Direct Embedding

Joris Beau

EPFL - LAMP
Project in computer science
Directed by **Martin Odersky**, Supervised by **Vojin Jovanovic**

January 16, 2015

Motivation

As a DSL author...

Motivation

would you like to write this kind of code?

```
t.symbol.owner.isType
&& t.symbol.owner.asType == typeOf[ch.epfl.__.__.type].typeSymbol
&& x.symbol.name.toString == "query"
...
```

OR

```
case Query ...
```

Outline

Motivation

- Instigated by Slick database library
- Embedding DSLs simply!
- Provide a painless logic for reification
- Direct embedding has avantages:
  - better error messages
  - compile-time

Goal

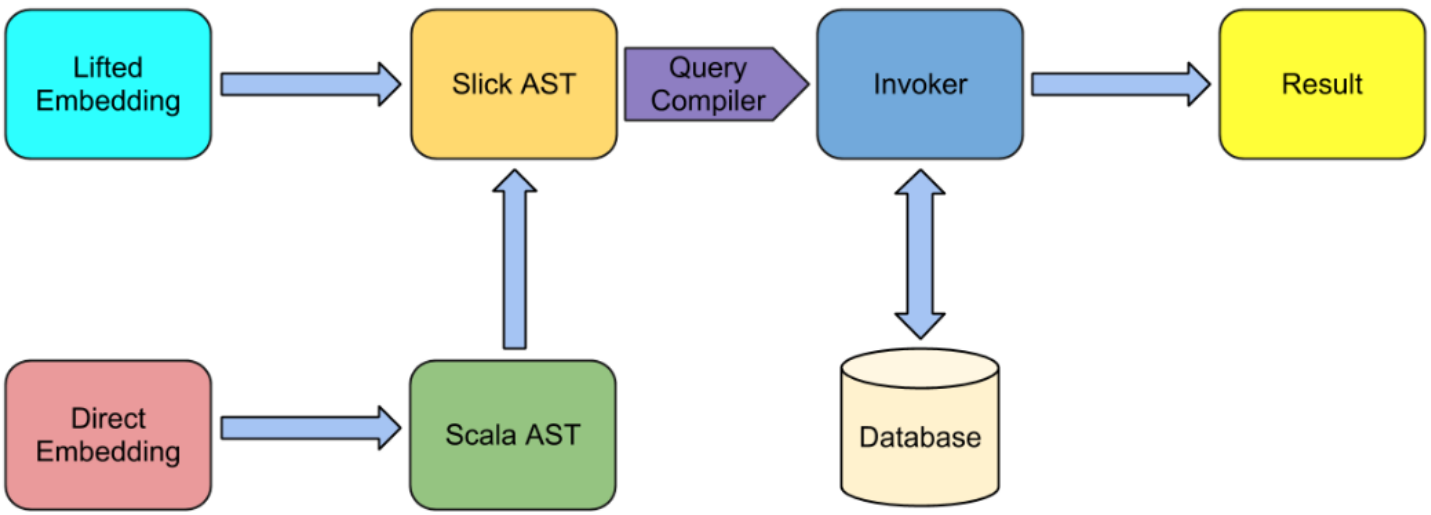
- Educational aspect
- Implement a core prototype
- Make it work & extend it
- Apply on real examples
- Might evolve to a complete library

Features

Users' friendly

- Scala-like language in Scala projects
- Take care of difficulties (macro)
- Provide easy and general tool
- Avoid complex and too specific code

Big Picture of Slick



Direct embedding aspects

- AST generated compile-time
- macro based
- Scala type
- at runtime, errors for unsupported methods
- experimental on Slick

Reification: Need to modify Scala AST

Recall: we want to avoid this kind of ugly and specific code

```
t.symbol.owner.isType
&& t.symbol.owner.asType == typeOf[ch.epfl.__.__.type].typeSymbol
&& x.symbol.name.toString == "query"
```

Reification: Need to modify Scala AST

What do we want, exactly?

- Write simple code,
- without knowledge about the DSL aspects,
- but that is for DSL

Let us write it and let someone else take care of details

Reification: Need to modify Scala AST

Let us thinking on an example:

```
def query[T](q:QueryableValue[T]) = ...
val coffees = Queryable[Coffee]
```

But here Coffee refers to a SQL table...
We need to specify it without knowledge about the domain and without huge overhead
Intermediate Representation (IR):

```
case class Coffee(...) // where Coffee will refer to SQL table
```

Later on we will examine the IR and transform the corresponding AST

Beginning with AST

How to modify Scala AST?

Quasiquote

How to modify Scala AST?
→ **quasiquote?**

No quasiquote

No quasiquote because of AST output is not fixe

but ...

Symbols

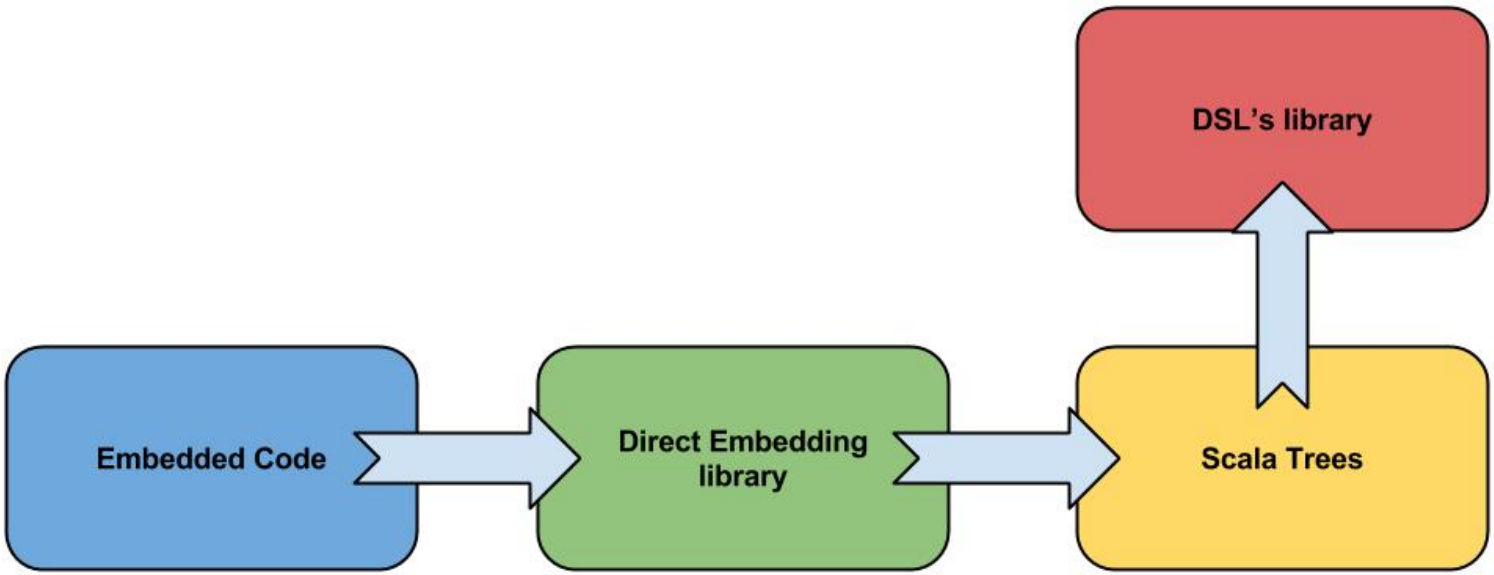
but

we can access to the symbols → @ANNOTATIONS

Annotation

```
@table("COFFEES") // the annotation
case class Coffee(  // the IR
...
)
```

Overview


DSL's library
Embedded Code
Direct Embedding library
Scala Trees

Macro

```
@table("COFFEES")
case class Coffee(
    @column("COF_NAME")
    name: String,
    @column("PRICE")
    price: Double
)
```

Tree

Apply(Select(New(Ident(ch.epfl.directembedding.test.ClassExample)), termNames.CONSTRUCTOR), List())

Result idea

From symbol, arguments, type, we can use the annotation to reify the tree:

Macro

$\Longrightarrow$ macro q"..."

Return

table(name: column[String], price: column[Double])

Summary of the example

- Users accordingly annotate DSLs expressions
- ASTs generated and transformed via macro
- After the modification, the tree contains the DSL representation .i.e. the annotation

Objects

Cases	Object	Nested	Classes
val value	✓	✓	✓
def foo	✓	✓	✓
def foo(args)	✓	✓	✓
def foo[T, U]: (T, U)	✓	✓	✓
def foo[T, U](t: T, u: U): (T, U)	✓	✓	✓
def foo[T](t_1: T)(...)(t_a: T)	✓	✓	✓

Language specification

if	X
while	X
do while	X
lazy val	X
return	X

Going further

- Operator
- Recursion
- Raw block of code

Demonstration

Thank you

Special thanks