Scarph: an implementation of the dagger categorical model for graph data

Alexey Alekhin[1] and Eduardo Pareja–Tobes[1]

[1] *oh no sequences!* research group, Era7 bioinformatics

October 5, 2015

Abstract

What is *Scarph*?
Baby, don't hurt me,
Don't hurt me no more!

Contents

Introduction

At the moment the standard way of accessing a graph database is either by using Blueprints Java API or one of the specialized graph query languages, such as Cypher or Gremlin. The common problem of all such approaches is that they don't take into account the graph schema on the language level. So using these query languages one can easily make a mistake and write a *syntactically correct* query, which doesn't make any sense from the point of view of the data model and therefore will *fail in runtime*.

As most of our work is concentrated on using Bio4j in the cloud distributed systems, it is really important to be able to formulate a query to the database in such way that will strictly conform to the data model, making the safety of evaluating this query known in advance.

While there are several frameworks to work with SQL databases from Scala *in a type-safe manner* (see ScalaQuery, Squeryl, Sqltyped and Slick), there is nothing similar for graph databases. Scala is our language of choice because of it's effective combination of powerful type system, functional programming paradigm and compatibility with Java. So we want to take advantage of the host language type system to be able to work with Bio4j in a way that is clearly and strictly determined by the graph schema. This the aim of the **Scarph** project which obviously target the second global objective of our project: to develop a new system to extract a particular type of data from the Bio4j graph database.

So, in short, **Scarph** is an API / DSL (domain specific language) for accessing graph DB and building queries in a type-safe manner.

Methods

Declaring a graph schema in **Scarph**

Properties

Vertex types

Edge types

Graph schema

Type wrapping and type denotation

Denoting graph types

Building queries

Backend implementations

Results

With this work we have shown that it is possible to create an Embedded Domain Specific Language in Scala for operating on a graph databases in a type-safe manner. **Scarph** is an extensible and flexible DSL, which allows one to define a graph schema and then use it for constructing compile-time safe queries, which is an essential requirement when it is used in highly scalable distributed systems.

As a proof-of-a-concept **Scarph** has an implementation for the **TitanDB** backend, though staying backend independent and easily extensible. Quoting the official website

> *Titan is a scalable graph database optimized for storing and querying graphs containing hundreds of billions of vertices and edges distributed across a multi-machine cluster. Titan is a transactional database that can support thousands of concurrent users executing complex graph traversals in real time.*

Also as the graph schema is defined separately from the implementation bindings, it is important to emphasize that the *queries code is actually independent* from the particular implementation used. This allows not only to reuse queries code easily, but also to rewrite queries to optimize them before execution.

During the development process **Scarph** is continually tested using the ScalaTest routine. These tests work not only as a guarantee of the library consistency, but also as an important part of documentation, containing a lot of examples, which could help a potential user to understand the way library works and apply it for their own needs.

Discussion

It worths mentioning that although currently **Scarph** has a backend implementation only for TitanDB, it was developed with implementation independence in mind. Therefore it is easy to add more implementations, making this library applicable for more use cases, when the backend is an important factor. Some examples of alternative implementations would be **Neo4j**, **OrientDB** and a generic **Gremlin**/**Tinkerpop3**. The last is actually not a particular database technology, but a special abstraction layer, which has already implementations for the most popular databases such as Neo4j and Giraph. Also, one of our Google Summer of Code projects was aimed to design a **Scarph** implementation for Amazon DynamoDB. It is called **Dynamograph** and is in the active development at the moment.

Different implementations lead not only to different use cases when the rest of the project is dependent on a database technology. It is also an opportunity to *mix different backends in one graph schema*. As every vertex or edge type has it's own `Raw` type binding and they are declared separately, one can create a schema, where some elements of the graph are stored in one database and others in another. As different databases use different low-level implementations for indexes, this could be a great opportunity to tune storage effectiveness and query performance.

Scarph is not a standalone graph query language like, for example, Gremlin, but rather an Embedded Domain Specific Language. It means that **Scarph** provides the means for building simple type-safe queries and it's possible to combine them using common Scala constructions. It is not a limitation, but a particularity of the use case that it is going to be applied: a Scala API for working with Bio4j.

Some links

1. Ian Robinson, Jim Webber, Emil Eifrém, Graph Databases: The Definitive Book on Graph Databases, O'Reilly 2013
2. Chad Vicknair, Michael Macias, et al., A Comparison of a Graph Database and a Relational Database
3. Peter T. Wood, Query Languages for Graph Databases
4. Salim Jouili, Valentin Vansteenberghe, An empirical comparison of graph databases
5. Marko A. Rodriguez, Peter Neubauer The Graph Traversal Pattern, CoRR 2010
6. Scala Query: A type-safe database API for Scala
7. Squeryl: A Scala ORM and DSL for talking with Databases with minimum verbosity and maximum type safety
8. sqltyped: Embedding SQL as an external DSL into Scala
9. A quick tour of relational database access with Scala
10. Slick: Functional Relational Mapping for Scala