ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Directembedding: Concealing the Deep Embedding of DSLs

Ólafur Páll Geirsson

School of Computer and Communication Sciences

Semester Project

June 2015

Responsible
Prof. Martin Odersky
EPFL / LAMP

Supervisor
Vojin Jovanović
EPFL / LAMP

Abstract

Authors of embedded domain specific languages (DSLs) commonly struggle to find the right balance between the capability and usability of their DSL. On one hand, deeply embedded DSLs give great power to the DSL author but have a steep learning curve for end users. On the other hand, shallowly embedded DSLs are more limiting for the DSL author but offer a more familiar interface to the end users that enables them to quickly become productive with the DSL.

This report presents work on *Directembedding*, a Scala library to implement a thin user-friendly layer on top of an existing deeply embedded DSL[1]. The library accomplishes this by using annotations and macros, and requires little to no knowledge about the Scala reflection API. We used Directembedding to implement a front-end for the functional relational mapping library *Slick*. With a small amount of code and loose coupling from the underlying slick implementation, our front-end was able to support a large feature set of the Slick deep embedding.

1 Introduction

Domain specific languages (DSLs) provide a simple and high-level way for programmers to accomplish a domain specific task. DSLs differ from general purpose programming languages in the sense that they relieve the programmer from dealing with the low-level details of a problem at the price of restricted capabilities. One common use case for DSLs is to enable novice programmers and experts in fields outside of software development to become productive programmers.

One method to implement DSLs is to embed them inside a host language. This has the benefit that the DSL can leverage the facilities of the host language. The downside is that an embedded DSL has less flexibility to give arbitrary semantics to a given program. An embedded DSL (EDSL) must obey the host language's syntax and predefined behavior. EDSLs largely fall into two categories:

- *Shallowly embedded DSLs* offer an interface on top of values that are directly provided by the host language. In Scala, these are values such as `Int` and `String`. The benefit to shallow EDSLs is that they have a small learning curve for end users. The interface is familiar to programmers who already have some experience with the host language. The downside to shallow EDSLs is that they are inconvenient for the DSL author. The values in the DSL may have predefined behavior by the host language or third-party libraries. The DSL author must work around these limitations in order to give domain specific meaning to the programs in her DSL.

[1]Note. This work builds on a previous semester project on the Directembedding library

- *Deeply embedded DSLs* offer an interface through an intermediate representation (IR), designed by the DSL author. In Scala, this could be a type such as `Column[Int]` or `Column[String]` for a database DSL. The benefit to deep EDSLs is that they are convenient for the DSL author. The DSL author has full control over the IR, and can therefore give any meaning to programs which invoke operations on the IR. The downside to deep EDSLs is that they can have a steep learning curve for end users. The types in the IR and their behavior may be unfamiliar to the programmers even though they may have some experience with the host language. In a way, deep EDSLs are not too different from ordinary libraries in a general purpose programming language.

There is a clear trade-off between the two categories: end users like shallow EDSLs while DSL authors like deep EDSLs. Directembedding aims to please both parties. The DSL author can safely make her DSL deeply embedded and then use Directembedding to provide a shallow EDSL-like interface for end users.

The main contributions of the work presented in this report are the following:

- Extend previous work on the Directembedding library by adding the possibility to 1) override behavior of predefined and third party types 2) give arbitrary semantics to many standard Scala features 3) configure the reification of DSL programs. Moreover, much work has been put to improve the error messages generated by the library. This work is explained in Section 2.

- Do the first case study on the practical use of the Directembedding library. In under two weeks, we implemented *slick-direct*: a front-end for the `Query` API in the functional relational mapping library Slick. Slick-direct is under 300 lines of code and delegates all implementation logic to the underlying Slick API. Slick-direct supports query operations such as `map`, `flatMap`, `filter`, and `join` with greatly simplified type signatures compared to the lifted embedding in Slick. This work is covered in Section 3.

2 Directembedding

The architecture of the Directembedding library went through a major overhaul in this project. The reification has been extended with new annotations and new capabilities such as language virtualization. The reification is now highly customizable by the DSL author. The library also aims to provide useful error messages where possible.

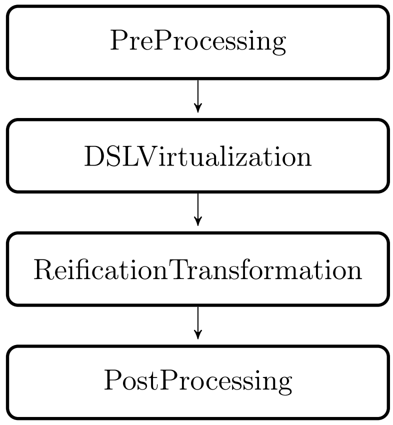

Figure 1: The Directembedding transformation pipeline.

2.1 Architecture

Figure 1 shows the new architechure of Directembedding. PreProcessing is an optional pass in the shallow embedding where the DSL author can transform the program in any way necessary before reification. PreProcessing requires knowledge of the Scala reflection API. DSLVirtualization reifies standard Scala language features according the configuration provided by the DSL author. This pass happens in the shallow embedding. ReificationTransformation is the major component of Directembedding and transforms the shallow embedding into the deep embedding. In this pass, the attached metadata to the shallow embedding is used to reify the program into the DSL authors IR. PostProcessing is an optional pass through the deep embedding where the DSL author can transform the program in any way necessary before the program is passed back to the user.

2.2 Language virtualization

Language virtualization is the process of converting standard language features into method calls. Such language features include if-then-else statements, loops, and variable assignments. It is generally not possible to override the semantics such features in a programming language.

Directembedding uses the language virtualization provided by the Yin-Yang [1] framework.

2.3 Overriding predefined and third-party types

2.4 Configurable reification

2.5 Improved error messages

- Explain `reifyAs` annotations, provide examples.

- Explain pipeline.

3 Case study: slick-direct

- Compare type signatures in `slick.direct` and `slick.lifted`, see below.

```
// slick.lifted
def map[F, G, T](f: E => F)
  (implicit shape: Shape[_ <: FlatShapeLevel, F, T, G]): Query[G, T, C]
// slick.direct
def map[F](f: E => F): Query[F, C]
```

```
// slick.lifted
def filter[T <: Rep[_]](f: E => T)(implicit wt: CanBeQueryCondition[T]): Query[E, U, C]
// slick.direct
def map[U](f: T => U): Query[U, C]
```

```
// slick.lifted
def joinFull[E1 >: E, E2, U2, D[_], O1, U1, O2](q2: Query[E2, _, D])
  (implicit ol1: OptionLift[E1, O1],
            sh1: Shape[FlatShapeLevel, O1, U1, _],
            ol2: OptionLift[E2, O2],
            sh2: Shape[FlatShapeLevel, O2, U2, _]): BaseJoinQuery[O1, O2, U1, U2, C, E1, E2]
// slick.direct
def joinFull[T2, D[_]](q: Query[T2, D]): BaseJoinQuery[Option[T], Option[T2], T, T2, C]
```

4 Related work

Yin-Yang.

5 Future research

- Complete slick-direct, an opinionated front-end for slick. Features include: custom types for primary keys and encrypted strings, customizable type provider, and implement remaining API.
- Explore alternative uses of directembedding.

References

[1] Vojin Jovanovic et al. "Yin-yang: concealing the deep embedding of DSLs". In: *Proceedings of the 2014 International Conference on Generative Programming: Concepts and Experiences-GPCE 2014*. ACM Press, 2014, pp. 73–82. URL: http://infoscience.epfl.ch/record/203432 (visited on 02/16/2015).