High Performance DSL's in OCaml

Final Year Project - Midterm Evaluation

Rohit Mukherjee

November 27, 2014

Supervisor: Dr. Chin Wei Ngan
Project ID: H018570

Outline

Problem

Lack of mature frameworks/DSLs for system testing

- DSLs provide a high-level user-oriented view for software development.
- Strongly-typed functional languages, like Scala - great for writing DSLs
- Dearth of mature frameworks and tools available for system testing which can make the process painless and convenient.
- **Existing Frameworks: Not exactly suitable for system testing**

Existing Frameworks

JUnit

ScalaTest™
simply productive™

Cucumber

Solution
A DSL for System Testing

- In this project, some of the different ways scalable, high - performance DSLs could be written were examined and evaluated.
- Focus for this semester was to develop the various types and language of the DSL.
- DSL objectives and goals discussed next slide

DSL Requirements

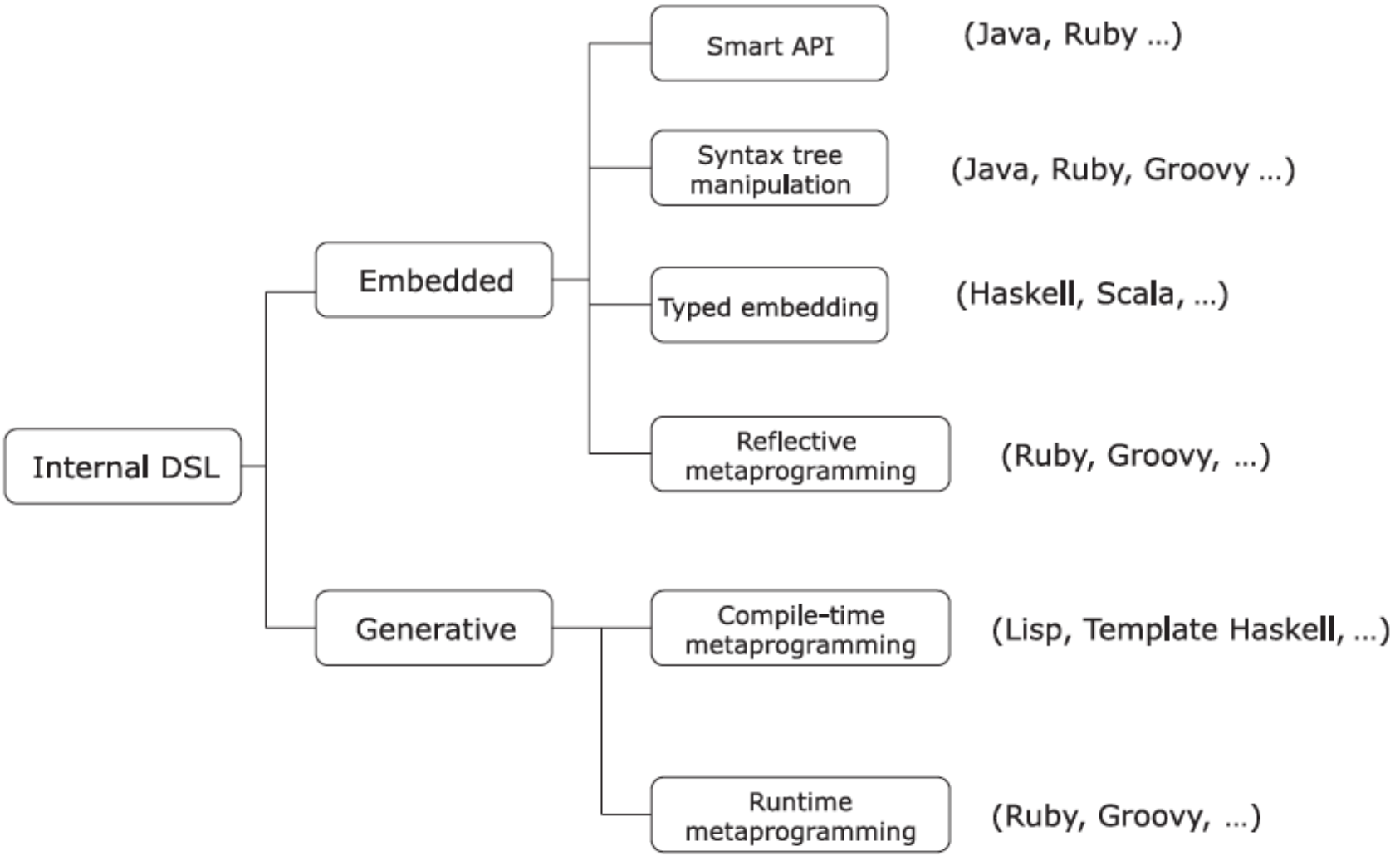
- Testing systems with inputs and matching against specified outputs
- Testing whether the system is functional as a whole and generates output
- Intuitive, declarative syntax for testers
- Ease of debugging and use
- Extensibility of the DSL
- Ease of integration with application code

1. Embedded DSLs in Scala

DSLs in Action, Ghosh, Manning Publications

Ghosh discusses two approaches to constructing internal DSLs - **Embedded** and **Generative**. Statically typed languages offer types as one of the means to abstract domain semantics and make the surface syntax of the DSL concise. Typed models come with a guarantee of some level of implicit consistency in the programming model.

Approaches to DSL Design



A simple JSON DSL

```
def main(args: Array[String]): Unit = {
    val json2 =  ::(::(
        MutableList(
            %("hello", "hello")),
            ->("car", "honda")),
            ->("car2","mercedes"))
        print(json2)
```

Figure: JSON DSL using type embedding

Link: https://github.com/rohitmukherjee/High-Performance-DSLs

2: Lightweight Modular Staging

Lightweight Modular Staging, Rompf & Odersky

Rompf and Odersky (2010) talk about an alternative approach to writing DSLs in Scala using a run - time code generation approach called lightweight modular staging. This approach involves both a generative and an embedded approach. The DSL is provided as a library and involves run - time code generation in different stages. The approach is called **Light - Weight Modular Staging (LMS)**.

LMS

```
class Vector[T:Numeric:Manifest](val data: Rep[Array[T]]) {
  def foreach(f: Rep[T] => Rep[Unit]): Rep[Unit] =
    for (i <- 0 until data.length) f(data(i))
  def sumIf(f: Rep[T] => Rep[Boolean]) = {
    var n = zero[T]; foreach(x => if (f(x)) n += x); n }
}

val v: Vector[Double] = ...
println(v.sumIf(_ > 0))
```

Figure: Code written using LMS

LMS

```
var n: Double = 0.0
var i: Int = 0
val end = data.length
while (i < end) {
  val x = data(i)
  val c = x > 0
  if (c) n += x
}
println(n)
```

Figure: Code at compile time

3: Delite, a framework for high performance DSLs

Delite, Odersky

A third approach to writing embedded, high - performance DSLs conducted by Odersky built upon the concept of using lightweight modular staging. The research resulted in a framework called Delite. Delite's compilation pipeline takes care of optimizing for target hardware such as multi - core processors, GPUs and computing clusters.

Delite Compilation Pipeline

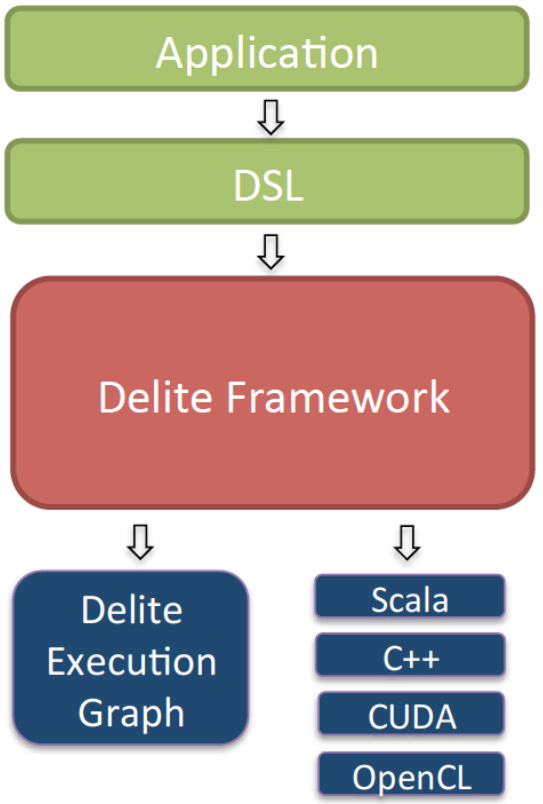


Figure: Delite compilation pipeline

Approach chosen for this semester

- Approach 1 of embedded DSL in Scala was chosen for this semester
- This was done to understand the various types required and implement functionality
- Performance optimizations will be considered in the next semester

Overview of Progress

- Types required to model the domain
- Basic DSL which can test any system/executable
- Declarative, natural language like syntax

```
1  new SleekTestCaseBuilder
2  runCommand "sleek"
3  onFile "/home/rohit/hg/sleek_hip/examples/working/sleek/sleek9.slk"
4  withArguments "--elp"
5  storeOutputInDirectory "results"
6  withOutputFileName "sleek9.out"
7  checkAgainst "Valid, Fail, Valid, Valid"
```

Figure: Code Snippet

Reasons for making certain design choices

- Internal DSL approach
- Choice of Scala as language of implementation
- Choice of certain design patterns

Functionality Completed

- Construct test cases for individual systems against expected output
- Run tests for all test files in a directory against previously generated output
- Custom matchers based on regex and diff
- Declarative syntax
- generate scripts to run test files in a directory

Thoughts on investigation

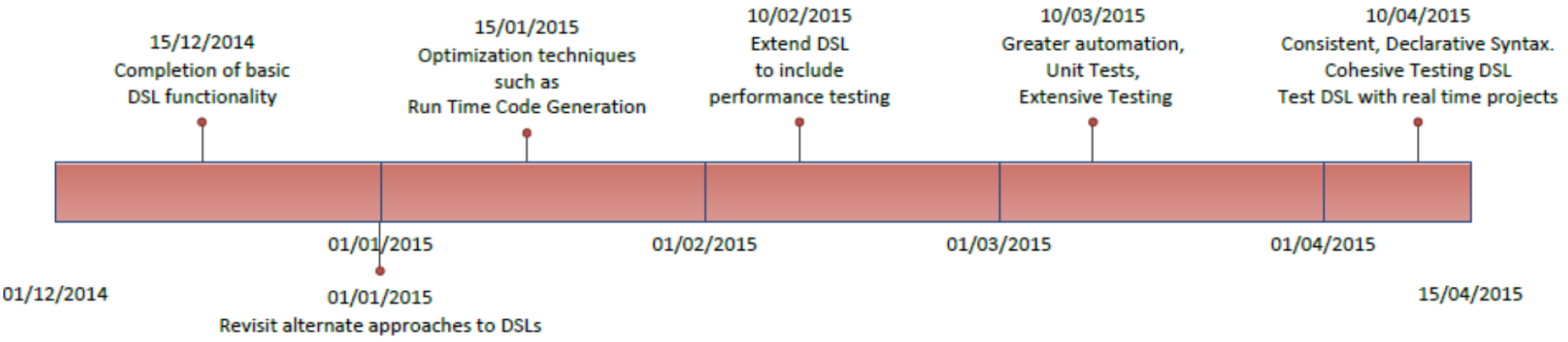
The following conclusions were drawn from the initial investigation:

- Scala provides a friendly ecosystem to write DSLs in many ways (as discussed)
- System Testing is a domain which requires more tooling/frameworks
- Ongoing research to optimize Scala for DSLs
- Design Patterns yield clean, reusable syntax and API

Research Plan

- Complete DSL feature list
- Features to provide greater automation/unit testing
- Lower level optimizations, revisit Delite and LMS
- Extend DSL to include performance testing
- Make DSL easy to use with real projects

Research Plan



Summary & QA

- DSL for system testing using Scala.
- Aim to make it an extensible system testing DSL
- Provide high performance through LMS/Delite like optimizations in the next semester.