Discovering and Using Functions via Content Negotiation

Ben De Meester, Anastasia Dimou, Ruben Verborgh, and Erik Mannens

Ghent University – iMinds – Data Science Lab, Belgium
{firstname.lastname}@ugent.be

Abstract. Data has been made reusable and machine-interpretable by publishing it as Linked Data. However, automatically processing Linked Data is not fully achieved yet, as manual effort is still needed to integrate existing tools and libraries within a certain technology stack. To enable automatic processing, we propose exposing functions and methods as Linked Data, and publishing it in different programming languages. Content negotiation can be used to cater to different technology stacks, and common, technology-independent identifiers make them discoverable. As such, we can enable automatic processing of Linked Data across formats and technology stacks. By using discovery endpoints, similar to those used to discover vocabularies and ontologies, the publication of these functions can remain decentralized.

Keywords: Content Negotiation, Function, Linked Data

1 Introduction

By publishing data as Linked Data, we are moving to a Web of integrated, reusable, and machine-interpretable data. However, to process that data, we need processing instructions. E.g., calculating a distance between two points `[x1, y1]` and `[x2, y2]` can be done using the Euclidean distance. This, in JavaScript, could be calculated as `Math.sqrt( (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) )`.

Libraries and repositories of common functions and methods exist[1], but integrating them still requires manual effort to 'glue' the different libraries together. Human intervention is needed, as functions are implemented in different technologies, and the way of executing these functions is not declared semantically, thus ruling out machine-interpretability.

At the same time, there are many ongoing efforts to integrate processing instructions and (Linked Data) applications. For example, integrating processing functions when mapping from non-RDF data to RDF data [2], adding custom functions in SPARQL queries [5], or creating compositions of hypermedia APIs [7]. Also, a lot of ongoing work is focussed on specifying implementation-independent processing instructions using Web services (such as Hydra [4]). However, relying on Web services means that (1) the applications using them and the

[1] See, e.g., https://www.npmjs.com/package/euclidean-distance

services themselves must always be online, which is not feasible in every context, and (2) all input and output needs to be transferred over HTTP, which hurts performance. This to be transferred data might be too trivial (e.g., calculating a geographic distance between two data points), or too large to be easily handled in practice (e.g., calculating aggregates over billion-triple local data).

In this paper, we propose publishing technology-independent function descriptions as Linked Data. These descriptions provide a common reference for processing instructions across technology stacks (e.g., JavaScript snippets vs. Web services). To integrate the processing instructions in different technology stacks, content negotiation can be used to publish the same functions in different implementations [3], either accessed remotely as a Web service or downloaded locally to be automatically integrated in the local technology stack. This way, we make similar methods in different programming languages available, just as websites are made available in different human (natural) languages, or as content is made available both for humans as for machines. This allows for more automatic composition of processing instructions, thus building towards the intelligent agents as initially envisioned when first proposing the Semantic Web.

2 Methodology

The proposed methodology consists of the following parts (Figure 1):

1. provide technology-independent semantic descriptions of functions,
2. publish these functions, both their semantic description and the specific implementations (not necessarily all at the same place),
3. make the semantic descriptions discoverable and queryable, and
4. provide content negotiation to allow uniform access to different implementations.

We achieve the first part of this methodology using for instance the Function Ontology [1]. The Function Ontology is a technology-independent way of describing functions of various complexity, without any assumptions on programming paradigms. It consists of only six base classes and five relations. It is used to describe a `Function` that possibly `solves` a certain `Problem`, and possibly `implements` some `Algorithms`. The `Function` `expects` zero or more `Parameters` and `returns` zero or more `Outputs`. An `Execution` `executes` a certain `Function` by binding values to the `Parameters`. The Euclidean distance function could thus be described as follows:

```
ex:euclidDistanceFn a fno:Function ;
  fno:solves ex:EuclideanDistanceProblem ;
  fno:expects ( ex:dataPoint1 ex:dataPoint2 ) ;
  fno:returns ( xsd:double ) .
```

We consider the Function Ontology as it is small and technology/problem-domain independent, and thus allows for easier reuse[2]. Such a function description serves as a common reference across technology stacks.

[2] `http://w3id.org/function/spec`

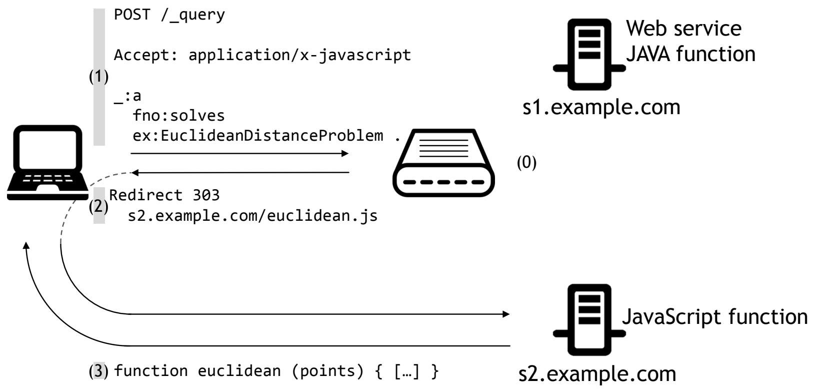
POST /_query
Accept: application/x-javascript
(1)
_:a
fno:solves
ex:EuclideanDistanceProblem .
(0)
Redirect 303
(2)
s2.example.com/euclidean.js
Web service
JAVA function
s1.example.com
JavaScript function
s2.example.com
(3) function euclidean (points) { […] }

Fig. 1: General overview of content negotiation over functions. Function descriptions can be published to discovery endpoints (0). When a user queries such an endpoint (1), the response can redirect (2) to the actual implementation (3).

Just as Linked Open Vocabularies [6] provides for a discovery endpoint to vocabularies and ontologies, functions can be submitted to similar discovery endpoints (Figure 1 (0))[3]. No actual implementations are hosted on these endpoints, but the semantic descriptions can be aggregated, and content negotiation can be set up to direct the user to the different implementations across servers. For example, when a user needs a certain functionality in a certain technology, she can query discovery endpoints for functions that solve a given problem, and are available in a specific format (in the case of Figure 1 (1), a JavaScript snippet). The discovery endpoint can then redirect (2) the user to a server hosting the actual code (3).

3 Discussion

The proposed methodology uses widespread methods to publish and discover functions, catered to different needs. However, they also inherit the same risks as there are in the current Web: just as websites are not always accessible in the language you prefer, functions might not be implemented in the technology needed. Fortunately, similar workarounds can be used: as best-effort automatic translation systems can provide for translations of websites to your preferred language, there exist engines that use emulation or code translation to allow incorporating code snippets from a different programming language[4].

[3] This compares to a technology-independent `https://www.haskell.org/hoogle/` (that helps users discover Haskell functions by type signature), but where the semantics of a function is also taken into account.

[4] See, e.g. the Nashorn engine to use JavaScript procedures within the Java Virtual Machine (`https://blogs.oracle.com/nashorn/`).

Furthermore, the proposed methodology allows for choosing between remote web services and locally downloaded methods. Depending on the context (online/offline, execution rate and/or data throughput) a user can decide whether to use the online service, or download the method locally (given that both exist). The latter would involve (automatically) integrating the downloaded code in the parent application. Depending on the technology used, the client will need to interpret the semantic description of a function, to know how to actually execute a function. For Web services, many declarative description formats exist to automatically derive this (e.g., Hydra [4]). For other technologies, additional descriptions might be necessary.

4 Conclusion

Functions can be described semantically and technology-independent using the Function Ontology. By publishing these descriptions alongside their implementations, we provide a uniform access to similar functions for different technologies, both accessible remotely and downloadable locally. These implementations can exist distributed, but are made discoverable using centralized endpoints, similar to widely adopted portals such as Linked Open Vocabularies. This in turn allows for machine-interpretable and discoverable libraries of functions.

References

1. De Meester, B., Dimou, A., Verborgh, R., Mannens, E., Van de Walle, R.: An ontology to semantically declare and describe functions. In: Proceedings of the 13th ESWC: Satellite Events (2016), accepted for publication
2. Debruyne, C., O'Sullivan, D.: R2RML-F: Towards sharing and executing domain logic in R2RML mappings. In: Workshop on Linked Data on the Web (2016), `http://events.linkeddata.org/ldow2016/papers/LDOW2016_paper_14.pdf`
3. Fielding, R.T., Reschke, J.: Hypertext Transfer Protocol (HTTP/1.1): Semantics and content – content negotiation. Tech. rep., IETF (June 2014), `http://tools.ietf.org/html/rfc7231#section-3.4`, accessed January 26th, 2015
4. Lanthaler, M., Gütl, C.: Hydra: A Vocabulary for Hypermedia-Driven Web APIs. In: LDOW, WWW (2013)
5. Regalia, B., Janowicz, K., Gao, S.: VOLT: A provenance-producing, transparent SPARQL proxy for the on-demand computation of linked data and its application to spatiotemporally dependent data. In: The Semantic Web. Latest Advances and New Domains (2016), `http://geog.ucsb.edu/~jano/eswc2016.pdf`
6. Vandenbussche, P.Y., Atemezing, G.A., Poveda-Villalón, M., Vatant, B.: Linked Open Vocabularies (LOV): a gateway to reusable semantic vocabularies on the web. Semantic Web (2015), `http://www.semantic-web-journal.net/system/files/swj1178.pdf`, accepted for publication
7. Verborgh, R., Arndt, D., Van Hoecke, S., De Roo, J., Mels, G., Steiner, T., Gabarró Vallés, J.: The pragmatic proof: Hypermedia API composition and execution. Theory and Practice of Logic Programming (2016), `http://arxiv.org/pdf/1512.07780v1.pdf`, accepted for publication