Application-Specific Language-Oriented Modularity: A Case Study of the oVirt Project *

Arik Hadas[1] David H. Lorenz[1,2] †

[1]Open University, Raanana 43107, Israel
[2]Technion–Israel Institute of Technology
Haifa 32000, Israel
arik.hadas@openu.ac.il, dhlorenz@cs.technion.ac.il

Abstract

Despite the availability of general purpose aspect languages (GPALs) and the availability of frameworks for creating domain specific aspect languages (DSALs), tangled and scattered code still prevails in modern software projects. Through the prism of a case study of the oVirt open source project we examine the conjecture that it may simply be too costly to implement crosscutting concerns in today's GPALs and DSALs. We introduce a subcategory of DSALs, called *application specific aspect languages (ASALs)*, that along with a programming methodology, called *language oriented modularity (LOM)*, allows such concerns to be modularized in a cost-effective, practical way. We illustrate this process concretely for the oVirt project.

Categories and Subject Descriptors D.2.11 [*Software Engineering*]: Software Architectures—Domain-specific architectures; D.2.6 [*Software Engineering*]: Programming Environments—Programmer workbench.

General Terms Language, Design.

Keywords Aspect oriented programming (AOP), Domain specific aspect language (DSAL), Application specific aspect language (ASAL), Domain specific language (DSL), Language workbench, AWESOME.

1. Introduction

Consider the task of modularizing crosscutting concerns in a project implemented in Java. Using an off-the-shelf aspect language like AspectJ [12] is the most economical solution in terms of language design and language implementation. However, in terms of language use, there is a nontrivial price tag to be paid for using AspectJ in the project. First, AspectJ has a steep learning curve in terms of mastering the language. Developers would have to be familiar with an additional complex programming language, and training a new programmer that joins the project will take significantly longer. Second, AspectJ may not always be the best language for expressing the crosscutting concerns at hand.

In contrast to GPALs, DSALs provide syntax that is closer to the problem domain, thereby making it easier for developers to program in the aspect language. Of course, one should be able to implement most domain-specific concerns as aspects in AspectJ. But due to its general purpose nature, aspects written in AspectJ are less declarative than aspects written in DSALs.

1.1 Off-The-Shelf Domain Specific Aspect Languages

One could try to express the crosscutting concerns found in a project using DSALs that have already been implemented by third-parties. Indeed, Fabry *et al.* [7] report on a corpus of 36 DSALs for different problem domains. Obviously, reusing one of these DSALs eliminates the effort of language design and implementation. However, only a few of the DSALs in the corpus (reportedly, 5 out of the 36) were implemented using a common infrastructure that would permit them to be used simultaneously with other DSALs. Therefore, in order to use these off-the-shelf DSALs, one would typically need to integrate several tools, which in turn increases the effort of development and maintenance. Even if one finds several off-the-shelf DSALs that are relevant to the domains of interest and were implemented using a common infrastructure, chances are that these DSALs would not be the best fit. For example, COOL [14] is a DSAL for handling synchronization but is not suitable for the kind of synchronization found in the oVirt project, because a COOL coordinator locks Java objects, whereas oVirt-Engine locks logical entities (Sect. 2).

1.2 In-House Domain Specific Aspect Languages

One could develop in-house DSALs for the problem domains found in the project. Of course, there will be a high price tag for language design and implementation, but that effort can be alleviated to some extent by using a language workbench [8] for defining a custom grammar for these DSALs and producing front-end editing tools for programming in them. However, there will still remain significant implementation effort, because language workbenches provide little to no support for defining the weaving semantics needed for DSALs. For that one would need to use either an extensible aspect compiler such as the Aspect Bench Compiler (abc) [1, 2] or an aspect composition framework such as AWESOME [13] (when multiple DSALs are expected to be used simultaneously). Defining the weaving semantics would likely require low-level programming, which is an uncommon expertise among Java programmers and would significantly lessen the cost-effectiveness of DSAL development.

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

† Work done in part while visiting the Faculty of Computer Science, Technion—Israel Institute of Technology.

MODULARITY Companion'16, March 14–17, 2016, Málaga, Spain
ACM. 978-1-4503-4033-5/16/03...
http://dx.doi.org/10.1145/2892664.2892692

Listing 1: ExportVmTemplateCommand.java

```
public class ExportVmTemplateCommand<T extends MoveOrCopyParameters> extends ... {
 private String cachedTemplateIsBeingExportedMessage;
 public ExportVmTemplateCommand(T parameters) {...}
 protected ExportVmTemplateCommand(Guid commandId) {...}
 @Override
 protected LockProperties applyLockProperties(LockProperties lockProperties) {...}
 @Override
 protected void moveOrCopyAllImageGroups(final Guid containerID, final Iterable<DiskImage>
        disks) {...}
 @Override
 protected Map<String, Pair<String, String>> getSharedLocks() {...}
 @Override
 protected Map<String, Pair<String, String>> getExclusiveLocks() {...}
 @Override
 protected void executeCommand() {...}
 private String getTemplateIsBeingExportedMessage() {...}
 @Override
 protected boolean canDoAction() {...}
 @Override
 public AuditLogType getAuditLogTypeValue() {...}
 @Override
 protected void setActionMessageParameters() {...}
 @Override
 protected void incrementDbGeneration() {...}
 @Override
 protected void endActionOnAllImageGroups() {...}
 @Override
 public Map<String, String> getJobMessageProperties() {...}
}
```

Listing 2: AddDiskCommand.java

```
public class AddDiskCommand<T extends AddDiskParameters> ... {
 protected AddDiskCommand(Guid commandId) { ... }
 public AddDiskCommand(T parameters) { ... }
 public AddDiskCommand(T parameters, CommandContext commandContext) { ... }
 @Override
 protected boolean canDoAction() { ... }
 protected boolean checkIfLunDiskCanBeAdded(DiskValidator diskValidator) { ... }
 protected boolean checkIfImageDiskCanBeAdded(VM vm, DiskValidator diskValidator) { ... }
 private boolean isShareableDiskOnGlusterDomain() { ... }
 private boolean canAddShareableDisk() { ... }
 private boolean checkExceedingMaxBlockDiskSize() { ... }
 private boolean isStoragePoolMatching(VM vm) { ... }
 protected boolean checkImageConfiguration() { ... }
 private double getRequestDiskSpace() { ... }
 @Override
 protected boolean isVmExist() { ... }
 private DiskImage getDiskImageInfo() { ... }
 private boolean isExceedMaxBlockDiskSize() { ... }
 protected DiskLunMapDao getDiskLunMapDao() { ... }
 protected DiskImageDynamicDao getDiskImageDynamicDao() { ... }
 private Guid getDisksStorageDomainId() { ... }
 @Override
 public Guid getStorageDomainId() { ... }
 @Override
 public List<PermissionSubject> getPermissionCheckSubjects() { ... }
 @Override
 protected void setActionMessageParameters() { ... }
 @Override
 protected void executeVmCommand() { ... }
 private void createDiskBasedOnLun() { ... }
 protected VmDevice addManagedDeviceForDisk(Guid diskId, Boolean isUsingScsiReservation) {
        ... }
 protected VmDevice addManagedDeviceForDisk(Guid diskId) { ... }
 protected boolean shouldDiskBePlugged() { ... }
 private void createDiskBasedOnImage() { ... }
 private void createDiskBasedOnCinder() { ... }
 private VdcActionParametersBase buildAddCinderDiskParameters() { ... }
 private void setVmSnapshotIdForDisk(AddImageFromScratchParameters parameters) { ... }
 private void addDiskPermissions(Disk disk) { ... }
 @Override
 public AuditLogType getAuditLogTypeValue() { ... }
 private boolean isDiskStorageTypeRequiresExecuteState() { ... }
 private AuditLogType getExecuteAuditLogTypeValue(boolean successful) { ... }
 protected AuditLogType getEndSuccessAuditLogTypeValue(boolean successful) { ... }
 @Override
 protected VdcActionType getChildActionType() { ... }
 @Override
 protected List<Class<?>> getValidationGroups() { ... }
 @Override
 protected Map<String, Pair<String, String>> getSharedLocks() { ... }
 @Override
 protected Map<String, Pair<String, String>> getExclusiveLocks() { ... }
 @Override
 protected void setLoggingForCommand() { ... }
 private Guid getQuotaId() { ... }
 @Override
 protected void endSuccessfully() { ... }
 private void plugDiskToVmIfNeeded() { ... }
 protected boolean setAndValidateDiskProfiles() { ... }
 @Override
 public List<QuotaConsumptionParameter> getQuotaStorageConsumptionParameters() { ... }
 protected StorageDomainValidator createStorageDomainValidator() { ... }
}
```

1.3 Application Specific Aspect Languages

While the trade-offs between using a GPAL in a project and using DSALs instead affect where the effort is put, the overall programming effort remains high in both cases. In order to significantly reduce the programming effort, we consider in this paper a subset of DSALs, namely *Application Specific Aspect Languages (ASALs)*.

ASALs are a special case of DSALs restricted to a specific application, just like DSALs are a special case of GPALs restricted to a specific domain. By foregoing any claims for reuse across applications, ASALs can be made simple and cheap to design, implement, and use. We illustrate this concretely through a case study of the oVirt project. ASALs, coupled with *Language Oriented Modularity (LOM)* [15], make a cost-effective process for modularizing crosscutting concerns.

2. oVirt – a Case in Study

oVirt[1] is an open-source production-ready enterprise application for providing and managing virtual data centers and private cloud solutions. For example, Red Hat Enterprise Virtualization (RHEV), a commercial competitor to VMware vSphere, is based on oVirt and is deployed in big organizations, such as British Airways,[2] DreamWorks Animation,[3] etc. oVirt is also used in Academia for teaching and for research.[4]

oVirt-Engine is the control center of the oVirt distributed system that manages the different hosts that run virtual machines (VMs). It consists of 761K lines-of-code, contributed by 173 developers. Specifically, oVirt-Engine is a Java server application running on top of a JBoss application server. It serves as the front-end of the entire data center, executing operations it receives from clients and reports back to them the up-to-date status of the data center.

The core design of oVirt-Engine is based on the COMMAND design pattern [9]. Each operation that is supported by oVirt-Engine is modeled by a command class that inherits from a common root called `CommandBase`. This design allows one to encapsulate all the implementation details for a particular operation in a single object, making these operations easier to understand. Having a common ancestor imposes a common structure on all commands, allowing one to design frameworks that treat the commands uniformly. An example of such a framework is the command coordinator framework for asynchronous execution of operations.[5]

2.1 Crosscutting Concerns

We focus on three concerns found in oVirt-Engine, namely *synchronization*, *auditing*, and *permissions*. These concerns crosscut many modules in the oVirt-Engine application.

Synchronization This concern is about preventing conflicting commands from running simultaneously. For that, each command defines its read-locks (the method `getSharedLocks`), its write-locks (the method `getExclusiveLocks`), and global properties

[1] `http://www.ovirt.org`

[2] `http://www.redhat.com/en/about/press-releases/british-airways-chooses-rhev-to-improve-it-systems-to-build-internal-cloud`

[3] `http://www.redhat.com/en/about/blog/dreamworks-animation-utilizes-the-red-hat-portfolio-to-build-a-private-cloud-for-financial-and-creative-applications`

[4] `http://www.ovirt.org/community/user-stories/universidad-de-sevilla-case-study`

[5] `http://www.ovirt.org/Features/Design/CommandCoordinator`

Listing 3: MigrateVmCommand.java

```
public class MigrateVmCommand<T extends MigrateVmParameters> ... {
    private VDS destinationVds;
    private EngineError migrationErrorCode;
    private Integer actualDowntime;
    public MigrateVmCommand(T parameters) { ... }
    public MigrateVmCommand(T migrateVmParameters, CommandContext cmdContext) { ... }
    @Override
    protected LockProperties applyLockProperties(LockProperties lockProperties) { ... }
    public final String getDestinationVdsName() { ... }
    public String getDueToMigrationError() { ... }
    protected VDS getDestinationVds() { ... }
    @Override
    protected void processVmOnDown() { ... }
    protected boolean initVdss() { ... }
    private List<Guid> getDestinationHostList() { ... }
    @Override
    protected void executeVmCommand() { ... }
    private boolean perform() { ... }
    private boolean migrateVm() { ... }
    private MigrateVDSCommandParameters createMigrateVDSCommandParameters() { ... }
    @Override
    public void runningSucceded() { ... }
    protected void getDowntime() { ... }
    private void updateVmAfterMigrationToDifferentCluster() { ... }
    private Boolean getAutoConverge() { ... }
    private Boolean getMigrateCompressed() { ... }
    private int getMaximumMigrationDowntime() { ... }
    private boolean isTunnelMigrationUsed() { ... }
    private String getMigrationNetworkIp() { ... }
    private String getMigrationNetworkAddress(Guid hostId, String migrationNetworkName) {
            ... }
    protected boolean migrationInterfaceUp(VdsNetworkInterface nic, List<
            VdsNetworkInterface> nics) { ... }
    @Override
    public AuditLogType getAuditLogTypeValue() { ... }
    private AuditLogType getAuditLogForMigrationStarted() { ... }
    protected AuditLogType getAuditLogForMigrationFailure() { ... }
    protected Guid getDestinationVdsId() { ... }
    protected void setDestinationVdsId(Guid vdsId) { ... }
    @Override
    protected boolean canDoAction() { ... }
    protected void setActionMessageParameters() { ... }
    @Override
    public void rerun() { ... }
    @Override
    protected void reexecuteCommand() { ... }
    protected void determineMigrationFailureForAuditLog() { ... }
    @Override
    protected Guid getCurrentVdsId() { ... }
    public String getDuration() { ... }
    public String getTotalDuration() { ... }
    public String getActualDowntime() { ... }
    @Override
    protected String getLockMessage() { ... }
    private List<Guid> getVdsBlackList() { ... }
    protected List<Guid> getVdsWhiteList() { ... }
    @Override
    public List<PermissionSubject> getPermissionCheckSubjects() { ... }
    @Override
    public void onPowerringUp() { ... }
}
```

Listing 4: internalCanDoAction

```
private boolean internalCanDoAction() {
 boolean returnValue = false;
 try {
  Transaction transaction = null;
  if (!isCanDoActionSupportsTransaction()) {
   transaction = TransactionSupport.suspend();
  }
  try {
   returnValue =
    isUserAuthorizedToRunAction() && isBackwardsCompatible()
    && validateInputs() && acquireLock()
    && canDoAction() && internalValidateAndSetQuota();
    if (!returnValue && getReturnValue().getCanDoActionMessages().size() > 0) {
     log.warn("CanDoAction of action '{}' failed for user {}. Reasons: {}",
         getActionType(), getUserName(),
         StringUtils.join(getReturnValue().getCanDoActionMessages(), ','));
    }
  } finally {
   if (transaction != null) {
    TransactionSupport.resume(transaction);
   }
  }
 } 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);
 } finally {
  if (!returnValue) {
   freeLock();
  }
 }
 return returnValue;
}
```

for its locks (the method `applyLockProperties`). Each lock includes the type and identifier of the entity being locked along with a message that is displayed once this lock prevents another command from being executed. The global properties include the scope of the locks (defines whether the locks should be released at the end of the synchronous or at the end of the asynchronous execution of the command), and whether the command should wait until it manages to acquire the locks or fail if they cannot be acquired.

Auditing This concern is mainly about producing informative messages at different stages of command execution. For that, each command defines the messages to be produced in each stage of its execution (the method `getAuditLogTypeValue`).

Permissions This concern is about ensuring that only users with sufficient permissions on entities are able to execute commands that affect them. For that, each command defines the entities one needs to have permission on along with the required permission type in order to execute it (the method `getPermissionCheckSubjects`).

2.2 Scattered and Tangled Code

The code of three classes, `ExportVmTemplateCommand`, `AddDiskCommnad`, and `MigrateVmCommand` is outlined in Listings 1, 2, and 3, respectively. The code related to the crosscutting concerns *synchronization*, *auditing*, and *permissions* are marked in the listings in yellow, green, and red, respectively.

The implication of having scattered code is twofold. First, when a large part of the code of a single class is not about its core responsibility, it is difficult to understand the business logic. For example, more than a quarter of the 425 lines-of-code of `ExportVmTemplateCommand` are related to implementing the synchronization and auditing concerns. Second, it is difficult to understand the overall system behavior. For example, the synchronization concern is scattered across the command classes, making it unclear which commands lock a specific entity. This lack of traceability reduces productivity and even blamed directly for bugs.[6]

The code of the `CommandBase` class presents problems that are typical to a class that has grown too large: 1269 lines-of-code. This class contains code that handles different system concerns that are common to all commands, resulting in severe tangling. For example, the `internalCanDoAction` method (Listing 4) checks preconditions for running a command, but its code also handles synchronization and permissions issues like releasing locks. The tangled code makes the class hard to maintain.

2.3 Summary

Apparently, at least in the oVirt project, developers do not use aspects to separate out certain crosscutting concerns, such as synchronization, auditing, and permissions, even at the price of scattering and tangling that negatively affect the modularity of the code (and have been blamed as the source for bugs in oVirt).

3. Applying LOM to oVirt

Next we illustrate the cost-effectiveness of ASALs and the LOM programming methodology for solving the crosscutting concerns we identified in oVirt-Engine.

[6] The cause of some reported bugs can be traced back to the problem of scattering and tangling, e.g., `https://bugzilla.redhat.com/1251956`

Listing 5: Language definition

```
oVirtSync ::= Commands;

Commands ::= Command ["," Commands];

Command ::= "locks for" CommandType "(" [Scope] ["wait"] ")" ":"
      [ExclusiveLocks] [InclusiveLocks] [Message] ";";

ExclusiveLocks ::= "exclusively" "{" Locks "}";

InclusiveLocks ::= "inclusively" "{" Locks "}";

Locks ::= Lock ["," Locks];

Lock ::= "group:" LockingGroup "instance:" MethodName [
     Condition];

Condition ::= "if" MethodName;

Message ::= "message" EngineMessage Vars;

Vars ::= Var ["," Vars];

Var ::= "<" String "," MethodName ">";

Scope ::= "sync" | "async";
```

Listing 6: ovirt.locks

```
1  locks for org.ovirt.engine.core.bll.ExportVmTemplateCommand (
       sync):
2    exclusively (overrides) {
3      group: REMOTE_TEMPLATE instance: getVmTemplateId
4    }
5    inclusively (overrides) {
6      group: TEMPLATE instance: getVmTemplateId
7    }
8    message: ACTION_TYPE_FAILED_TEMPLATE_IS_BEING_EXPORTED
         <"TemplateName", getVmTemplateName>
9  ;
10
11 locks for org.ovirt.engine.core.bll.MigrateVmCommand (async):
12   exclusively (overrides) {
13     group: VM instance: getVmId
14   }
15   message: ACTION_TYPE_FAILED_VM_IS_BEING_MIGRATED <"
         VmName", getVmName>
16 ;
17
18 locks for org.ovirt.engine.core.bll.AddDiskCommand (sync):
19   exclusively (overrides) {
20     group: VM_DISK_BOOT instance: getVmId if isBootableDisk
21   }
22   inclusively (overrides) {
23     group: VM instance: getVmId
24   }
25 ;
```

3.1 LOM for oVirt-Engine

Following the LOM methodology, we start by developing an ASAL for each of the crosscutting concerns in oVirt-Engine that were described in Sect. 2, namely *synchronization*, *auditing*, and *permissions*. Then, we program aspect solutions for these concerns in the produced languages. These languages are designed specifically for oVirt-Engine, and developed using our implementation of an ASAL workbench, comprising the Xtext language workbench [6] and an extended version of the AWESOME composition framework.

3.2 *oVirtSync*

Due to space considerations, we describe in more detail the definition, implementation, and use of just one of the ASALs, namely *oVirtSync*, which handles the synchronization crosscutting concern.

Aspect language definition Listing 5 shows the grammar definition of *oVirtSync* in BNF format. The language model consists of `Command` elements. Each `Command` element includes: the type of the command the locks are defined for, the scope of the locks (for the entire lifetime of the command or only for the synchronous part), whether or not the command is supposed to wait until all the locks can be acquired (otherwise it fails if any of the locks cannot be acquired), and optionally: a list of read-locks, a list of write locks, and a message to present if another command cannot be executed because of the acquired locks. Each `Lock` element inside the lock lists includes: the type of the entity to lock, the identifier of the entity instance to lock and optionally, a method that defines if the lock should be acquired for the specific instance of the command or not.

The defined ASAL is tailored to oVirt-Engine. First, it uses types that are defined in oVirt-Engine such as *LockingGroup*. Second, the ASAL provides only the constructs that are needed in order to solve the synchronization concern in oVirt-Engine. To keep its implementation and use simple, there is no attempt to generalize its constructs in order to make the language reusable.

Aspect language use An aspect written in *oVirtSync* that defines the locks for three commands in oVirt-Engine is shown in Listing 6. Lines 1-9 specify that `ExportVmTemplateCommand` neeeds to lock exclusively the template on the destination storage with the identifier that is returned by the `getVmTemplateId` method, and to lock inclusively the template on the source storage with the same identifier. These locks should be released at the end of the synchronous part of the command execution (line 1). The command should fail if the locks cannot be acquired ('wait' is not specified in line 1), and the message that is defined in line 8 should be displayed if another command cannot be executed because of the acquired locks. Similarly, lines 11-16 specify the locks for `MigrateVmCommand`. Note that this time, the locks should be released at the end of the entire command execution (line 11). Lastly, lines 18-25 specify the locks for `AddDiskCommnad`. Note the usage of a condition in line 20 to indicate that the exclusive lock should be acquired only if the disk is bootable.

To write the aspect, we used an Eclipse plugin that was generated by Xtext for *oVirtSync* and provided front-end editing tools for programming in *oVirtSync*. In addition, AJDT produced a cross-reference view that presented which lines in the *oVirtSync* aspect affect a certain command and vice versa.

Aspect language implementation Xtext was very instrumental in the implementation of *oVirtSync*. First, we defined the language using the grammar definition format of Xtext. This format provides a language designer with useful constructs for defining custom languages, such as constants, references to JVM (Java Virtual Machine) types, conditions, etc. Xtext provides IDE support for writing in that format.

Second, we defined a transformation of code written in *oVirtSync* to a kernel language that extends AspectJ. Listing 7 shows part of the code that was generated by the code transformation for the aspect in Listing 6. Lines 3-15 show two pieces of advice that were generated for `MigrateVmCommand`. The advice in lines 3-6 defines the general properties of its locks, i.e., their scope and that the command should fail if any of the locks cannot be acquired. The

Listing 7: Locks.aj

```
public privileged aspect Locks {
  ... skipped ...
  @BridgedSourceLocation(line=11, file="/bll/src/main/java/org/
       ovirt/engine/core/bll/ovirt.locks", module="ovirt.locks")
  LockProperties around(LockProperties lockProperties, org
       .ovirt.engine.core.bll.MigrateVmCommand command):
       execution(* applyLockProperties(..)) && args(
       lockProperties) && target(command) {
    return lockProperties.withScope(Scope.Command).withWait(
         false);
  }

  @BridgedSourceLocation(line=11, file="/bll/src/main/java/org/
       ovirt/engine/core/bll/ovirt.locks", module="ovirt.locks")
  Map<String, Pair<String, String>> around(org.ovirt.
       engine.core.bll.MigrateVmCommand command): execution(*
        getExclusiveLocks()) && target(command) {
    Map<String, Pair<String, String>> locks = new
         HashMap<String, Pair<String, String>>();
    locks.put(command.getVmId().toString(),
      LockMessagesMatchUtil.makeLockingPair(LockingGroup.
           VM,
      "ACTION_TYPE_FAILED_VM_IS_BEING_MIGRATED
           "+"$VmName "+command.getVmName()));
    return locks;
  }
  ... skipped ...
}
```

advice in lines 8-15 defines its exclusive locks. Note the use of *@BridgedSourceLocation*, which is a part of the kernel language annotations for specifying the original location of advice in the ASAL code in order to preserve compatibility with AspectJ development tools. The transformation was defined in the Xtend programming language [3], which is a domain specific language for code transformation provided by Xtext and we enjoyed the IDE support Xtext provides for Xtend developers.

4. Evaluation

We successfully separated out the crosscutting concerns we identified in oVirt-Engine by defining ASALs and writing aspects in these ASALs. Code scattering was resolved by encapsulating the code that was spread across different commands in oVirt-Engine into a single module implemented in the corresponding ASAL. Code tangling was resolved by extracting the code that was tangled inside the `CommandBase` class into the generated kernel language code that each ASAL aspect was transformed to.

The effectiveness of ASALs in separating crosscutting concerns is no surprise, since that is what DSALs and ASALs are designed to do. Rather, the uniqueness of our approach is in improving the cost-effectiveness of the ASAL-based LOM development process.

On the one hand, in our approach the programming effort required for developing an ASAL is comparable to that of developing a DSL. This is because both the process and the tools for developing an ASAL are on par with those used for developing a DSL. The transformation of ASAL code into the kernel language is easier to implement than writing custom weaving semantics in low-level programming constructs. This transformation is similar to the transformation of DSL code into a general purpose language, which is a common task when applying LOP. Moreover, in both language definition and implementation, our approach offers IDE support similar to that provided when developing DSLs. This is a direct result of using Xtext, which is tailored for DSL development.

In our approach one can program effectively with the produced ASALs and use ASAL code everywhere AspectJ code is used. One gets supportive IDE tools as those available when programming with AspectJ. The IDE plugin that is produced by Xtext provides front-end editing tools that are commonly provided by IDEs for programming with AspectJ. The seamless integration of AJDT with ASAL code, thanks to preserving with *@BridgedSourceCode* the original source location of important elements in the ASAL code, provides aspect specific development tools, such as a cross-reference view, for programming in the ASAL. Moreover, the fact that the task of programming in AspectJ is shifted from developers that program in oVirt-Engine to only few language designers who implement the language supposedly reduces the development cost, since most developers do not need to learn AspectJ and can program instead with the simplified (almost configuration-like) languages that are tailored to the problems they need to solve. Since the ASALs are transformed to a kernel language that is based on AspectJ, ASALs gain the same run-time performance and ease of deployment as AspectJ. Furthermore, with the stand-alone compiler for ASALs, i.e., a compiler that can receive ASAL code as input and transform it internally to the kernel language, ASAL code can be compiled like AspectJ code.

Threats to Validity One might doubt whether our case study is a representative example of a problem that is common in the software industry, or whether our approach will be effective for similar problems in other applications, since in this case study we analyzed the problem and evaluated our approach only in oVirt-Engine. However, the fact that oVirt-Engine is similar to other projects in the industry in terms of requirements, design and technologies, reduces the chances that the problem is unique to oVirt-Engine. Neither the LOM methodology nor our ASAL workbench were designed specifically for oVirt-Engine. This increases the chances that our approach is valid for other applications.

The fact that our ASALs were not developed as a part of the development process of oVirt-Engine as dictated by the LOM methodology, but rather were designed as alternative solutions to existing implementations, might hide additional complexity in applying ASAL-based LOM development processes. In addition, the solutions for crosscutting concerns using our ASALs were developed by the ASAL designer. It might be that when developers other than the language designer program in the ASALs it will reveal hidden complexity that was not exposed in this case study. Further study in which ASALs are designed as original solutions for crosscutting concerns and then used by developers that have not been involved in their development should be done in order to eliminate these concerns.

5. Related Work

Elsewhere [10] we proposed a DSAL workbench as a solution for what can be considered a DSAL-based LOM development process. We argued there that by making DSALs as easy to develop as DSLs and as easy to program with as GPALs, a DSAL-based LOM development process would be cost-effective. In this work, we refine, implement, and evaluate this approach. Applying LOM to ASALs instead of DSALs, which are not expected to be reusable across applications, allows us to rely on existing standards, conventions, and tools, which collectively simplify significantly the cost of implementing these ASALs.

Various tools have been proposed for facilitating the development of DSALs. The Aspect Bench Compiler (abc) [1, 2] provides an extensible design. Aspect composition frameworks like AWESOME [13] provide tools to resolve weaving conflicts. In comparison, in our approach we make it easier to use the language (by providing front-end editing tools and aspect development tools).

Low-level programming is not required. Conflict resolution is also somewhat easier due to the use of ASAL. In oVirt, for example, there was no need to resolve conflicts because the languages were simple. However, the kernel language is equipped with constructs for resolving such conflicts.

Several approaches introduce an intermediate layer between the base and the aspect modules, e.g., *explicit join points* [11], *closure join points* [4], and *join point interfaces* [5]. There is a similarity between LOM and these approaches in the sense that the languages developed in the LOM methodology can also be considered as an intermediate layer or an interface. However, the motivation is different. In LOM we do so in order to improve the overall development productivity: only a few programmers need to program in AspectJ in order to implement the ASALs, while the majority of the programmers can program in languages that are much more simplified and closer to the application domain.

6. Conclusion

Despite its power to improve modularity, the LOM methodology, which promotes separation of crosscutting concerns via use of DSALs, has yet to be widely adopted by the software industry. This might indicate that there are hidden ramifications to using LOM in real-world software development. In this paper we present a new approach for using LOM with a subcategory of DSALs, called ASALs, that are tailored to a specific application. ASALs offer an alternative to using DSALs or GPALs.

We identify integrated crosscutting concerns in the oVirt open source project and demonstrate the process of using our approach to separate them. Three ASALs were created for three crosscutting concerns, namely synchronization, permissions, and auditing. Using these ASALs we developed aspect solutions for problems of synchronization, permissions, and auditing in some of the commands that exist in oVirt.

We illustrate that our approach simplifies language design, implementation, and use, affording a cost-effective development process for LOM. Language design is simpler because the languages are tailored to a specific application. Language implementation is simpler mainly because the languages are transformed to another language instead of requiring the implementation of dedicated weavers for them. Language use is simpler thanks to the availability of supportive development tools.

The work suggests that the applicability of the LOM methodology for real-world software development can be improved by applying it with simpler and more specific languages, even at the expense of lower reusability.

References

[1] P. Avgustinov, A. S. Christensen, L. Hendren, S. Kuzins, J. Lhoták, O. Lhoták, O. de Moor, D. Sereni, G. Sittampalam, and J. Tibble. abc: an extensible AspectJ compiler. In *Proceedings of the 4th International Conference on Aspect-Oriented Software Development (AOSD'05)*, pages 87–98, Chicago, Illinois, USA, Mar. 2005. ACM Press.

[2] P. Avgustinov, A. S. Christensen, L. J. Hendren, S. Kuzins, J. Lhoták, O. Lhoták, O. de Moor, D. Sereni, G. Sittampalam, and J. Tibble. abc: an extensible AspectJ compiler. In A. Rashid and M. Aksit, editors, *Transactions on Aspect-Oriented Software Development I*, number 3880 in Lecture Notes in Computer Science, pages 293–334. Springer Verlag, 2006.

[3] L. Bettini. *Implementing Domain-Specific Languages with Xtext and Xtend*. Packt Publishing, 2013.

[4] E. Bodden. Closure joinpoints: block joinpoints without surprises. In *Proceedings of the 10th International Conference on Aspect-Oriented Software Development (AOSD'11)*, pages 117–128, Porto de Galinhas, Brazil, Mar. 2011. ACM Press.

[5] E. Bodden, É. Tanter, and M. Inostroza. Join point interfaces for safe and flexible decoupling of aspects. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(1):7, 2014.

[6] S. Efftinge and M. Völter. oAW xText: A framework for textual DSLs. In *Eclipse Modeling Symposium, Eclipse Summit Europe 2006*, Esslingen, Germany, Oct. 2006.

[7] J. Fabry, T. Dinkelaker, J. Noyé, and E. Tanter. A taxonomy of domain-specific aspect languages. *ACM Computing Surveys (CSUR)*, 47(3), Apr. 2015.

[8] M. Fowler. Language workbenches: The killer-app for domain specific languages, 2005.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing. Addison-Wesley, 1995.

[10] A. Hadas and D. H. Lorenz. Demanding first-class equality for domain specific aspect languages. In *Companion to the 14th International Conference on Modularity (Modularity'15 Comp.)*, pages 35–38, Fort Collins, CO, USA, Mar. 2015. ACM Press. Position paper.

[11] K. Hoffman and P. Eugster. Bridging Java and AspectJ through explicit join points. In *Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java (PPPJ'07)*, pages 63–72, Lisboa, Portugal, Sept. 2007. ACM.

[12] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. An overview of AspectJ. In *Proceedings of the 15th European Conference on Object-Oriented Programming (ECOOP'01)*, number 2072 in Lecture Notes in Computer Science, pages 327–353, Budapest, Hungary, June 2001. Springer Verlag.

[13] S. Kojarski and D. H. Lorenz. Awesome: An aspect co-weaving system for composing multiple aspect-oriented extensions. In *Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'07)*, pages 515–534, Montreal, Canada, Oct. 2007. ACM Press.

[14] C. V. Lopes. *D: A Language Framework for Distributed Programming*. PhD thesis, Northeastern University, 1997.

[15] D. H. Lorenz. Language-oriented modularity through Awesome DSALs: summary of invited talk. In *Proceedings of the 7th AOSD Workshop on Domain-Specific Aspects Languages (DSAL'12)*, pages 1–2, Potsdam, Germany, Mar. 2012. ACM Press.

[16] D. H. Lorenz and O. Mishali. SpecTackle: Toward a specification-based DSAL composition process. In *Proceedings of the 7th AOSD Workshop on Domain-Specific Aspects Languages (DSAL'12)*, Potsdam, Germany, Mar. 2012. ACM Press.

[17] M. Shonle, K. Lieberherr, and A. Shah. XAspects: An extensible system for domain specific aspect languages. In *Companion to the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 28–37, Anaheim, California, 2003. ACM Press.

[18] É. Tanter. Aspects of composition in the Reflex AOP kernel. In *Proceedings of the 5th International Symposium on Software Composition (SC'06)*, number 4089 in Lecture Notes in Computer Science, pages 98–113, Vienna, Austria, Mar. 2006. Springer Verlag.