The Path-o-Logical Gremlin

Marko A. Rodriguez
Graph Systems Architect

http://markorodriguez.com
@twarko

The 2nd International Workshop on Graph Data Management (GDM’11)
Hannover, Germany – April 16, 2011
Abstract

Gremlin is a graph traversal language that connects to various graph databases/frameworks.

- Neo4j [http://neo4j.org]
- OrientDB [http://orienttechnologies.com]
- DEX [http://www.sparsity-technologies.com/dex]
- OpenRDF Sail [http://openrdf.org]
- JUNG [http://jung.sourceforge.net]

This lecture addresses the state of Gremlin as of the 0.9 (April 16th, 2011).
TinkerPop Productions

1

To understand Gremlin, it's important to understand Groovy, Blueprints, and Pipes.
Gremlin is a Domain Specific Language

- Groovy is a superset of Java and as such, natively exposes the full JDK to Gremlin.

- Gremlin 0.7+ uses Groovy as its host language.²

- Gremlin 0.7+ takes advantage of Groovy’s meta-programming, dynamic typing, and closure properties.

Gremlin is for Property Graphs

1. name = "marko"  
age = 29  

2. name = "vadas"  
age = 27  

3. name = "josh"  
age = 32  

4. name = "vadas"  
age = 27  

5. name = "ripple"  
lang = "java"  

6. name = "peter"  
age = 35  

7. created  
weight = 1.0  

8. created  
weight = 1.0  

9. created  
weight = 0.4  

10. created  
weight = 1.0  

11. created  
weight = 0.4  

12. created  
weight = 0.2

knows

weight = 0.5

name = "lop"
lang = "java"
Gremlin is for Property Graphs

**VERTEX**
- **key/value**: name = "marko", age = 29

**EDGE**
- **key/value**: weight = 0.4

**EDGE LABEL**
- **key/value**: name = "lop", lang = "java"

**ID**
- 1
- 9
- created
Gremlin is for Blueprints-enabled Graph Databases

- Blueprints can be seen as the JDBC for property graph databases.\(^3\)
- Provides a collection of interfaces for graph database providers to implement.
- Provides tests to ensure the operational semantics of any implementation are correct.
- Provides support for GraphML and other “helper” utilities.

\(^3\)Blueprints is available at http://blueprints.tinkerpop.com.
A Blueprints Detour - Basic Example

Graph graph = new Neo4jGraph("/tmp/neo4j");
Vertex a = graph.addVertex(null);
Vertex b = graph.addVertex(null);
a.setProperty("name","marko");
b.setProperty("name","peter");
Edge e = graph.addEdge(null, a, b, "knows");
e.setProperty("since", 2007);
graph.shutdown();
A Blueprints Detour - Sub-Interfaces

- **TransactionalGraph** extends Graph
  - For transaction-based graph databases.\(^4\)

- **IndexableGraph** extends Graph
  - For graph databases that support the indexing of properties.\(^5\)

\(^4\)Graph Transactions [https://github.com/tinkerpop/blueprints/wiki/Graph-Transactions](https://github.com/tinkerpop/blueprints/wiki/Graph-Transactions).

\(^5\)Graph Indices [https://github.com/tinkerpop/blueprints/wiki/Graph-Indices](https://github.com/tinkerpop/blueprints/wiki/Graph-Indices).
A Blueprints Detour - Implementations

- TinkerGraph
- Neo4j the graph database
- OrientDB
- reXster
- openRDF.org
- sparsity technologies

performance in action
A Blueprints Detour - Implementations

- TinkerGraph implements IndexableGraph
- Neo4jGraph implements TransactionalGraph, IndexableGraph\(^6\)
- OrientGraph implements TransactionalGraph, IndexableGraph\(^7\)
- DexGraph implements TransactionalGraph, IndexableGraph\(^8\)
- RexsterGraph implements IndexableGraph: Binds to Rexster REST API.\(^9\)
- SailGraph implements TransactionalGraph: Turns any Sail-based RDF store into a Blueprints Graph.\(^{10}\)

---

\(^6\)Neo4j is available at [http://neo4j.org](http://neo4j.org).

\(^7\)OrientDB is available at [http://www.orientechnologies.com](http://www.orientechnologies.com).

\(^8\)DEX is available at [http://sparsity-technologies.com/dex](http://sparsity-technologies.com/dex).

\(^9\)Rexster is available at [http://rexster.tinkerpop.com](http://rexster.tinkerpop.com).

\(^{10}\)Sail is available at [http://openrdf.org](http://openrdf.org).
A Blueprints Detour - Ouplementations

openRDF.org

JUNG
Java Universal Network/Graph Framework

Blueprints
A Blueprints Detour - Ouplementations

- **GraphSail**: Turns any IndexableGraph into an RDF store.  

- **GraphJung**: Turns any Graph into a JUNG graph.  

---

Gremlin Compiles Down to Pipes

- Pipes is a data flow framework for evaluating lazy graph traversals.\(^\text{14}\)

- A Pipe extends Iterator, Iterable and can be chained together to create processing pipelines.

- A Pipe can be embedded into another Pipe to allow for nested processing.

\(^{14}\)Pipes is available at http://pipes.tinkerpop.com.
A Pipes Detour - Chained Iterators

- This Pipeline takes objects of type A and turns them into objects of type D.

```java
Pipe<A,D> pipeline =
    new Pipeline<A,D>(Pipe1<A,B>, Pipe2<B,C>, Pipe3<C,D>)
```
A Pipes Detour - Simple Example

“What are the names of the people that marko knows?”
A Pipes Detour - Simple Example

Pipe<Vertex,Edge> pipe1 = new OutEdgesPipe("knows");
Pipe<Edge,Vertex> pipe2 = new InVertexPipe();
Pipe<Vertex,String> pipe3 = new PropertyPipe<String>("name");
Pipe<Vertex,String> pipeline = new Pipeline(pipe1,pipe2,pipe3);
pipeline.setStarts(new SingleIterator<Vertex>(graph.getVertex("A"));

HINT: The benefit of Gremlin is that this Java verbosity is reduced to g.v('A').outE('knows').inV.name.
A Pipes Detour - Pipes Library

<table>
<thead>
<tr>
<th>FILTERS</th>
<th>GRAPHS</th>
<th>SIDE EFFECTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AndFilterPipe</td>
<td>OutEdgesPipe</td>
<td>AggregatorPipe</td>
</tr>
<tr>
<td>BackFilterPipe</td>
<td>InEdgesPipe</td>
<td>GroupCountPipe</td>
</tr>
<tr>
<td>CollectionFilterPipe</td>
<td>OutVertexPipe</td>
<td>CountPipe</td>
</tr>
<tr>
<td>ComparisonFilterPipe</td>
<td>InVertexPipe</td>
<td>SideEffectCapPipe</td>
</tr>
<tr>
<td>DuplicateFilterPipe</td>
<td>IdFilterPipe</td>
<td></td>
</tr>
<tr>
<td>FutureFilterPipe</td>
<td>IdPipe</td>
<td></td>
</tr>
<tr>
<td>ObjectFilterPipe</td>
<td>LabelFilterPipe</td>
<td></td>
</tr>
<tr>
<td>OrFilterPipe</td>
<td>LabelPipe</td>
<td></td>
</tr>
<tr>
<td>RandomFilterPipe</td>
<td>PropertyFilterPipe</td>
<td></td>
</tr>
<tr>
<td>RangeFilterPipe</td>
<td>PropertyPipe</td>
<td></td>
</tr>
<tr>
<td>Pipeline</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
public class NumCharsPipe extends AbstractPipe<String, Integer> {
    public Integer processNextStart() {
        String word = this.starts.next();
        return word.length();
    }
}

When extending the base class AbstractPipe<S,E> all that is required is an implementation of processNextStart().
Now onto Gremlin proper...
The Gremlin Architecture
The Many Ways of Using Gremlin

- Gremlin has a REPL to be run from the shell.

- Gremlin can be natively integrated into any Groovy class.

- Gremlin can be interacted with indirectly through Java, via Groovy.

- Gremlin has a JSR 223 ScriptEngine as well.
The Seamless Nature of Gremlin/Groovy/Java

Simply Gremlin.load() and add Gremlin to your Groovy class.

// a Groovy class
class GraphAlgorithms {
    static {
        Gremlin.load();
    }

    public static Map<Vertex, Integer> eigenvectorRank(Graph g) {
        Map<Vertex, Integer> m = [:]; int c = 0;
        g.V.outE.inV.groupCount(m).loop(3) {c++ < 1000} >> -1;
        return m;
    }
}

Writing software that mixes Groovy and Java is simple...dead simple.

// a Java class
public class GraphFramework {
    public static void main(String[] args) {
        System.out.println(GraphAlgorithms.eigenvectorRank(new TinkerGraph("/tmp/tinker"));
    }
}
15Load up the toy 6 vertex/6 edge graph that comes hardcoded with Blueprints.
Basic Gremlin Traversals - Part 1

Look at the neighborhood around marko.

16

---

16 Look at the neighborhood around marko.
Basic Gremlin Traversals - Part 2

gremlin> g.v(1)
==>[1]
gremlin> g.v(1).outE('created').inV
==>[3]
gremlin> g.v(1).outE('created').inV.name
==>'lop'
gremlin>

17

17What has marko created?
Basic Gremlin Traversals - Part 3

Who does marko know? Who does marko know that is under 30 years of age?

```
gremlin> g.v(1)
==>v[1]
gremlin> g.v(1).outE('knows').inV
==>v[2]
==>v[4]
gremlin> g.v(1).outE('knows').inV.name
==>vadas
==>josh
gremlin> g.v(1).outE('knows').
      inV{it.age < 30}.name
==>vadas
```

18
Who are marko’s codevelopers? That is, people who have created the same software as him, but are not him (marko can’t be a codeveloper of himself).
I don’t want to talk in terms of `outE`, `inV`, etc. Given my domain model, I want to talk in terms of higher-order, abstract adjacencies — e.g. `codeveloper`. 
• outE, inv, etc. is low-level **graph speak** (the domain is the graph).
• codeveloper is high-level **domain speak** (the domain is software development).\(^{21}\)

\(^{21}\)In this way, Gremlin can be seen as a DSL (domain-specific language) for creating DSL’s for your graph applications. Gremlin’s domain is “the graph.” Build languages for your domain on top of Gremlin (e.g. “software development domain”).
You need not make derived graphs explicit. You can, at runtime, compute them. Moreover, generate them locally, not globally (e.g. "Marko's friends from work relations").

This concept is related to automated reasoning and whether reasoned relations are inserted into the explicit graph or computed at query time.
Let's explore more complex traversals...
The Grateful Dead were an American band that was born out of the San Francisco, California psychedelic movement of the 1960s. The band played music together from 1965 to 1995 and is well known for concert performances containing extended improvisations and long and unique set lists. [http://arxiv.org/abs/0807.2466]
Grateful Dead Concert Graph Schema

- vertices
  - song vertices
    - **type**: always song for song vertices.
    - **name**: the name of the song.
    - **performances**: the number of times the song was played in concert.
    - **song_type**: whether the song is a cover song or an original.
  - artist vertices
    - **type**: always artist for artist vertices.
    - **name**: the name of the artist.

- edges
  - **followed_by** (song → song): if the tail song was followed by the head song in concert.
    - **weight**: the number of times these two songs were paired in concert.
  - **sung_by** (song → artist): if the tail song was primarily sung by the head artist.
  - **written_by** (song → artist): if the tail song was written by the head artist.
A Subset of the Grateful Dead Concert Graph
Load the GraphML (http://graphml.graphdrawing.org/) representation of the Grateful Dead graph, iterate through its vertices and get the name property of each vertex.
Becoming

gremlin> v = g.idx(T.v)[[name: ‘DARK STAR’]] >> 1
==> v[89]
gremlin> v.outE('sung_by').inV.name
==> Garcia

23

---

23 Use the default vertex index (T.v) and find all vertices index by the key/value pair name/DARK STAR. Who sung Dark Star?
Changeling

gremlin> v.outE('followed_by').inV.name
==>EYES OF THE WORLD
==>TRUCKING
==>SING ME BACK HOME
==>MORNING DEW
...
gremlin> v.outE('followed_by').transform{[it.inV.name >> 1, it.weight]}
==>EYES OF THE WORLD, 9
==>TRUCKING, 1
==>SING ME BACK HOME, 1
==>MORNING DEW, 11
...

24

\footnote{What followed Dark Star in concert?}
How many times did each song follow Dark Start in concert?
Back Stabber

```
gremlin> v.outE('followed_by').inV[[name:'BERTHA']]
==>v[4]
gremlin> v.outE('followed_by').inV[[name:'BERTHA']].name
==>BERTHA
```

```
gremlin> v.outE('followed_by').inV[[name:'BERTHA']].name.back(1)
==>v[4]
gremlin> v.outE('followed_by').inV[[name:'BERTHA']].name.back(3)
==>e[7029][89-followed_by->4]
gremlin> v.outE('followed_by').inV[[name:'BERTHA']].name.back(3).weight
==>1
```

25

---

25As you walk a path, its possible to go back to where you were n-steps ago.
gremlin> m = [:]; c = 0
==>0
gremlin> g.V.outE('followed_by').inV.groupCount(m).loop(3){c++ < 1000}
...
gremlin> m.sort{a,b -> b.value <=> a.value}
==>v[13]=762
==>v[21]=668
==>v[50]=661
==>v[153]=630
==>v[96]=609
==>v[26]=586
...

26

\footnote{Emanate from each vertex the \texttt{followed\_by} path. Index each vertex by how many times its been traversed over in the map \texttt{m}. Do this for 1000 times. The final \texttt{filter\{false\}} will ensure nothing is outputted.}
Path-o-Logical

gremlin> g.v(89).outE.inV.name
  ==>EYES OF THE WORLD
  ==>TRUCKING
  ==>SING ME BACK HOME
  ==>MORNING DEW
  ==>HES GONE
...

gremlin> g.v(89).outE.inV.name.paths
  ==>[v[89], e[7021][89-followed_by->83], v[83], EYES OF THE WORLD]
  ==>[v[89], e[7022][89-followed_by->21], v[21], TRUCKING]
  ==>[v[89], e[7023][89-followed_by->206], v[206], SING ME BACK HOME]
  ==>[v[89], e[7006][89-followed_by->127], v[127], MORNING DEW]
  ==>[v[89], e[7024][89-followed_by->49], v[49], HES GONE]
...

27

What are the paths that are out.inV.name emanating from Dark Star (v[89])?
Paths of Length < 4 Between Dark Star and Drums

gremlin> g.v(89).outE.inV.loop(2){it.loops < 4 & !(it.object.equals(g.v(96)))}[[id:'96']].paths
==>[v[89], e[7014][89-followed_by->96], v[96]]
==>[v[89], e[7021][89-followed_by->83], v[83], e[1418][83-followed_by->96], v[96]]
==>[v[89], e[7022][89-followed_by->21], v[21], e[6320][21-followed_by->96], v[96]]
==>[v[89], e[7006][89-followed_by->127], v[127], e[6599][127-followed_by->96], v[96]]
==>[v[89], e[7024][89-followed_by->49], v[49], e[6277][49-followed_by->96], v[96]]
==>[v[89], e[7025][89-followed_by->129], v[129], e[5751][129-followed_by->96], v[96]]
==>[v[89], e[7026][89-followed_by->149], v[149], e[2017][149-followed_by->96], v[96]]
==>[v[89], e[7027][89-followed_by->148], v[148], e[1937][148-followed_by->96], v[96]]
==>[v[89], e[7028][89-followed_by->130], v[130], e[1378][130-followed_by->96], v[96]]
==>[v[89], e[7019][89-followed_by->39], v[39], e[6804][39-followed_by->96], v[96]]
==>[v[89], e[7034][89-followed_by->91], v[91], e[925][91-followed_by->96], v[96]]
==>[v[89], e[7035][89-followed_by->70], v[70], e[181][70-followed_by->96], v[96]]
==>[v[89], e[7017][89-followed_by->57], v[57], e[2551][57-followed_by->96], v[96]]
...

Get the adjacent vertices to Dark Star (v[89]). Loop on outE.inV while the amount of loops is less than 4 and the current path hasn't reached Drums (v[96]). Return the paths traversed.
Generic Steps

Gremlin is founded on a collection of atomic “steps.” The syntax is generally seen as `step.step.step`. There are 3 categories of steps.

- **transform**: map the input to some output. \( S \rightarrow T \)
  - `outE`, `inV`, `paths`, `copySplit`, `fairMerge`, ...

- **filter**: either output the input or not. \( S \rightarrow (S \cup \emptyset) \)
  - `back`, `except`, `uniqueObject`, `andf`, `orf`, ...

- **sideEffect**: output the input and yield a side-effect. \( S \rightarrow S \)
  - `aggregate`, `groupCount`, ...
Integrating the JDK (Java API)

- Groovy is the host language for Gremlin. Thus, the JDK is natively available in a path expression.

```
gremlin> v.outE('followed_by').inV{it.name.matches('D.*')}.name
==>
DEAL
==>
DRUMS
```

- Examples below are not with respect to the Grateful Dead schema presented previously. They help to further articulate the point at hand.

```
gremlin> x.outE.inV.transform{JSONParser.get(it.uri).reviews}.mean()
...
```

```
gremlin> y.outE('rated').filter{TextAnalysis.isElated(it.review)}.inV
...
```
In the End, Gremlin is Just Pipes

gremlin> (g.v(89).outE('followed_by').inV.name).toString()
==> [OutEdgesPipe<followed_by>, InVertexPipe, PropertyPipe<name>]

29

---

29Gremlin is a domain specific language for constructing lazy evaluation, data flow Pipes (http://pipes.tinkerpop.com). In the end, what is evaluated is a pipeline process in pure Java over a Blueprints-enabled property graph (http://blueprints.tinkerpop.com).
Credits

- Marko A. Rodriguez: designed, developed, and documented Gremlin.
- Pavel Yaskevich: heavy development on earlier versions of Gremlin.
- Darrick Wiebe: inspired Groovy Gremlin and develops on Pipes and Blueprints.
- Peter Neubauer: promoter and evangelist of Gremlin.
- Ketrina Yim: designed the Gremlin logo.

http://gremlin.tinkerpop.com
http://groups.google.com/group/gremlin-users
http://tinkerpop.com