Language Oriented Modularity: From Theory to Practice

Arik Hadas

Dept. of Mathematics and Computer Science
The Open University of Israel

Adviser:
David H. Lorenz
**Language Oriented Modularity (LOM)**

- **Methodology**
  - For modularization via development and use of Domain Specific Aspect Languages (DSALs)

- **Theory**
  - Very powerful methodology

- **Practice**
  - Underutilized in modern projects
Open source enterprise application for providing and managing virtual data centers
Scattered Code in oVirt

MigrateVmCommand

AddDiskCommand

synchronization

Auditing

Permissions
Tangled Code in oVirt

CommandBase

```java
private boolean internalCanDoAction() {
    boolean returnVa
    try {
        boolean returnVa
        try {
            boolean returnVa
            if (isCanDoActionSupportsTransaction()) {
                transaction = TransactionSupport.suspend();
            }
            boolean returnVa
            if (isUserAuthorizedToRunAction() && isBackwardsCompatible() && validateInputs() && acquireLock() && canDoAction() && internalValidateAndSetUserState()) {
                boolean returnVa
                if (returnValue && getReturnValue().getCanDoActionMessages().size() > 0) {
                    log.warn("Failed to CanDoAction of action \"\" failed for user \"\", getActionType(), getUserName(), String.join(getReturnValue().getCanDoActionMessages(), ",");
                }
                boolean returnVa
                if (transaction != null) {
                    TransactionSupport.resume(transaction);
                }
                boolean returnVa
                catch (DataAccessException dataAccessEx) {
                    log.error("Data access error during CanDoActionFailure.", dataAccessEx);
                    addCanDoActionMessage(EngineMessage.CAN_DO_ACTION_DATABASE_CONNECTION_FAILURE);
                }
                catch (RuntimeException ex) {
                    log.error("Error during CanDoActionFailure.", ex);
                    addCanDoActionMessage(EngineMessage.CAN_DO_ACTION_GENERAL_FAILURE);
                }
                boolean returnVa
                if (returnValue) {
                    freeLock();
                }
                return returnV
            }
        }
    }
```

Permissions

Synchronization

Synchronization
Problem in a Nutshell

- General Purpose Aspect Languages (GPALs)
  - Too complex to use
Problem in a Nutshell

- **General Purpose Aspect Languages (GPALs)**
  - Too complex to use

- **Domain Specific Aspect Languages (DSALs)**
  - Too complex to develop
Contribution in a Nutshell

- **Practical LOM**
  - Make the DSAL development process more like that of DSLs
Outline

- Introduction
- Problem
- Approach
- Evaluation
- Conclusion
Language Oriented Modularity (LOM)

- A methodology that puts Domain Specific Aspect Languages (DSALs) at the center of the software modularization process.
Language Oriented Modularity (LOM)

- A methodology that puts Domain Specific Aspect Languages (DSALs) at the center of the software modularization process.
  - On-demand development and use of DSALs
Language Oriented Modularity (LOM)

- A methodology that puts Domain Specific Aspect Languages (DSALs) at the center of the software modularization process.
  - On-demand development and use of DSALs
Pros of LOM

- **Separation of crosscutting concerns**
  - Improved software modularity compared to GPLs or DSLs

- **Domain specific languages**
  - Programming in more declarative and simpler languages than GPALs

![Venn diagram showing the relationship between DSLs, DSALs, and AOP]

- DSLs
- DSALs
- AOP
  - COOL
  - RIDL
  - AspectGrid
  - Racer
  - KALA
Cons of LOM

- **Cost**
  - Definition and implementation cost is higher

- **Effectiveness**
  - Use of DSALs (compared to GPALs) is less effective than DSLs (compared to GPLs)

<table>
<thead>
<tr>
<th></th>
<th>DSLs</th>
<th>DSALs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost-effectiveness</td>
<td>🌟</td>
<td>😞</td>
</tr>
</tbody>
</table>
Working Hypothesis

- Making LOM more like LOP could make LOM more practical
  - DSALs more like DSLs (definition; implementation)
  - DSALs more like GPALs (use)
## Problem Preview

<table>
<thead>
<tr>
<th></th>
<th>DSLs</th>
<th>DSALs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Language Definition</strong></td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td><strong>Language Implementation</strong></td>
<td>☑️</td>
<td>☞️</td>
</tr>
<tr>
<td><strong>Language Use</strong></td>
<td>☑️</td>
<td>☞️</td>
</tr>
</tbody>
</table>
Language Definition

- **Syntax**
  - Domain-specific notations and abstraction

- **Semantics**
  - Complex to define the weaving semantics when multiple DSALs are being used simultaneously

<table>
<thead>
<tr>
<th></th>
<th>DSLs</th>
<th>DSALs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain-Specific Syntax</td>
<td>🌻</td>
<td>🌻</td>
</tr>
<tr>
<td>Weaving Semantics</td>
<td>Not Needed</td>
<td>🙁</td>
</tr>
</tbody>
</table>
Language Implementation

- **Language workbenches are for DSLs**
  - Produces a parser for the custom syntax
  - Produces a transformation to some GPL

- **No equivalent tool for DSALs**
  - The implementation of weaving semantics is generally a costly task

<table>
<thead>
<tr>
<th></th>
<th>DSLs</th>
<th>DSALs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parsing</td>
<td>![Smiley]</td>
<td>![Sad]</td>
</tr>
<tr>
<td>Compilation</td>
<td>![Smiley]</td>
<td>![Sad]</td>
</tr>
</tbody>
</table>
Language Use

- **Programming with a DSL**
  - Language workbench produces editing tools

- **Programming with a DSAL**
  - Simpler language but lacks development tools

<table>
<thead>
<tr>
<th></th>
<th>DSLs</th>
<th>DSALs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common Editing Tools</td>
<td><img src="emoji" alt="Smiley" /></td>
<td><img src="emoji" alt="Sad" /></td>
</tr>
<tr>
<td>Build Tools</td>
<td><img src="emoji" alt="Smiley" /></td>
<td><img src="emoji" alt="Sad" /></td>
</tr>
<tr>
<td>Aspect Development Tools</td>
<td>Not Needed</td>
<td><img src="emoji" alt="Sad" /></td>
</tr>
</tbody>
</table>
Outline

- Introduction
- Problem
- Approach
- Evaluation
- Conclusion
Key Idea

- **Transform DSALs into a kernel language that is based on a GPAL**
  - No need to implement a weaver per DSAL
  - Aspect development tools for the GPAL would work with the DSAL code
Transformation-based Approach

- **Restriction on crosscutting concerns**
  - CCC that could be modularized using a GPAL
- **DSALs can be transformed into that GPAL**
  - Aspect development tools for the GPAL would work with the DSAL code
  - Most of the developers program with simpler and more declarative languages
GPAL-based Kernel Language

- The kernel language provides constructs for resolving possible multi-DSALs conflicts
  - Hide joinpoint shadows in order to resolve foreign advising issues
  - Sort advise to resolve co-advising issues

- During transformation of DSAL code these constructs can be defined declaratively
  - Annotate join points that should be hidden
  - Annotate advice so they could be sorted

- The simpler the DSALs are, the less common these conflicts are
Leveraging Language Workbench

- Most of the DSAL development can be done using a language workbench
  - Grammar definition for the DSAL
  - Transformation of the DSAL to the kernel language

- Supportive tools provided by a language workbench
  - Reduce the implementation cost

- Editing tools for programming with the DSALs
  - Generated by the language workbench
LOM for oVirt

- We implemented DSALs for 3 crosscutting concerns found in the oVirt project
  - Synchronization
  - Permission checks
  - Auditing
Demonstration - oVirtSync

- Developing a DSAL for synchronization in oVirt: https://youtu.be/uj80yWutQak
- Resolving synchronization in oVirt with DSAL: https://youtu.be/PTy9rYDQSo4
- The code is available on GitHub https://github.com/OpenUniversity
Implementation Effort

• One time effort
  – Compiler for the kernel language

• Per-application effort
  – Compile oVirt with AspectJ compiler

• The produced DSALs were
  – Relatively easy to define
  – Relatively easy to implement
  – Relatively easy to use
Outline

• Introduction
• Problem
• Approach
• Evaluation
• Conclusion
Related Work

- **Domain Specific Aspect Languages**
  - [Fabry at al., 2015] A Taxonomy of Domain-Specific Aspect Languages.

- **Transformation-based AOP Composition Frameworks**
  - [Shonle at al., 2003] XAspects: An extensible system for domain specific aspect languages.

- **SpecTackle**
We bring the DSAL development process one step closer to the development process of DSLs. For a class of DSALs that are in a sense reducible to a GPAL, their cost-effectiveness is improved. That way, their cost-effectiveness is improved:

- The implementation cost is reduced
- The definition cost could be reduced
- The effectiveness of using them is increased

That may make the LOM methodology practical for real-world software development process.
Conclusion

- New classes of DSALs
  - Application specific
  - Disposable
Conclusion

- New classes of DSALs
  - Application specific
  - Disposable

Disposable Aspect Languages (DispALs)

Tailored to a specific application

Reusability

Application-specific  Cross-applications  Cross-domains
Conclusion

- New classes of DSALs
  - Application specific
  - Disposable

- Challenge general conception of language design
  - Lower reuse may improve cost-effectiveness
Conclusion

- New classes of DSALs
  - Application specific
  - Disposable
- Challenge general conception of language design
  - Lower reuse may improve cost-effectiveness
- Agile-like software modularization process
  - Start with disposable DSALs and gradually move to reusable DSALs
Thank You!

Arik Hadas
Dept. of Mathematics and Computer Science
The Open University of Israel

arik.hadas@openu.ac.il
https://github.com/OpenUniversity