



Domain Specific Languages

Walter Cazzola

Dipartimento di Informatica
Università degli Studi di Milano
e-mail: cazzola@di.unimi.it
twitter: @w_cazzola



Domain Specific Languages (DSLs)

Introduction

A **DSL** is a programming language that mimics the terms, idioms, and expressions used among the experts in the target domain

- ideally, a domain expert, with no experience in programming, can read, understand and validate such code

Benefits

- **Encapsulation** — A DSL hides the implementation details;
- **Productivity** — A DSL simplifies the coding as it uses the domain jargon;
- **Communication** — Non-programmers can be involved in the development;
- **Quality** — Minor “impedance mismatch” between domain experts’ requirements and the implementing code

Drawbacks

- difficulties of creating good DSLs and
- long-term maintenance



Domain Specific Languages (DSLs)

Internal vs External DSLs

DSLs are classified as:

- internal or embedded and
- external

Internal DSLs are an idiomatic way of writing code in a general purpose programming language

- no special-purpose parser is necessary
- internal DSLs are parsed as any other code written in the language

External DSLs are a custom language with its own custom grammar and parser

Comparison

- internal DSLs are easier to create than external one since they don’t require a special-purpose parser
- the constraints of the underlying language limit the options for expressing domain concepts.



Domain Specific Languages (DSLs)

Case Study

Problem: to create a payroll application that

- computes an employee’s paycheck every pay period (2 weeks long);
- the paycheck includes gross salary, net salary and deductions.

```
import payroll.api._
import payroll.api.DeductionsCalculator._
import payroll._
import payroll.Type2Money._

val buck = Employee(Name("Buck", "Trends"), Money(80000))
val jane = Employee(Name("Jane", "Doe"), Money(90000))

List(buck, jane).foreach { employee =>
  val biweeklyGross = employee.annualGrossSalary / 26.
  val deductions = federalIncomeTax(employee, biweeklyGross) +
    stateIncomeTax(employee, biweeklyGross) +
    insurancePremiums(employee, biweeklyGross) +
    retirementFundContributions(employee, biweeklyGross)

  val check = Paycheck(biweeklyGross, biweeklyGross - deductions, deductions)
  print(format("%s %s: %s\n", employee.name.first, employee.name.last, check))
}
```

Notes on the traditional solution

- it is noisy, e.g., it mentions employee and biweeklyGross incessantly;
- the code is imperative, with the DSL it will be more declarative.





Domain Specific Languages (DSLs)

Case Study: Under the Hood

Domain
Specific
Languages

Walter Cazzola

DSLs

Introduction
case study
embedded DSL

References

```
package payroll
case class Paycheck(gross: Money, net: Money, deductions: Money) {
  def plusGross (m: Money) = Paycheck(gross + m, net + m, deductions)
  def plusDeductions (m: Money) = Paycheck(gross, net - m, deductions + m)
}
```

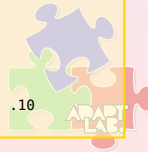
```
package payroll
case class Name(first: String, last: String)
case class Employee(name: Name, annualGrossSalary: Money)
```

```
package payroll
import java.math.BigDecimal

object Type2Money {
  implicit def bigDecimal2Money(b: BigDecimal) = Money(b)
  implicit def double2Money(d: Double) = Money(d)
  implicit def long2Money(l: Long) = Money(l)
  implicit def int2Money(i: Int) = Money(i)
}
```

```
package payroll.api
import payroll.Type2Money._
import payroll._

object DeductionsCalculator {
  def federalIncomeTax(empl: Employee, gross: Money) = gross * .25
  def stateIncomeTax(empl: Employee, gross: Money) = gross * .05
  def insurancePremiums(empl: Employee, gross: Money) = Money(500)
  def retirementFundContributions(empl: Employee, gross: Money) = gross * .10
}
```



Slide 5 of 12



Domain Specific Languages (DSLs)

Case Study: Under the Hood

Domain
Specific
Languages

Walter Cazzola

DSLs

Introduction
case study
embedded DSL

References

```
package payroll
import java.math.{BigDecimal, MathContext, RoundingMode}

class Money(val amount: BigDecimal) {
  def + (m: Money) = Money(amount.add(m.amount))
  def - (m: Money) = Money(amount.subtract(m.amount))
  def * (m: Money) = Money(amount.multiply(m.amount))
  def / (m: Money) = Money(amount.divide(m.amount, Money.scale, Money.roundingMode))
  def < (m: Money) = amount.compareTo(m.amount) < 0
  def <= (m: Money) = amount.compareTo(m.amount) <= 0
  def > (m: Money) = amount.compareTo(m.amount) > 0
  def >= (m: Money) = amount.compareTo(m.amount) >= 0
  override def hashCode = amount.hashCode * 31
  override def toString = String.format("%.2f", double2Double(amount.doubleValue))
  override def equals (o: Any) = o match {
    case m: Money => amount equals m.amount
    case _ => false
  }
}

object Money {
  def apply(amount: BigDecimal) = new Money(amount)
  def apply(amount: Double) = new Money(scaled(new BigDecimal(amount)))
  def apply(amount: Long) = new Money(scaled(new BigDecimal(amount)))
  def apply(amount: Int) = new Money(scaled(new BigDecimal(amount)))
  def unapply(m: Money) = Some(m.amount)

  protected def scaled(d: BigDecimal) = d.setScale(scale, roundingMode)
  val scale = 4; val roundingMode = RoundingMode.HALF_UP
  val context = new MathContext(scale, roundingMode)
}
```



Slide 6 of 12



Domain Specific Languages (DSLs)

Case Study

Domain
Specific
Languages

Walter Cazzola

DSLs

Introduction
case study
embedded DSL

References

Apart the payroll package that

- represents the "under the hood" of our paycheck application and should be hidden to the domain experts
- the payroll script using it is difficult to be interpreted from a domain expert to check its correctness.

What about something like?

```
Rules to calculate an employee's paycheck:
employee's gross salary for 2 weeks
minus deductions for
  federalIncomeTax, which is 25% of gross
  stateIncomeTax, which is 5% of gross
  insurancePremiums, which are 500. in gross's currency
  retirementFundContributions are 10% of gross
```

- this reads like normal English not as code
- it contains some "BUBBLE" words as "is", "which", ...
- it is less obscure since we minimized explicit references to contextual information



Slide 7 of 12



Domain Specific Languages (DSLs)

Embedded DSL

Domain
Specific
Languages

Walter Cazzola

DSLs

Introduction
case study
embedded DSL

References

```
import payroll._
import payroll.dsl._
import payroll.dsl.rules._

val payrollCalculator = rules { employee =>
  employee salary_for 2.weeks minus_deductions_for { gross =>
    federalIncomeTax is (25. percent_of gross)
    stateIncomeTax is (5. percent_of gross)
    insurancePremiums are (500. in gross.currency)
    retirementFundContributions are (10. percent_of gross)
  }
}

val buck = Employee(Name("Buck", "Trends"), Money(80000))
val jane = Employee(Name("Jane", "Doe"), Money(90000))

List(buck, jane).foreach { employee =>
  val check = payrollCalculator(employee)
  print(format("%s %s: %s\n", employee.name.first, employee.name.last, check))
}
```

Some notes

- infix operator notation
- implicit conversions and user-defined types
- apply methods



Slide 8 of 12



Domain Specific Languages (DSLs)

Embedded DSL (Cont'd)

Domain
Specific
Languages

Walter Cazzola

DSLs

Introduction
case study
embedded DSL

References

```
package payroll.dsl

case class Duration(val amount: Int) {
  def weeks = amount * 5
  def years = amount * 260
}

package payroll.dsl
import payroll._

object rules {
  def apply(rules: Employee => Paycheck) = new PayrollBuilderRules(rules)

  implicit def int2Duration(i: Int) = Duration(i)
  implicit def employee2GrossPayBuilder(e: Employee) = new GrossPayBuilder(e)
  implicit def grossPayBuilder2DeductionsBuilder(b: GrossPayBuilder) =
    new DeductionsBuilder(b)
  implicit def double2DeductionsBuilderDeductionHelper(d: Double) =
    new DeductionsBuilderDeductionHelper(d)
}

protected[dsl] class PayrollBuilderRules(rules: Employee => Paycheck) {
  def apply(employee: Employee) = {
    try { rules(employee) }
    catch {
      case th: Throwable => new PayrollException(
        "Failed to process payroll for employee: " + employee, th)
    }
  }
}
```



Slide 9 of 12



Domain Specific Languages (DSLs)

Embedded DSL (Cont'd)

Domain
Specific
Languages

Walter Cazzola

DSLs

Introduction
case study
embedded DSL

References

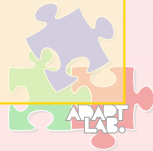
```
import payroll.Type2Money._

protected[dsl] class GrossPayBuilder(val employee: Employee) {
  var gross: Money = 0
  def salary_for(days: Int) = {
    gross += dailyGrossSalary(employee.annualGrossSalary) * days
    this
  }

  def weeklyGrossSalary(annual: Money) = annual / 52.0
  def dailyGrossSalary(annual: Money) = annual / 260.0
}

protected[dsl] class DeductionsBuilder(gpb: GrossPayBuilder) {
  val employee = gpb.employee
  var paycheck: Paycheck = new Paycheck(gpb.gross, gpb.gross, 0)
  def currency = this
  def minus_deductions_for(deductionRules: DeductionsBuilder => Unit) = {
    deductionRules(this)
    paycheck
  }

  def addDeductions(amount: Money) = paycheck = paycheck plusDeductions amount
  def addDeductionsPercentageOfGross(percentage: Double) = {
    val amount = paycheck.gross * (percentage/100.)
    addDeductions(amount)
  }
}
```



Slide 10 of 12



Domain Specific Languages (DSLs)

Embedded DSL (Cont'd)

Domain
Specific
Languages

Walter Cazzola

DSLs

Introduction
case study
embedded DSL

References

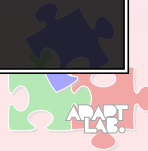
```
class DeductionCalculator {
  def is(builder: DeductionsBuilder) = apply(builder)
  def are(builder: DeductionsBuilder) = apply(builder)
  def apply(builder: DeductionsBuilder) = {}
}

object federalIncomeTax extends DeductionCalculator
object stateIncomeTax extends DeductionCalculator
object insurancePremiums extends DeductionCalculator
object retirementFundContributions extends DeductionCalculator

protected[dsl] class DeductionsBuilderDeductionHelper(val factor: Double) {
  def in (builder: DeductionsBuilder) = {
    builder addDeductions Money(factor)
    builder
  }

  def percent_of (builder: DeductionsBuilder) = {
    builder addDeductionsPercentageOfGross factor
    builder
  }
}
```

```
[16:31]cazzola@surtur:~/lp/scala/payroll-dsl>scala payroll-dsl.scala
Buck Trends: Paycheck($3076.92,$1346.15,$1730.77)
Jane Doe: Paycheck($3461.54,$1576.92,$1884.62)
```



Slide 11 of 12



References

Domain
Specific
Languages

Walter Cazzola

DSLs

Introduction
case study
embedded DSL

References

- ▶ Martin Odersky and Matthias Zenger.
Scalable Component Abstractions.
In Proceedings of OOPSLA'05, pages 41-57, San Diego, CA, USA,
October 2005. ACM Press.
- ▶ Nathanael Schärli, Stéphane Ducasse, Oscar Nierstrasz, and Andrew P. Black.
Traits: Composable Units of Behaviour.
In Proceedings of the ECOOP'03, LNCS 2743, pages 248-274,
Darmstadt, Germany, July 2003. Springer.
- ▶ Venkat Subramaniam.
Programming Scala.
The Pragmatic Bookshelf, June 2009.
- ▶ Dean Wampler and Alex Payne.
Programming Scala.
O'Reilly, September 2009.



Slide 12 of 12