One API to Rule Them All

Alois Cochard
@aloiscochar d

Ismael Juma
@ijuma
Agenda

- Motivation
- Shona
  - Graph DSLs
  - Entity Library
  - Execution Engine
- Future Work
- Questions
Motivation at Time Out

- 1 API (heterogeneous services) → N API/Feeds
- Simple Domain => Graph
- Feed ingestion
- Migrating new cities
- 1 DSL Describing domain Graph
  - Multiple sub-dsl for specific business needs
    - Feed ingestion
    - Feed output
    - Redirection
    - Cities migration (ETL)
Not just Time Out

- One or more applications in each device (desktops, laptops, phones, tablets, TVs, consoles…).
  - Reach as many users as possible.
  - In as many contexts as possible.
- For each device and use-case:
  - Optimise.
  - Support innovation.
API Progression

- Start with Single API
- Introduce ad-hoc ways to customise response
- Introduce device-specific endpoints
- Multiple Specialised APIs
Facebook Graph API

- Unified API
- Multiple graph queries can be “joined” in a single call
- Data hierarchy maintained in response

GET https://graph.facebook.com/me?fields=name,birthday,photos.limit(10).fields(id, picture)

Shona
Shona

film[="Empire of the Sun", "rating": 4]
(Label)

- Challenge
  - Data structure with dynamic set of fields (label, value)
  - Keep type safety and allow access like a case class
- How to define a label at type level?
  - "foo".type
- Macro to expose string value singleton type (shapeless)

```scala
scala> import shona._
import shona._

scala> implicitly[label("name") <-- String]
res4: <--[String("name"),String] = <function1>
```
Graph Model Language
Graph Model Language

- Internal DSL
- Can be traversed at run-time
- Can be traversed at compile-time
  - Encoding of the graph at type-level
  - Traversal currently implemented using macro
    - Should be possible to implement traversal using implicits only
      - ??? Performance of implicit resolutions ???
Graph Model Language – Defining vertices

```scala
val film = Vertex[label("film")] ~ (  
  int [label("id")],  
  string[label("name")],  
  string[label("directorId")]  
)

val director = Vertex[label("director")] ~ (  
  string[label("id")],  
  string[label("name")]  
)

val review = Vertex[label("review")] ~ (  
  int [label("filmId")],  
  int [label("rating")],  
  string[label("annotation")]  
)
```
val graph = Graph(film, director, review)(
    Edge[label("director")] ~ (film,
                           director,
                           Mapping.identity(string[label("directorId")], string[label("id")]))
),
    Edge[label("review")] ~ (film,
                           review,
                           Mapping.identity(int[label("id")], int[label("filmId")])))
)
Graph Model Language

- Property

```scala
class Property[N <: String, T](override implicit val label: Label[N])
```

- Vertex

```scala
class Vertex[N <: String, Properties <: HList : <=:[AnyProperty]#λ](
    val name: Label[N],
    val properties: Properties
)
```
class Edge[
    N <: String,
    VFN <: String, VFPL <: HList : <<:[AnyProperty]#λ, PFN <: String, PFT,
    VTN <: String, VTPL <: HList : <<:[AnyProperty]#λ, PTN <: String, PTT
]
{
    val name: Label[N],
    val from: Vertex[VFN, VFPL],
    val to: Vertex[VTN, VTPL],
    val mapping: Mapping[PFN, PFT, PTN, PTT]
}
Graph Model Language

```scala
class Graph[
  Vertices <: HList : <=: [AnyVertex]#λ,
  Edges <: HList : <=: [AnyEdge]#λ
](
  val vertices: Vertices,
  val edges: Edges
)
```
Graph Query Language
Graph Query Language

- **JSONQuery**
  - Similar XQuery/XPath but for JSON object graph
  - Currently only simple map operations are supported
- **Can be used internally**
  - By being parsed at compile-time using macro
- **Implemented using parser combinators**
  - Potential performance issues
Graph Query Language

- `film.name`
  - “Empire of the Sun”
- `film.director.id`
  - 1
- `film[={name, director.name}]`
  - (“Empire of the Sun”, “Steven Spielberg”)

```json
{
  film: {
    name: Empire of the Sun,
    director: { id: 1,
      name: Steven Spielberg
    }
  }
}
```
Entity Library

Graph Model

Execution Engine

Entities

JSONQuery
Entity Library

- Basically an entity is a HList of Field

```scala
class Entity[Fields <=: HList : <=:[Field[_, _]]#λ](
  val fields: Fields
)
```

- A Field is composed of a label (singleton type) and a value

```scala
class Field[N <=: String, T](
  val label: Label[N],
  val value: T
)
```

- Can be manipulated using shapeless
- If needed can be converted to a List (vs HList), to be dynamically manipulated at runtime
Entity Library

- Internal DSL

```scala
val entity = Entity(
  Field[label("id")].~1000L,
  Field[label("email")].~"alois.cochar@cmail.com"
)
```

- Can be accessed like a class by using a View

```scala
val view = View(entity)
view.id === 1000L
view.email === "alois.cochar@cmail.com"
```

No 22 limit!
Execution Engine

Diagram:
- Graph Model
- JSONQuery
- Execution Engine
- Entities
Execution Engine

- Very simple implementation
  - Only support 1 → 1 relationship between vertices
  - Read-only
  - No parameters support (pagination, ...)
  - Entity are loaded used in memory data-structure
object Model {
    val film = Vertex[label("film")][label("id")]
    string[label("name")],
    string[label("directorId")]
}

val director = Vertex[label("director")][label("id")]
    string[label("name")]

val review = Vertex[label("review")][label("filmId")]
    int[label("rating")],
    string[label("annotation")]

val graph = Graph(film, director, review)
    Edge[label("director")][label("id")]
    Mapping.identity(string[label("directorId")], string[label("id")]),
    Edge[label("review")][label("id")]
    Mapping.identity(int[label("id")], int[label("filmId")])
}
object Model {
  val film = Vertex[label("film")] ~ (  
    int [label("id")],  
    string[label("name")],  
    string[label("directorId")]
  )

  val director = Vertex[label("director")] ~ (  
    string[label("id")],  
    string[label("name")]
  )

  val review = Vertex[label("review")] ~ (  
    int [label("filmId")],  
    int [label("rating")],  
    string[label("annotation")]
  )

  val graph = Graph(film, director/*, review*/)(  
    Edge[label("director")] ~ (film, director, Mapping.identity(string[label("directorId")]), string[label("id")]),  
    Edge[label("review")], ~ (film, review, Mapping.identity(int[label("id")], int[label("filmId")]))
  )
}
object Model {
val film = Vertex[label("film")]) ~ (
    int [label("id")],
    string[label("name")],
    string[label("directorId")]
)

val director = Vertex[label("director")]) ~ (
    string[label("id")],
    string[label("name")]
)

val review = Vertex[label("review")]) ~ (
    int [label("filmId")],
    int [label("rating")],
    string[label("annotation")]
)

val graph = Graph(film, director, review)(
    Edge[label("director")]) ~ (film, director, Mapping.identity(string[label("don't exist!")]), string[label("id")]),
    Edge[label("review")]) ~ (film, review, Mapping.identity(int[label("id")], int[label("filmId")]))
}
object Provider {

  val films = Seq(
    Entity((Field[label("id")]) ~ 1, Field[label("name")] ~ 'In the Mood for Love', Field[label("directorId")] ~ "/m/01f7v_"),
    Entity((Field[label("id")]) ~ 2, Field[label("name")] ~ 'Empire of the Sun', Field[label("directorId")] ~ "/m/06pj8"))
  )

  val directors = Seq(
    Entity((Field[label("id")]) ~ "/m/01f7v_", Field[label("name")] ~ 'Wong Kar-wai'),
    Entity((Field[label("id")]) ~ "/m/06pj8", Field[label("name")] ~ 'Steven Spielberg'))
  )

  val reviews = Seq(
    Entity((Field[label("filmId")]) ~ 1, Field[label("rating")] ~ 4, Field[label("annotation")] ~ "TBD"),
    Entity((Field[label("filmId")]) ~ 2, Field[label("rating")] ~ 5, Field[label("annotation")] ~ "TBD"))
  )

  implicit val filmsLoader = EntityLoader.fromSeq(Model.film, films)
  implicit val directorsLoader = EntityFinder.fromFunction(Model.director, string[label("id")])((x) => directors.find(_.fields hx
  implicit val reviewsLoader = EntityFinder.fromFunction(Model.review, int[label("filmId")])((x) => reviews.find(_.fields.head.vx
}
REPL

scala> import shona.engine._
import shona.engine._
scala> import shona.entity._
import shona.entity._
scala> import shona.demo.Model.graph
import shona.demo.Model.graph
scala> import shona.demo.Provider._
import shona.demo.Provider._

scala> val xs = Execute(graph)("film")
x: Seq[shona.entity.Entity[shapeless.[shona.entity.Field[String("id"),Int],shapeless.[shona.entity.Field[String("name"),String],shapeless.[shona.entity.Field[String("directorId"),String],shapeless.HNil]]]] = List(Entity(id(1) :: name(In the Mood for Love) :: directorId(/m/81f7v_) :: HNil),
                    Entity(id(2) :: name(Empire of the Sun) :: directorId(/m/68pj8) :: HNil))

scala> val xs = Execute(graph)("film=[name, review.rating]")
x: Seq[shona.entity.Entity[shapeless.[shona.entity.Field[String("name"),String],shapeless.[shona.entity.Field[String("rating"),Int],shapeless.HNil]]]] = List(Entity(name(In the Mood for Love) :: rating(4) :: HNil), Entity(name(Empire of the Sun) :: rating(5) :: HNil))

scala> val x = View(xs.head)
x: shona.entity.views.View974A5F79F3CE127F5214EA6587BE4096 = View.Entity(name(In the Mood for Love) :: rating(4) :: HNil)

scala> x.name
res0: String = In the Mood for Love

scala> x.rating
res1: Int = 4
We need string singleton type!

- Without them no way of encoding label in types
- Currently implemented using type macros
- Could be exposed directly by the language
  - “foo”.type
Feed Ingestion
- Describe mapping between feed (JSON/XML) and model
  
  ```json
  {  
    movie_title: "Empire of the Sun," 
    production_year: 1987  
  }
  ```

![Diagram showing mapping between movie_title and film.name]
Time Out Integration

- Website redirects
  - Define mapping between slug and model

```
/movie/empire_of_the_sun → film(1)
/movie/in_the_mood_for_love → film(2)
```
What’s next

- Slick integration
  - Compose slick query from graph query
- Exposing the graph as a REST API
  - Automatically generate spray routing
What’s next

- Entity library
  - Integrate into Shapeless records (Serialisation for free)
- Advanced edge mapping support
  - Mapping as a function (= query parameters)
  - Pagination support
  - Create/Update/Delete
Thanks!

- We’re hiring!
- [http://github.com/aloiscochard/shona](http://github.com/aloiscochard/shona)

Alois Cochard  
@aloiskochar

d  
Ismael Juma  
@ijuma