of the aspect code

Aspect in AspectJ that uses the stack

```
public aspect AJAuditor {
    pointcut toLog(): call(* *.*(..)) &&
                      !cflow(within(AJAuditor));
    before(): toLog() { log(thisJoinPoint); }
    protected void log(JoinPoint jp) {
        BoundedStack buf = BoundedStack.getInstance();
        try { if (buf != null) buf.add(jp); }
        catch(Exception e) { /** ... skipped ... */ }
    }
}
```

Resolution of external variables inside the coordinator

Implementation	Grammar	Code Transformation		Weaver Plugin
		EV	Other	
Language	SDF	Stratego (AST)	Stratego (AST)	Java
CF Approach	34	761 (4168)	297 (3001)	1557
Our Approach	34	0	382 (3008)	0

- **Significantly less code**
- **Simpler (High-level programming with AspectJ instead of bytecode manipulation)**
- **Implementation done completely using the Spoofax language workbench**

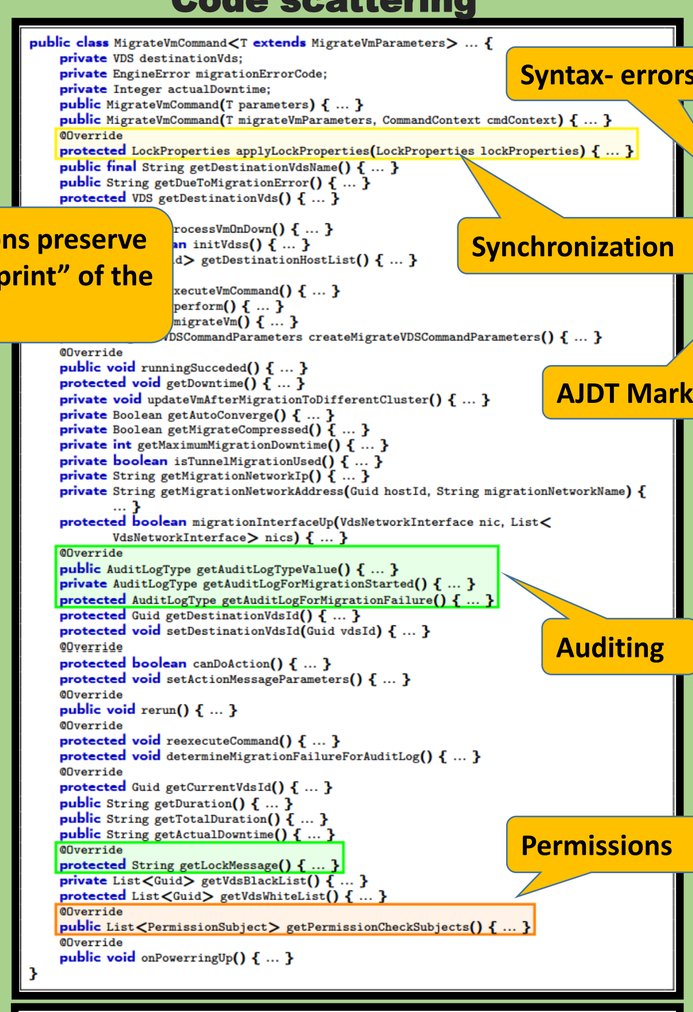
Language Oriented Modularity: From Theory to Practice

Research track, Wed, 10:45, D0.07

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

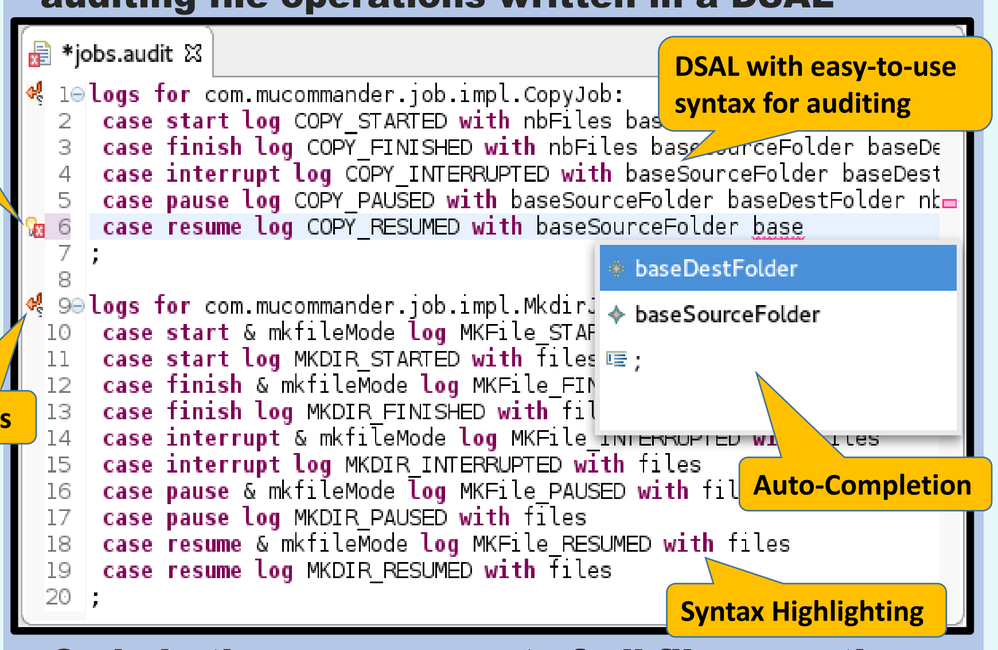
Separating crosscutting concerns in the oVirt open source virtualization platform

Code scattering

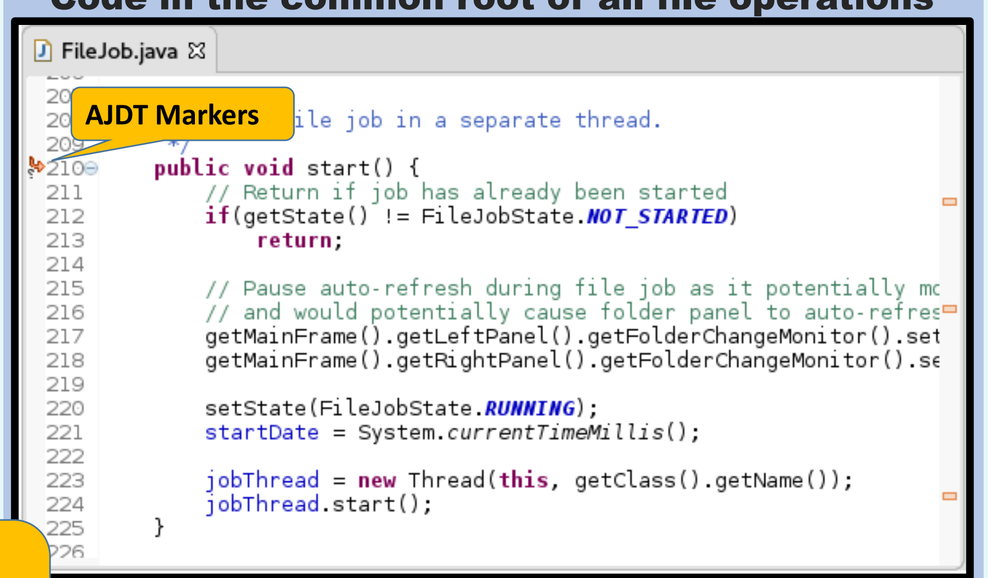


New crosscutting feature in the muCommander open source file manager

Aspect solution for a missing feature of auditing file operations written in a DSAL



Code in the common root of all file operations



Code tangling