**SPECTACLE: Toward a Specification-based DSAL Composition Process**

David H. Lorenz  
Oren Mishali

Open University of Israel,  
1 University Rd., P.O.Box 808, Raanana 43107 Israel  
{lorenz,omishali}@openu.ac.il

---

**Abstract**

DSAL composition frameworks are tools used in the process of composing multiple DSAL mechanisms into a single multi-D SAL weaver. The DSAL composition process starts with specifying the desired interactions between the DSAL mechanisms being composed, and concludes with producing a multi-D SAL weaver which satisfies the composition specification. However, the lack of tool support for defining the composition specification, and the coding effort required in composition frameworks to implement the specification, make this process complex and error prone.

This work presents a specification-based approach to DSAL composition. The approach is based on having a specification manifest file for the composition and for each of the individual mechanisms involved. A novel tool, named SPECTACLE, analyzes the manifests and helps the composition designer define the desired specification. Based on the composition specification produced, the composition framework can generate a significant part of the implementation code for the mechanisms and for the multi-D SAL weaver. The specification-based DSAL composition process is illustrated in the context of the AWESOME composition framework.

**Categories and Subject Descriptors**  
D.2.1 [Software Engineering]: Requirements/Specifications—Tools; D.3.3 [Programming Languages]: Language Constructs and Features—Frameworks.

**General Terms**  
Design, Languages.

**Keywords**  
Domain Specific Aspect Languages (DSALs).

---

1. **Introduction**

DSAL composition refers to a process in which a multi-D SAL weaver is composed out of individual DSAL mechanisms [3]. The DSAL composition process begins with defining a composition specification that determines how the DSALs interact. The process ends with an implementation of a multi-D SAL weaver capable of weaving code written in these multiple DSALS [8]. In this process, the role of the composition designer is to write the composition specification. The role of the composition implementer is to realize the multi-D SAL weaver according to the specification.

The DSAL composition process is aided by composition frameworks (such as AWESOME [5]). However,

- composition frameworks do not assist the composition designer in formulating a desired composition specification;
- composition frameworks offer the composition implementer only limited assistance in coding the composition specification.

In a realistic setting, where multiple DSALS are being combined, there is a need for also the definition of the composition specification to be guided by the composition framework. For instance, tools for inspecting basic composition properties and for investigating potential feature interactions should be a part of the service that the composition framework offers the designer.

Even the service that is provided to the composition implementer by composition frameworks leaves much to be desired. Typically, the specification is implemented imperatively rather than declaratively. This makes the composition process less intuitive and demands a greater coding effort. For example, in AWESOME, the composition configuration logic is coded by implementing an aspect in ASPECTJ (or several aspects). This is a significant improvement over a tangled crosscutting implementation, but is still a matter of writing code, not a specification.

Moreover, composition frameworks deal mainly with composition of existing mechanisms. The task of implementing new DSALs remains outside their scope. Therefore,
also here, due to a tool gap, it is difficult to implement a DSAL mechanism [4].

In this paper we present a specification-based DSAL composition process. The process is illustrated in the context of the AWESOME composition framework. The process features:

- **A formal composition specification.** The composition specification, which in state-of-the-art composition frameworks is informal, is made formal.
- **An explicit description of an aspect mechanism.** The properties characterizing each DSAL mechanism are made explicit in a dedicated manifest file.
- **A tool for analyzing the composition.** A new prototyped tool, named SPECTACLE, reads the manifests and assists the composition designer in analyzing the interactions and then in generating the composition specification.
- **A generative specification-based implementation.** The composition specification with the DSAL manifests are read by AWESOME, allowing part of the code of the individual DSAL mechanisms and of the composition configuration to be automatically generated.

The result is an improved DSAL composition process. The composition process becomes more effective and more affordable. Another advantage of a specification-based composition process is support for reasoning about the composition. The specification expresses the composition logic in high-level terms, making it more accessible to the development team. This is important, especially with domain experts that are not comfortable with code-level reasoning.

**Outline.** In Section 2 we demonstrate the SPECTACLE tool and illustrate how it helps the composition designer analyze and specify the desired DSAL interactions. In Section 3 we explain how AWESOME uses the produced specification to simplify the implementation of a multi-DSAL weaver.

## 2. A SPECTACLE Tool

SPECTACLE is a tool for analyzing a multi-DSAL composition as well as specifying the desired feature interactions. When initially tackling a multi-DSAL composition, and also when incrementally adding a new DSAL to an existing composition, many questions regarding the nature of the composition need to be answered. For example:

- Which mechanisms already participate in the composition?
- What join points in an input program may each mechanism affect?
- Are there join points that may be affected by multiple mechanisms, and if so, in what manner?
- Is there any order that is present between advice of existing mechanisms?

SPECTACLE is a command-line tool that helps answer such questions. The input to SPECTACLE is a set of manifest files characterizing the DSAL mechanisms that participate in the composition. Each manifest is defined by the provider of the corresponding DSAL, and contains particular properties of the DSAL mechanism, e.g., the Id of the mechanism, the kind of join points that it may affect, and the types of advice that the mechanism introduces. The intended user of SPECTACLE is the composition designer.

### 2.1 Basic Exploration of the Composition

To demonstrate the basic function of the SPECTACLE tool, consider a specific multi-DSAL composition for which the composition designer needs to resolve emerging feature interactions.

The composition designer starts by exploring the basic properties of the composition. Essentially, this means to identify the DSAL mechanisms that take part in the composition, and to query for the basic properties of each participating mechanism. The command `mech` presents a list of all the mechanisms in the composition. For our specific composition, `mech` lists three mechanisms:

```
spectackle> mech
  validate
cool
aspectj
```

**VALIDATE** [1] is a simple DSAL that supports validation of input parameters of methods, constructors and fields (field assignments). **COOL** [7] is a DSAL that handles synchronization of JAVA methods, and **ASPECTJ** [2] is a general purpose aspect language.

The composition designer continues to explore for the properties of each mechanism, beginning with a basic exploration of the COOL mechanism:

```
spectackle> adv cool
  lock → before
  unlock → after
spectackle> gran cool
  method-invocation → execution(method)
```

The first command in the transcript, `adv cool`, lists the advice types that COOL defines. Each line in the output refers to a single advice type, where the left-hand side is the name of the advice type in the terminology of COOL, and the right-hand side is the normalized advice type. In AWESOME, all advice types are normalized to a common base.

The second command in the transcript, `gran cool`, shows the granularity [6] of the COOL mechanism, i.e., the kinds of join point computations in the base system that COOL may affect (advise). The left-hand side of the
output describes the join point computation in a platform-independent fashion. The right-hand side is the mapping to a normalized join point model defined by AWESOME. Overall, the output implies that COOL may affect the behavior of the program by inserting lock and unlock advice before and after method executions, respectively.

2.2 Exploring and Configuring Co-Advising

After gaining a general understanding of the mechanisms participating in the composition, the composition designer proceeds with investigating the possible interactions between them. Each mechanism introduces one or more advice types. In our example, there are six advice types (three of ASPECTJ, two of COOL, and one of VALIDATE). The number of advice types may significantly increase as new DSALs are added to the composition. Naturally, advice belonging to different DSAL mechanisms may operate at the same join point. This kind of interaction is called co-advising [9].

SPECTACLE provides means for exploring and configuring the co-advising in a composition. The composition designer may investigate the co-advising by issuing the command `jp base -adv`. The output of is shown in Figure 1.

Each section in the figure lists the advice types that may surround a join point of a particular kind. The first section shows the advice that may be applied at any join point of kind `call(method)`. The possibility of a before advice is expressed by printing a line before the join point, around advice in the same line of the join point, and after advice are shown in the line after the join point. Since, in our composition, `call(method)` join points are neither in the granularity of COOL nor of VALIDATE, only ASPECTJ advice may affect method calls.

The second section in Figure 1 is more informative. Here, all the possible advice types surround the `execution(method)` join point kind. This indicates that all of them may be applied at join points of this kind. Note that the advice that operate before and after the join point appear in red (sans serif font). It indicates that no order is specified for these particular advice, which means that their execution order is arbitrary. This, of course, may be undesired. For instance, if an aspectj.before advice is executed before a cool.lock advice, then code executed within aspectj.before is not synchronized. If the code accesses shared application resources, this may lead to incorrect behavior.

Therefore, all sections in Figure 1 with red advice (three in our case) indicate a possible conflict that should be resolved with the SPECTACLE command `adv set`. The command supports resolution of advice ordering conflicts. For instance, by running the set of commands in Figure 1, the composition designer sets an ordering for before and after advice, and then verifies the result.

The `jp` command with the `~kind` flag lists the co-advising information for a specific kind of join point. We can see that now the before and after advice are printed in green (typewriter font), which means that an advice order was defined, and the order that they appear is their order of execution.

3. From Specification to Implementation

SPECTACLE supplies AWESOME with the composition specification and with the manifest files of the DSAL mechanisms that participate in the composition. This enables AWESOME to generate part of the multi-DSAL weaver code that would otherwise be coded manually by the composition implementer. In this section, we explain why a specification-based composition approach reduces significantly the manual coding effort.

```
spectackle> jp base -adv
aspectj.before
  call(method)  aspectj.around
  aspectj.after
aspectj.before  cool.lock  validate.validate
  execution(method)  aspectj.around
  aspectj.after  cool.unlock
aspectj.before
  call(constructor)  aspectj.around
  aspectj.after
aspectj.before  validate.validate
  execution(constructor)  aspectj.around
  aspectj.after
aspectj.before
  initialization
  aspectj.after
aspectj.before
  preinitialization
  aspectj.after
aspectj.before
  staticinitialization
  aspectj.around
  aspectj.after
aspectj.before
  get(field)  aspectj.around
  aspectj.after
aspectj.before  validate.validate
  set(field)  aspectj.around
  aspectj.after
aspectj.before
  handler
```

Figure 1: Co-advising weaving schedule for a composition of ASPECTJ, COOL, and VALIDATE.
3.1 Configuring Advice Order

In Section 2, the composition designer specified in SPECTACLE an order between pieces of before advice: a validate advice is executed first, followed by a COOL lock advice, and eventually by ASPECTJ before advice. It was also specified that ASPECTJ after advice should precede COOL unlock advice. The corresponding entries that SPECTACLE creates in the composition specification file are:

```
before-advice-order: validate.validate, cool.lock, aspectj.before
after-advice-order: aspectj.after, cool.unlock
```

Provided with this specification, AWESOME is able to automatically generate the aspects that advise the multi-DSAL weaver code and configure the specified advice order.

Figure 2 shows a simplified version of an ASPECTJ aspect that configures the before advice order. The aspect advises the multiOrderBefore method, which is a part of the AWESOME weaving process. The method is provided with multiEffects, a list of all before advice (effects) that are going to be woven at a specific join point shadow. Each element in the list is in itself a list, holding the effects of a particular mechanism. The method extracts all the effects from the inner lists and returns a single flattened list of effects. The aspect ensures that the advice are ordered according to their specified execution order.

The code in Figure 3 is not overly complicated, but still requires ASPECTJ coding skills and basic understanding of the AWESOME weaving process, including knowledge of low-level APIs (e.g., the BcelShadow class). Hence, an automatic generation of the aspect saves time and effort. Moreover, the specification is both explicit and precise, and thus promotes reasoning and communication.

4. Related Work

Similar to SPECTACLE, the Reflex composition framework supports the detection of co-advising interactions. The Reflex runtime is provided with a base system and with aspects of multiple DSALs, and translates each aspect to a common intermediate representation (Reflex API calls). The translation is handled by the appropriate DSAL plug-in. The common representation allows Reflex to detect co-advising interactions. The user is able to define composition rules for the interactions. If a composition rule is missing, the user is notified.

However, there are also significant differences between the approach presented here and that of Reflex. First, Reflex operates at the aspect level, detecting and resolving the interactions in a particular application. AWESOME, on the other hand, operates at the language level. The SPECTACLE tool helps to identify and resolve the interactions between the DSAL mechanisms, hence affecting the behavior of all derived multi-DSAL programs.

Second, the composition rules in Reflex are expressed imperatively in JAVA. The specification we discuss here is declarative and expressed in higher-level notations which are more intuitive.

Third, SPECTACLE allows to query composition properties other than co-advising interactions, e.g., the mechanisms that participate in the composition, basic mechanism properties, and the advice order that is currently set.

5. Conclusion

In this work we describe an improved process for composing multiple DSALS and illustrate it in the context of the AWESOME composition framework. The process is based on explicit specification manifests for each of the DSAL mechanism, and on the composition specification itself. The process begins with an analysis of the composition using a novel tool called SPECTACLE. The analysis helps the composition designer formulate the composition specification. AWESOME is provided with the specifications to facilitate automatic code generation.

The specification-based DSAL composition process was demonstrated by identifying and resolving co-advising interactions. Clearly, there are more kinds of interactions to (S)pectacle. We are currently working on extending SPECTACLE to support the detection and resolution of foreign advising interactions. Another topic for future work is to enhance the usability of the tool, e.g., by providing more sophisticated visualization.

DSAL composition frameworks make the development of applications using multiple DSALS possible. Yet, the complexity of the composition process hinders the adoption of the approach. We hope that a specification-based composition process, with the appropriate tool support, has the potential to make the multi-DSAL development more practical and more accessible.
public aspect BeforeAdviceOrderConfig {
  List around(MultiMechanism mm, List multiEffects, BcelShadow shadow):
    execution(List MultiMechanism.multiOrderBefore(List, BcelShadow))
      && this(mm) && args(multiEffects, shadow) {

    int coolPos = mm.getMechanismPos(COOLWeaver.class);
    int ajPos = mm.getMechanismPos(AJWeaver.class);
    int validatePos = mm.getMechanismPos(ValidateWeaver.class);
    List<IEffect> result = new ArrayList<IEffect>();

    // multiEffects is a List of List<IEffect>
    List<IEffect> ajEffects = (List<IEffect>)multiEffects.get(ajPos);
    List<IEffect> coolEffects = (List<IEffect>)multiEffects.get(coolPos);
    List<IEffect> validateEffects = (List<IEffect>)multiEffects.get(validatePos);

    // setting the desired advice order
    result.addAll(validateEffects);
    result.addAll(coolEffects);
    result.addAll(ajEffects);

    return result;
  }
}

Figure 3: An ASPECTJ aspect configuring an order for before advice

References


