`call`: A Nucleus for a Web of Open Functions

Maurizio Atzori

Math/CS Department
University of Cagliari
Via Ospedale 72
09124 Cagliari (CA), Italy
`atzori@unica.it`

Abstract. In our recent work we envisioned a Web where functions, like Linked Data, can be openly published and available to all the users of any remote SPARQL endpoint. The resulting *Web of Functions* can be realized by introducing a `call` SPARQL extension that can invoke any remote function (custom third-party extensions) by only knowing its corresponding URI, while the implementation and the computational resources are made available by the function publisher. In this paper we demo our framework with a set of functions showing *(1)* advanced use of its higher-order expressivity power featuring, e.g., function composition of third-party functions, and *(2)* a possible bridge between hundreds of standard Web APIs and the Web of Functions. In our view these functions found an initial nucleus to which anyone can contribute within the decentralized Web of Functions, made available through `call`.

1 Introduction

While extending the language with user-defined custom functions (sometimes called extension functions) represented by URIs is a native feature of the SPARQL language, the mechanism only works on the single endpoint featuring that specific function. In our recent work [1], we investigated interoperability, computational power and expressivity of functions that can be used within a SPARQL query, envisioning a Web where also functions can be openly published, making them available to all the users of any other endpoint. In [1] we define a `wfn:call` function with three possible architectures to deploy it in a backward compatible manner. It is the basis needed in order to realize a *Web of Open Functions*, meaning that users can call a function by only knowing its corresponding URI, as it is the case for entities and properties in the Web of Open Data[1], while the implementation and the computational resources are made available by the function publisher, as it happens with usual Web APIs.

Practically, supposing that Alice wants to use a Bob's SPARQL extension (only defined in Bob's endpoint) from her own endpoint, she will write the following:

```
PREFIX wfn: <http://webofcode.org/wfn/>
```

[1] We titled this paper after DBpedia milestone work in [2].

```
PREFIX bob: <http://bob-server.org/fn/>
SELECT *
WHERE {
    # within Alice data, find useful values for ?arg1, ?arg2
    ...
    # now use Bob's function
    FILTER(wfn:call(bob:complexFunction, ?arg1, ?arg2) )
}
```

Therefore, the function `wfn:call` takes care of finding the right endpoint (see [1, 3, 4]), i.e., Bob's, and then remotely call Bob's `complexFunction`. We believe this may be the first step toward a novel view of the Web as a place holding code and functions, not only data as the Linked Data is greatly doing. The Semantic Web already shifted URIs from pages to conceptual entities, primarily structured data. We believe that among these concepts there should be computable functions.

In this paper we demo our open source implementation for the `wfn:call` function, realized as an Apache Jena's custom function extension, and available with other resources (including a link to our endpoint that publishes it) at `http://atzori.webofcode.org/projects/wfn/`. In particular, we devise an initial nucleus of practical functions that may empower SPARQL queries with computations that exploit higher-order expressivity and hundreds of existing Web APIs.

2 Fully Higher-Order Functions in SPARQL

Higher-order functions (HOF) are functions that take functions as either input and/or output. Languages that allow functions to be used as any other kind of data, are said to feature first-class functions. Here we show that these advanced expressivity, typical of functional languages, can be used within SPARQL by means of `wfn:call`. In the following we exemplify it by describing the use of three important HOF: reduce, compose and memoize.

Reduce. In functional languages "reduce" (also called "fold", "inject" or "aggregate") is a function that takes a binary function (e.g., the + operator) and a list (e.g., a list of integers), producing a result by recursively applying the function to the remaining list (providing, e.g., the sum of all the elements). Thus, it represents a general-purpose aggregation mechanism potentially useful in SPARQL queries. In the following we show how it can be used to apply the binary max function provided by Jena to a list of 4 numbers:

```
PREFIX call: <http://webofcode.org/wfn/call>
PREFIX afn: <http://jena.hpl.hp.com/ARQ/function#>.

SELECT ?max {
    BIND( call:(wfn:reduce, afn:max, 5, 7, -1, 3) AS ?max)
}
```

resulting in `?max = 7`.

Compose. Another important HOF is the composition function. Given two functions g and f, it returns a third function that behaves as the application of f followed by the application of g, i.e., $g(f(.))$. The following query excerpt:

```
BIND(call:(wfn:compose, fn:upper-case, afn:namespace)
AS ?uppercase_ns).
BIND(call:(?uppercase_ns, <http://something.org/myentity>) AS ?result)
```

returns the uppercased namespace, that is, `HTTP://SOMETHING.ORG/`. In particular, variable `?uppercase_ns` is binded to a dynamically generated SPARQL function that, whenever invoked, applies `afn:namespace` followed by `fn:upper-case`.

Memoize. Many SPARQL queries may iterate over intermediate results, requiring the execution of the same function multiple times, possibly with the same paramenters. In order to speed up the execution of potentially time-consuming functions, we implemented a memoization function that keeps the results of function calls and then returns the cached result when the same inputs occur again. This part of the query:

```
BIND(call:(wfn:memoize, ?slow_function) AS ?fast_function).
BIND(call:(?fast_function, 1) AS ?result1). #1st time at normal speed
BIND(call:(?fast_function, 1) AS ?result2). #2nd time is faster
```

dynamically generates a `?fast_function` that is the memoization of function in `?slow_function`. Please notice that this kind of useful features are possible only in higher-order environments, such as the one resulting by the use of our `call` function [1].

3 Bridging Web APIs and the Web of Functions

In order to develop a useful Web of Functions, the small set of auxiliary functions presented in the previous section are clearly not enough. While some other powerful user-defined functions are already online (e.g., `runSPARQL` [5] computes recursive SPARQL functions), we need a larger nucleus of functions allowing any sort of computation from within SPARQL queries. In this section we propose the exploitation of a well-known Web API hub, namely *Mashape*[2], by using a simple bridge function that allows to call any mashape-featured API. This function, that we called `wfn:api-bridge`, pushes hundreds of Web APIs within the Web of Functions, ranging from weather forecast to face detection, from language translation to flight information lookup. For instance, we can iterate over DBpedia cities in Tuscany finding those with a close airport, cheap flight and good weather during the week after a given arrival day. In the following we find large Tuscany cities sorted by current weather temperature:

```
SELECT *  {
    ?city dbpedia-owl:region dbpedia:Tuscany;
          dbpedia-owl:populationTotal ?population;
```

[2] Freely available at `http://www.mashape.com/`

```
         geo:lat ?lat;  geo:long ?long.

    FILTER(?population > 80000).
    BIND(CONCAT("lat=",?lat,"&lon=",?long) AS ?parameters)

    BIND( call:(wfn:api-bridge, "community-open-weather-map", ?parameters,
      "main.temp") as ?temperature).
} ORDER BY ?temperature
```

The `wfn:api-bridge` function calls the Mashape Web API corresponding to the first argument, with parameters specified in the second argument, then returning the JSON field selected in the third argument. Different APIs necessary to answer the query can be combined together with `compose`, and the resulting function may be memoized for better performance if needed. Advanced uses may exploit Linked Data information to search through existing Web API repositories [6].

4 Conclusions and Demo Showcase

We presented a set of SPARQL extensions containing higher-order manipulation functions, allowing for instance function composition, together with a bridge function that allows the use of hundreds of existing Web APIs from any SPARQL endpoint featuring the `wfn:call` function, that we developed and opensourced for Apache Jena. This set, forming an initial nucleus for the *Web of Functions*, enables a wide spectrum of much powerful SPARQL queries w.r.t. the ones we are currently used to, with a number of practical examples that will be showcased during the demo and made available at our website.

Acknowledgments. This work was supported in part by the RAS Project CRP-17615 *DENIS: Dataspaces Enhancing Next Internet in Sardinia* and by MIUR PRIN 2010-11 project *Security Horizons*.

References

1. Atzori, M.: Toward the Web of Functions: Interoperable Higher-Order Functions in SPARQL. In: 13th International Semantic Web Conference (Research Track). (2014)
2. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: DBpedia: A Nucleus for a Web of Open Data. In: The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference (ISWC/ASWC). (2007) 722–735
3. Paulheim, H., Hertling, S.: Discoverability of SPARQL Endpoints in Linked Open Data. In: International Semantic Web Conference (Posters & Demos). (2013)
4. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing Linked Datasets with the VoID Vocabulary (December 2010)
5. Atzori, M.: Computing Recursive SPARQL Queries. In: 8th IEEE International Conference on Semantic Computing. (2014)
6. Bianchini, D., Antonellis, V.D., Melchiori, M.: A Linked Data Perspective for Effective Exploration of Web APIs Repositories. In: ICWE 2013. (2013) 506–509