Functional Programs as Linked Data

Joshua Shinavier
josh@fortytwo.net

Soph-Ware Associates, Inc.,
624 W. Hastings Rd, Spokane, WA 99218 USA
http://www.soph-ware.com

Abstract. The idea of linked programs, or *procedural* RDF metadata, has not been deeply explored. This paper introduces a dedicated scripting language for linked data, called Ripple, whose programs both operate upon and reside in RDF graphs. Ripple is a variation on the *concatenative* theme of functional, stack-oriented languages such as Joy and Factor, and takes a multivalued, pipeline approach to query composition. The Java implementation includes a query API, an extensible library of primitive functions, and an interactive command-line interpreter.

1 Introduction

Most of the data which populates today's Semantic Web is purely *descriptive* in nature, while the complex procedural machinery for querying, crawling, transforming and reasoning about that data is buried within applications written in high-level languages such as Java or Python, and is neither machine-accessible nor reusable in the Semantic Web sense. This paper explores the notion of *linked* or distributed programs as RDF graphs, and presents a functional, *concatenative* interpreted language, closely related to Manfred von Thun's Joy[1], as a proof of concept.

For a Turing-complete RDF query language, Ripple is an exercise in minimalism, both in terms of syntax and semantics. A Ripple *program* is a nested list structure described with the RDF collections vocabulary, and is thereby a first-class citizen of the Semantic Web. Significantly, the language is restricted to a single RDF query operation: the forward traversal of links. Broadly speaking, the goal of this project is to demonstrate that:

1. linked programs have all of the advantages of generic linked data[2]
2. a stack language is like a path language, only better
3. for most linked data purposes, forward traversal is all you need

[1] http://www.latrobe.edu.au/philosophy/phimvt/joy/j01tut.html
[2] http://www.w3.org/DesignIssues/LinkedData.html

The Java implementation resolves HTTP URIs, dynamically, in response to traversal operations, similarly to the Tabulator [1], the Semantic Web Client Library[3], and related tools[4]. Linked programs are drawn into the query environment in exactly the same manner, extending the the evaluation of queries to a distributed code base.

2 Syntax

Ripple's query model combines a computational scheme based on stack manipulation with a functional "pipes and filters" pattern. The stack paradigm makes for minimal, *point-free* syntax at the RDF level, while the *pipeline* mechanism accommodates RDF's multivalued properties by distributing operations over arbitrary numbers of intermediate results.

2.1 Textual Representation

The code samples in this paper are written in Ripple's own RDF syntax, which is very close to Turtle[5].
Note: throughout this paper, the namespace prefixes `rdf`, `rdfs`, `xsd`, `rpl`, `stack`, `stream`, `math`, `graph`, *and* `etc` *are assumed to be predefined. Typing any of these prefixes, followed by a tab character, at the command line will reveal a number of terms in the corresponding namespace.*

URIs may be written out in full or abbreviated using namespace prefixes.

```
<http://www.w3.org/2004/09/fresnel>.  # a URI reference
rdfs:Class.               # a qualified name
:marvin.                  # using the working (default) namespace
rdfs:.                    # this URI has an empty local name
:.                        # legal, and occasionally meaningful
```

Keywords are the local names of a fixed set of special URIs. Currently, the local name of every primitive function is a keyword.

```
swap.                     # same as stack:swap
```

RDF Literals are represented as numbers and strings.

```
42.                       # an integer (xsd:integer for now)
3.1415926535.             # a floating point value (xsd:double)
"the Universe".           # a string Literal (xsd:string)
"English"@en.             # a plain Literal with a language tag
"2007-05-07"^^xsd:date.   # a generic typed Literal
```

[3] http://sites.wiwiss.fu-berlin.de/suhl/bizer/ng4j/semwebclient/
[4] http://moustaki.org/swic/
[5] http://www.dajobe.org/2004/01/turtle/

Blank nodes lose their identity between sessions, but it's often useful to refer to them within a session.

```
_:node129n4ttifx2.       # a bnode with a generated id
_:tmp1.                  # a bnode with a user-defined id
```

Lists are indicated by parentheses.

```
("apple" "banana").      # a rdf:List
```

One symbol not found in Turtle is the *slash operator* (see Query Evaluation). Where it is affixed to an RDF property, it is equivalent to the forward traversal operator[6] of Notation3. A likely extension to the language will add quantifiers for *regular path expressions*, including `/?`, `/*`, and `/+`. E.g.

```
# @define smush: /*owl:sameAs.
```

2.2 Commands and Queries

The interface distinguishes between two kinds of statements: *queries*, which are expressions to be evaluated by the query processor, and commands or *directives*, which perform specific tasks. Currently supported directives include:

```
# Define a namespace prefix foo.
@prefix foo: <http://example.org/foo#>.

# Define :bar as the list (1 2 3).
@define bar: 1 2 3.

# Remove all statements about :bar
@undefine bar.

# Write (a bnode closure of) the terms in namespace foo: to a file.
@export foo: "file.rdf".

# Save the entire graph to a file.
@saveas "file.rdf".

# Quit the application.
@quit.
```

2.3 Compositional Syntax

At the level of lists and nodes, Ripple queries are expressed in postfix notation, or in *diagrammatic order*. Each query is a list in which the concatenation of symbols represents the composition of functions. When the functions are RDF properties, a query is equivalent to a path expression:

[6] http://www.w3.org/DesignIssues/N3Alternatives

```
# => "The RDF Vocabulary (RDF)"
("apple" "banana")/rdf:type/rdfs:isDefinedBy/dc:title.
```

This particular expression takes us from the list `("apple" "banana")` to its type, `rdf:List`, from the type to the ontology, `rdf:`, which defines it, and from the ontology to its title, `"The RDF Vocabulary (RDF)"`. The same idea applies to primitive functions:

```
# => recently pinged DOAP documents
10 "doap" /pingTheSemanticWeb/toString.
```

Here, too, we can imagine data (a *stream* of lists or *stacks*) flowing from the left hand side of the expression and emerging from the right hand side of the expression (the *head* of the stack) after undergoing a transformation of some kind. In this case, a stream of one stack containing the two arguments to the built-in "Ping the Semantic Web" function becomes a stream of ten stacks containing URIs which `toString` then converts to string literals.

2.4 RDF Representation

Every Ripple program is expressible in RDF, where it takes the form of a simple RDF list. In the Java implementation, conversion between the RDF graph representation of a list and its more efficient linked list counterpart is implicit. For example, you may navigate a list using either list primitives or the RDF collections vocabulary, interchangeably.

```
("apple" "banana")/rdf:first.  # => "apple"
("apple" "banana")/uncons/pop. # => "apple"
```

When we assign a program a URI, we're pushing its definition to the same RDF model from which our query results are drawn:

```
@define hello:
    "Hello world!".
```

Given an appropriate base URI and web-visible triple store, the program itself becomes a part of the global graph of linked data, enabling remote users and applications to read it in, execute it, and build upon it without restriction.

3 Query Evaluation

Ripple's evaluation strategy hinges on a dichotomy between between `active` and `passive` stack items. Active items exhibit type-specific behavior, whereas passive items simply push a "copy of themselves" to the stack. Every item has a well-defined *arity* which, roughly speaking, is the number of arguments it consumes. To be precise, the arity is the depth to which the stack must be reduced before the item can be applied. For instance, the `swap` function requires two arguments, so the evaluator must make sure that the stack is normalized

to two levels before `swap` receives it. Evaluation proceeds, *lazily*, until all active items have been eliminated from the head of the stack, at which point the stack is said to be in *normal form* to one level. The only symbol active by default is the `rpl:op` operator, which has the effect of making the preceding item active as follows:

1. *RDF properties* consume a subject and map it to a stream of zero or more objects
2. *primitive functions* exhibit "black box", custom behavior
3. *lists* become an extension of the stack (execution "removes the parentheses")
4. *all other items* consume nothing and produce nothing (they cause the stack to disappear)

Note: in the text notation, `rpl:op` is abbreviated as the slash prefix attached to the item it follows. For example, `(2 /dup)` is just a more compact way of writing `(2 dup rpl:op)`.

Now, consider the following definition and query:

```
# x => x*x
@define sq: /dup/mul.

# => 16
4/:sq.
```

The major steps in the evaluation of the query are as follows:

1. `(4 :sq rpl:op!)` – `rpl:op` is active by default
2. `(4 dup rpl:op! mul rpl:op!)` – dereference and dequote list `:sq`
3. `(4 dup rpl:op! mul!)` – `mul` primitive becomes active
4. `(4 dup! mul!)` – `mul` needs two arguments. Recurse
5. `(4 4 mul!)` – `dup` consumes its one argument and applies its rewrite rule
6. `(16)` – `mul` now has two reduced arguments, and applies its rule, yielding `16`

3.1 The Compositional Pipeline

Ripple differs from typical stack languages in that, rather than consuming a single stack as input and producing a single stack as output, Ripple's functions operate on *streams* containing any number of stacks. For instance, the following query yields not one, but several values:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
<http://www.w3.org/People/Berners-Lee/card#i>/foaf:knows.
```

If we compose it with another query, the second query needs to be capable of consuming not just a single stack, but many stacks, and of distributing its operation over all of them:

```
<http://www.w3.org/People/Berners-Lee/card#i>/foaf:knows/foaf:name.
```

To this end, Ripple conceives of functions as "filters", which behave like the elements of a pipeline: receiving input, transforming it, and passing it on. Filters may have state; for example, stream:limit filters (which count their input stacks and stop transmitting them after a certain point) or stream:unique filters (which remember their input stacks, and will not transmit a duplicate).

4 Examples and Use Cases

4.1 Arithmetic

In Ripple, as in Forth, stack shuffling operations take the place of bound variables in complex expressions.

```
# n => fibonacci(n)
@define fib:
    0 1 /rolldown     # push initial value pair and put n on top
    (/swap/dupd/add)  # push the step function
    /swap/times       # execute the step function n times
    /pop.             # select the low value

# => 13
7/:fib.
```

4.2 Recursion

Owing to the global nature of URIs, recursive definition is uncomplicated in Ripple. Functions may reference each other, and themselves, arbitrarily. Evaluation of @defined programs is delayed until forced by the evaluation of a query.

```
# n => factorial(n)
@define fact:
    /dup 0 /equal                  # if n is 0...
        (1 /popd)                  # yield 1
        (/dup 1 /sub /:fact /mul)  # otherwise, yield n*fact(n-1)
    /branch.

# => 120
5/:fact.
```

4.3 Exploring a FOAF Neighborhood

The ability to request and aggregate RDF metadata on the fly gives Ripple's query engine the properties of a web crawler. "Intelligent" applications aside, there are many conceivable use cases for simply discovering and aggregating a large chunk of data in a configurable fashion. The following program targets the decentralized, linked data of the FOAF network, beginning with Tim Berners-Lee's profile.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.

# foaf1 => foaf1, foaf2, foaf3, ...
@define foafStep:   # iterator for a FOAF crawler
    (   id          # include foaf1 itself
        owl:sameAs  # include nodes identified with foaf1
        foaf:knows  # include those foaf:known by foaf1
    )/each/i        # apply all three patterns at once
    /unique.        # eliminate duplicate results

# => names of TBL and friends, and of friends of friends
<http://www.w3.org/People/Berners-Lee/card#i>
    :foafStep 2/times /foaf:name.
```

4.4 Searching and Filtering in Revyu.com

Revyu.com is one of many[7] innovative web services which offer linked RDF views of their data. The web site links reviewers to reviews, reviews to things, and some things to book metadata in the RDF Book Mashup[8]. Here, we're more interested in narrowing the search space to a handful of "hits" than we are in the aggregated data as a whole.

```
# a f => a, if a/f is true, otherwise nothing
@define restrict:
    /dupd/i    # apply the filter criterion, f
        id     # keep the stack if a/f is true
        scrap  # throw the stack away if it isn't
    /branch.

@prefix scom: <http://sites.wiwiss.fu-berlin.de[no break]
/suhl/bizer/bookmashup/simpleCommerceVocab01.rdf#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix owl:  <http://www.w3.org/2002/07/owl#>.

# r => r, if r is a book review, otherwise nothing
@define bookReviewsOnly:
    (   /foaf:primaryTopic          # from review to topic
        /owl:sameAs                 # from topic to possible book
        /rdf:type scom:Book /equal  # is it really a book?
    )/:restrict.

@prefix rev:  <http://purl.org/stuff/rev#>.
```

[7] http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/DataSets
[8] http://sites.wiwiss.fu-berlin.de/suhl/bizer/bookmashup/

```
# => labels of all of Tom's more favorable book reviews
<http://revyu.com/people/tom> /foaf:made
    /:bookReviewsOnly              # books only
    (/rev:rating 3 /gt)/:restrict  # 4 stars or better
    /rdfs:label.                   # from review to label
```

The query yields two results:

```
rdf:_1  ("Review of The Unwritten Rules of Phd Research, [...]")
rdf:_2  ("Review of Designing with Web Standards, by Jeff[...]")
```

Unlike the FOAF example, this program places a heavy burden on a single network server. If possible, it would be in the best interest of both server and client to offload query evaluation to the web service. Knowing when to use an API for federated queries, as opposed aggregating data, probably has more to do with the economics of distributed computing than with the rest of this paper. However, the API itself is certain to be a simple one. It could even be as simple as the passing of the dereferenceable URI of a single expression, to be resolved and evaluated remotely. Perhaps a REST service and an OWL-S service description could be generated, based on a given query pattern.

4.5 Graph Transformations and Other Side Effects

The Ripple implementation includes a number of "experimental" primitives which may be useful for graph transformations and in common metadata "rewiring scenarios"[9]. As these primitives affect the state of their environment (for instance, by adding or removing statements), they are to be used with caution. The following is an example of a "safe" application of the `new` and `assert` primitives. It transforms a resource description by creating a new blank node which retains and renames a few of the original node's edges.

Note: to see a description of these or any other primitive functions, type in the name of the function at the command line, followed by a period.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

# node map => a new node with "mapped" edges
@define mapped:
    /each/i               # distribute over a list of pairs
    /new                  # create a new node
    i/dipd/rotate/assert  # assert a mapped statement
    1/limit.              # produce the node just once

# Maps to a minimal FOAF-like vocabulary.
@prefix ex: <http://example.org/minimalFoaf#>.
```

[9] http://simile.mit.edu/wiki/Rewiring_Scenarios

```
@define myMap:
    (rdf:type rdf:type)     # copy any rdf:type edges
    (foaf:name ex:name)     # map foaf:name to ex:name
    (foaf:knows ex:knows).  # map foaf:knows to ex:knows

# => minimal FOAF for TBL
<http://www.w3.org/People/Berners-Lee/card#i> :myMap/:mapped.
```

Other side effects may be more subtle. For instance, the following program affects the "Ping the Semantic Web" service which we queried earlier, possibly influencing subsequent query results.

```
# uri => PTSW's ping response
@define ping:
    /toString/urlEncoding
    "http://pingthesemanticweb.com/rest/?url=" /swap/strCat
    /get.

# => PTSW's response to a ping of Ripple's DOAP URI/document
<http://fortytwo.net/2007/03/ripple/doap#>/:ping.
```

Other potentially useful side-effects include the sending of an e-mail or the firing of a system-specific event.

5 Implementation

Ripple is implemented in Java and uses the Sesame 2 (beta) RDF framework. The command-line interface relies on ANTLR for lexer/parser generation and on JLine for command history and tab completion. The project is built with Maven and is distributed under an open source license. Software releases are available at `http://fortytwo.net/ripple`.

6 Related Work

Ripple is strictly a resource-centric language and is not intended as an alternative to SPARQL. The fact that Ripple is rather like a path language makes it much more effective for some tasks than the "relational" SPARQL, and vice versa. The same can be said of any of the other RDF Path[10] languages, such as Versa[11] or the path portion of PSPARQL[12]. Possibly the closest thing to Ripple is Ora Lassila's Wilbur[13] toolkit and path language, which integrates RDF with Common Lisp (coming soon: Python!). Deep integration [2] of languages such as Ruby [3] and Python [4] with RDF are related efforts, as well.

[10] http://esw.w3.org/topic/RdfPath
[11] http://copia.ogbuji.net/files/Versa.html
[12] http://psparql.inrialpes.fr/
[13] http://www.lassila.org/publications/2001/swws-01-abstract.shtml

7 Conclusion and Future Work

Ripple is an exploratory project, which is to say that further development will be driven by discoveries made along the way. If the hypertext web is any indication of the future of the web of data, it will be vast, complex, and overall, loosely structured. As it grows, we will need a sophisticated *web of programs* to keep pace with it.

In addition to the command-line interpreter, the distribution contains a query API, and the embedding of the Ripple query processor in further Semantic Web applications is a very likely use case. In the short term, I would like to integrate Ripple with a graphical RDF browser such as Longwell[14], and investigate the federated query scenario mentioned above.

Christopher Diggins has done some work on static typing for stack languages [5], and it would be interesting to see whether and how such a type system could be expressed with an ontology. Currently, Ripple's type system is just a form of API documentation, so a more rigorous one would definitely be a step forward.

Compilation is another interesting possibility. While Ripple will always have an interpreted component, a modular Ripple program could be compiled to optimized Java bytecode and then reinserted into the environment as a primitive function.

References

1. Berners-Lee, T., et al., Tabulator: Exploring and Analyzing linked data on the Semantic Web. In Proceedings of the 3rd International Semantic Web User Interaction Workshop, 2006.
2. Vrandecic, D., Deep Integration of Scripting Languages and Semantic Web Technologies. In 1st International Workshop on Scripting for the Semantic Web, volume 135 of CEUR Workshop Proceedings. Herakleion, Greece, May 2005. ISSN: 1613-0073
3. Fernandez, O., Deep Integration of Ruby with Semantic Web Ontologies. See http://obiefernandez.com/DeepIntegration.pdf
4. Babik, M., Hluchy, L., Deep Integration of Python with Web Ontology Language. In 2nd International Workshop on Scripting for the Semantic Web, volume 181 of CEUR Workshop Proceedings. Budva, Montenegro, June 2006. ISSN: 1613-0073.
5. Diggins, C., Typing Functional Stack-Based Languages. See http://www.cat-language.com/paper.html

[14] http://simile.mit.edu/wiki/Longwell