Solving the TTC'15 Train Benchmark Case Study with SIGMA

Filip Křikava

Facutly of Information Technology
Czech Technical University
filip.krikava@fit.cvut.cz

CZECH TECHNICAL UNIVERSITY IN PRAGUE

FIT

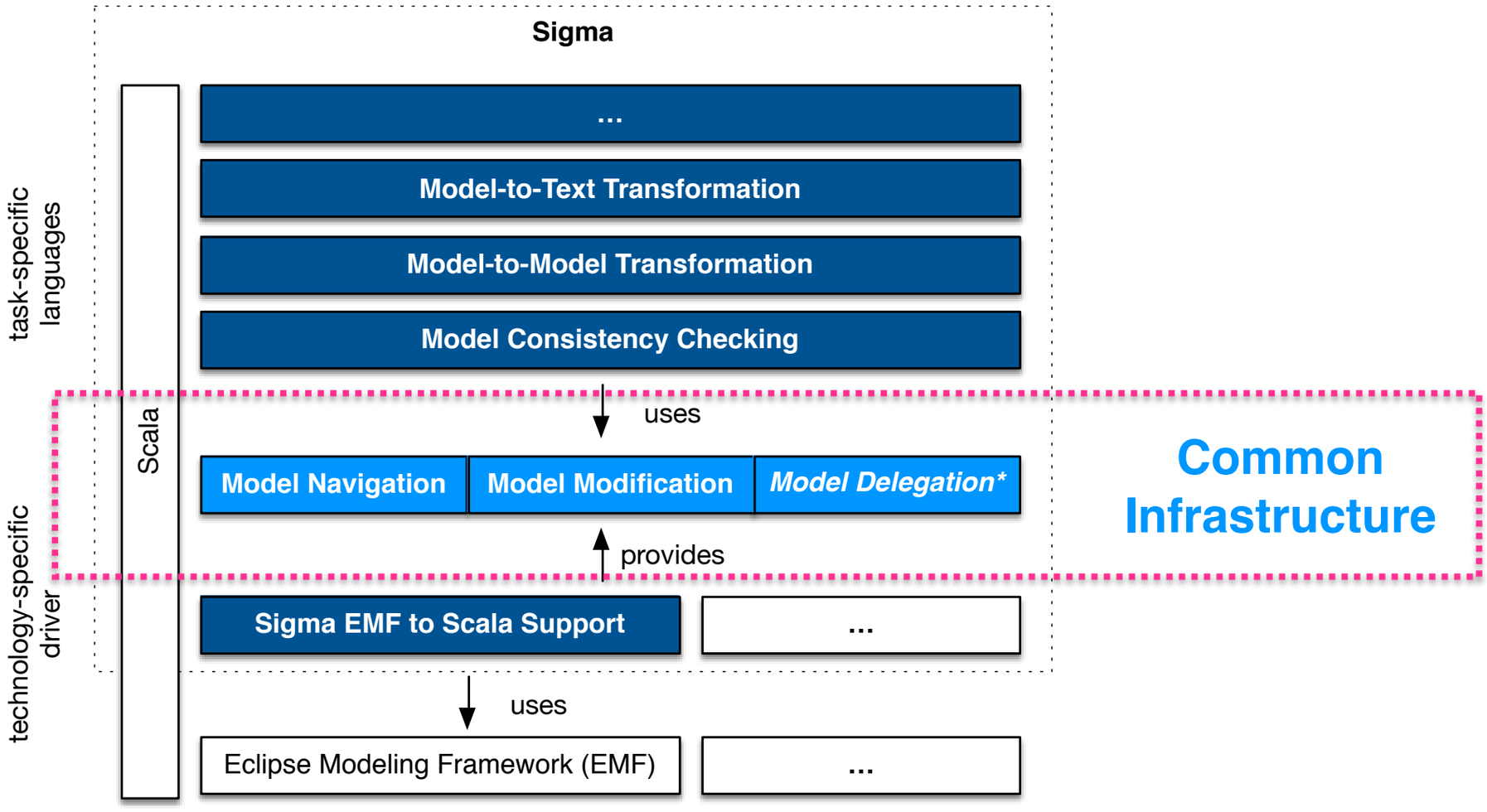
Solution Overview

- Solution for the train benchmark case study
- SIGMA as the transformation tool
- **Approach**

1. Developed Query and Repair Transformation internal *(tiny)* DSL on the top of SIGMA
2. Used the DSL to implement case study constraints
3. Integrated the DSL into the provided framework

https://github.com/fikovnik/trainbenchmark-ttc

SIGMA[1] Overview



Scala Internal Domain-Specific Languages for practical model manipulations within a familiar environment with improved usability and performance.

[1)] F. Křikava, P. Collet, R. France, *SIGMA: Scala Internal Domain-Specific Languages for Model Manipulations*, In Proceedings of the 17th International Conference on Model Driven Engineering Languages and Systems (MODELS '14), 2014

Internal *(embedded)* **DSL for model manipulation**

- **External** model manipulation DSLs
  - embeds a GPL into a model-manipulation DSL
- **Internal** model manipulation DSLs
  - embeds model manipulation DSL into a GPL (*a host language*)
  - with
    - increased level of abstraction
    - similar features, expressiveness and versatility
    - improved tool support, interoperability and performance
    - low engineering cost

but all depends on the host language

- Mixes function programming with OOP
- Modern GPL
- JVM compatible
- Designed to host DSLs
- Statically typed with type inference
- Well supported by major tool vendor

---

http://www.scala-lang.org

Query and Repair Transformation internal *(tiny)* DSL

- Domain concepts (from the description):
  - Constraint
    - Query - finds all instances violating a model restriction
    - Repair transformation - corrects wrong instances

```
case class Constraint[A <: EObject, B <: AnyRef](
  query: (A) => Iterable[B],
  repair: (B) => Unit
)
```

```
case class BooleanConstraint[A <: EObject](
  query: (A) => Boolean,
  repair: (A) => Unit
)
```

Query and Repair Transformation DSL

in Action

Task #1 - PosLength

- **Query:** The query checks for segments with a length less than or equal to zero.
- **Match:** ⟨segment⟩
- **Repair transformation:** The length attribute of the segment in the match is updated to –length + 1.

```
val PosLength = BooleanConstraint[Segment](
  query = segment => segment.length < 0,
  repair = segment => segment.length += -segment.length + 1
)
```

Task #2 - SwitchSensor

- **Query:** The query checks for switches that have no sensors associated with them.
- **Match:** ⟨sw⟩
- **Repair transformation:** For a given match, a sensor is created and connected to the switch.

```
val SwitchSensor = BooleanConstraint[Switch](
  query = switch => switch.sensor.isEmpty,
  repair = switch => switch.sensor = Sensor()
)
```

Query and Repair Transformation DSL

in Action

Task #3 - SwitchSet

- **Query:** The query checks for routes which have a semaphore that show the GO signal. Additionally, the route follows a switch position (swP) that is connected to a switch (sw), but the switch position (swP.position) defines a different position from the current position of the switch (sw.currentPosition).

- **Match:** ⟨semaphore, route, swP, sw⟩

- **Repair transformation:** For a given match, the currentPosition attribute of the switch is set to the position of the switchPosition.

```
Constraint[SwitchPosition, (Semaphore, Route, SwitchPosition, Switch)](
  query = swP => {
    for {
      // faulty meta-model - nullable entry should have cardinality [0..1]
      semaphore <- Option(swP.route.entry) if semaphore.signal == Signal.GO
      sw = swP.switch if sw.currentPosition != swP.position
    } yield (semaphore, swP.route, swP, sw)
  },

  repair = {
    case (_, _, swP, sw) => sw.currentPosition = swP.position
  }
)
```

Query and Repair Transformation DSL

in Action

Extension Task #1 - RouteSensor

- **Query:** The query looks for sensors that are connected to a switch, but the sensor and the switch are not connected to the same route.

- **Match:** ⟨route, sensor, swP, sw⟩

- **Repair transformation:** For a given match, the missing definedBy edge is inserted by connecting the route in the match to the sensor.

```
Constraint[Route, (Route, Sensor, SwitchPosition, Switch)](
  query = route => {
    for {
      swP <- route.follows
      sw = swP.switch
      sensor <- sw.sensor if !(route.definedBy contains sensor)
    } yield (route, sensor, swP, sw)
  },

  repair = {
    case (route, sensor, _, _) => route.definedBy += sensor
  }
)
```

Query and Repair Transformation DSL

in Action

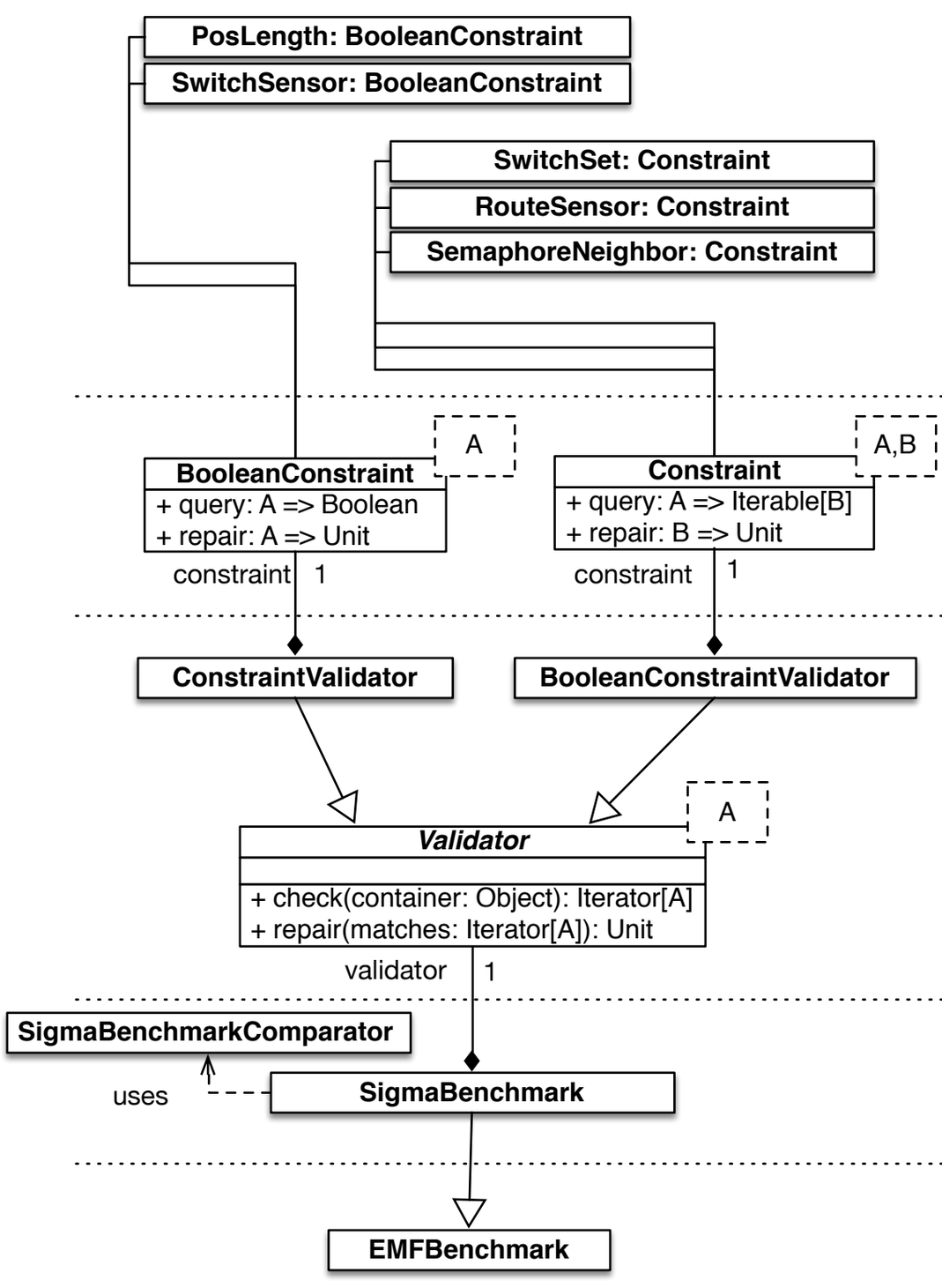
Extension Task #2 - SemaphoreNeighbor

- **Query:** The query checks for routes (route1) which have an exit semaphore (semaphore) and a sensor (sensor1) connected to a track element (te1). This track element is connected to another track element (te2) which is connected to another sensor (sensor2) which (partially) defines another, different route (route2), while the semaphore is not on the entry of this route (route2).
- **Match:** ⟨semaphore, route1, route2, sensor1, sensor2, te1, te2⟩
- **Repair transformation:** For a match, the route2 node is connected to the semaphore node with an entry edge.

```
Constraint[Route, (Semaphore, Route, Route, Sensor, Sensor, TrackElement,
                   TrackElement)](
  query = route1 => {
    for {
      sensor1 <- route1.definedBy if route1.exit != null
      te1 <- sensor1.elements
      te2 <- te1.connectsTo
      sensor2 <- te2.sensor
      route2 <- sensor2.sContainer[Route] if route1 != route2
      semaphore = route1.exit if semaphore != route2.entry
    } yield (semaphore, route1, route2, sensor1, sensor2, te1, te2)
  },

  repair = {
    case (semaphore, _, route2, _, _, _, _) => route2.entry = semaphore
  }
)
```

Putting it Together



Solution itself

syntax

DSL

semantics

Plumbing code

Case study

Assessment - Conciseness

- **Conciseness**
  - Solution: 52 SLOC
  - DSL: 20 SLOC
  - Plumbing code: 65 SLOC
- **Readability**
  - Significant improvement over Java RI
  - Type-safe (i.e. static verification)
  - Full featured editor (e.g., Eclipse, IntelliJ)
- **Performance**
  - Compiles into Java byte code
  - Similar performance to Java RI

Conclusions

- A concise solution that performs like the Java RI
- Uses SIGMA for EMF model querying and modification
- Builds on Scala*bility* idea - small compassable internal DSLs

https://github.com/fikovnik/trainbenchmark- ttc