A Web That Does

Denny Vrandečić

KIT, Germany
denny.vrandecic@kit.edu

Abstract. The Semantic Web has introduced a number of languages for exchanging data, and for providing semantics for the data that allows to infer further knowledge. But these languages remain limited in their scope. In this paper we propose a wiki that enables its users to write and edit functions that can use Web data to enable completely novel functionalities. Whereas Wikipedia provided a human-editable and -readable encyclopedia of all knowledge, Freebase a human-editable database, this new system will provide a space for the easy creation of functionalities that can be executed anywhere in the cloud.

1 Introduction

When wikis were introduced in 1995 they enabled anyone who could access these first wiki pages to easily write content and publish it on the Web. Publishers and authors of content did no longer have to deal with web servers, protocols, not even markup languages such as HTML, but only with the new simple wiki syntax. Today, many wiki-inspired systems exist that work with rich WYSIWYG editors and further reduce the complexity of publishing content on the Web. Many people use YouTube for publishing video, Facebook for publishing updates on their lives, Flickr to upload images. More and more sites also allow the creation of structured data, most prominently Freebase, Drupal, and Semantic MediaWiki.

So far the publication of content has been further and further simplified, which has increased tremendously the amount of content on the Web. But no such trend of a comparable strength can be seen with regards to *functionalities* (be they called *APIs*, *services*, or *programs*).

Often functionality is also deployed as a binary or source code, thus burdening the user with the necessity to install them and connect them to their infrastructure. Publishing functionality on the Web today is often even harder. Developers need not only to program the functionality, but they also need to decide how to make it available to external users: which service interface should be used, how to implement them, which programming language to use, which service deployment framework to use, etc.

This paper suggests and sketches an infrastructure that enables the easy publication of functionality. It is obvious that the publication of functionality will never be as popular as the publication of content, but it can instead offer a whole new dimension to the Web.

2 Idea: a wiki for functionalities

Collaboratively developing functionalities within a wiki would have a number of advantages that are given by the wiki nature of the system:

1. Wikis are publishing platforms. The code would be immediately on the Web, and available for anyone to use.
2. (Semantic) Wikis will ensure that each functionality has its own URI, which is offered by the wiki, and that sufficient metadata about the functionality can be easily and partially automatically created.
3. Wikis retain the complete edit history. The development of the code is transparent and revisable.
4. Wikis allow for older versions to be referenced. This allows to use a specific version of the code, which might be further annotated, e.g. with a signature or with some other trust mechanism.
5. Wikis allow for easy and simple editing. Although a normal wiki should be extended with some code development environment features, like syntax highlighting or autocompletion.
6. Wikis allow for simple interlinking with the other parts of the wiki. In this case this would mean that the development on several pages will form a library of reusable code, and at the same time the wiki may contain textual content (i.e. documentation) but also data that can be used (in a semantic wiki).

Regarding the programming language of the code, there are obviously several possibilities. But one possibility strikes out as particularly interesting: using either JavaScript or a language that can be compiled to JavaScript, like CoffeeScript [3] or Java through Google's Web Toolkit. In this case the code can be run either by the wiki on the server side, it can be moved to the browser and run there, it can be moved to an application and run there (there are JavaScript engines available for every major programming language, most notably Java where it is part of the core language since version 6), or be pushed to the cloud and be run on a massive amount of computer nodes, as needed.

The interface for each functionality can be gathered by forms in the wiki, and thus service descriptions and wiki-internal parameter lists can be created automatically. This would turn the wiki into a full-fledged service publishing platform, where also semantic web service descriptions can be held ready, thus allowing external integration into outside tools and workflows.

3 Functionalities and the Semantic Web

Whereas the Semantic Web languages are useful for describing data for interchange, the self-description of the schema of such data, and for transporting the formal semantics of these schema terms in order to allow inferring further data and transforming from one schema to another, there is no standard way to connect functionality to such data objects. It's like object programming without methods.

Web services are the missing complement for this wide-spread publication of data. And whereas services have experienced quite some growth over the last year, they did not keep up with the explosive growth that we have seen in the Web of Data. This paper suggests to take the service publication process and, following the route that has been taken by wikis and blogs in simplifying the publication of content, simplify this process as well. A wiki that allows for the publication and execution of collaboratively developed functionalities can lower the requirements for creating services considerably, and thus possibly increase the possibilities in using and reusing the Web of Data in a significant way.

One of the possibilities to connect the Web of Data with these functionalities is simply the usage of URIs: every functionality published in the wiki will have an URI, and resolving the URI will return the code, possibly compiled in JavaScript. This code can then be executed by the application using the data (probably after some trust checks) and integrate the results in its knowledge base.

4 Example

Assume a functionality `population density`, assigned to an URI by the wiki like `wikifunc:population_density`, can be used for example by a SPARQL endpoint in the following fashion (following the ideas given by Linked Open Services [2] or Linked Data Services):

```
select ?popdense where {
  wiki:Germany wikifunc:population_density ?popdense
}
```

The functionality could be defined by the wiki as (the actual surface syntax of the programming language is not relevant here):

```
return getObject(?x, p:population) / getObject(?x, p:area)
```

The parameter `?x` would be defined in the header of the function (i.e. in forms outside of the actual code) and would be passed through by the SPARQL processing engine, which would download the JavaScript code from the wiki and execute it locally to create a value to bind to `popdense` in the original SPARQL query.

At the same time it would be a published service to call, e.g. via REST (or other mechanisms), using one parameter being the URI for a city that would be resolved against the Web of data and return the appropriate calculated value.

5 Related work

A number of programming languages have been developed for the Semantic Web. The most popular have been Fabl [1] and Adenine [4]. Both languages provide native support for RDF, provide at the same time a serialization in RDF, and the possibility to use different surface syntaxes. These features make the languages very accessible for many code analysis tasks. They do, on the other hand, not provide a publishing framework as suggested here. Nevertheless they could become a foundation for the language that will be embedded in the wiki.

The ScraperWiki[1] is a website for collaboratively building programs to extract and analyze public (online) data, in a wiki-like fashion. Scraper refers to screen scrapers, programs that extract data from websites. This is very close to the idea suggested here, but does not provide for the possibility to transport the code and execute it in different environments.

The Cloud 9 IDE[2] is a web based IDE that provides very extensive support for development in the cloud and also many collaboration features discussed here. It lacks an easy access mechanism to the Web of Data and the automatic integration into the Web of Services.

The basic building blocks of the idea presented in this paper are already in place, but the new combination will enable a novel venue for the simple publication of functionalities that has yet not been achieved.

6 Outlook

The idea to extend the Semantic Web with functionality is by no means new. We suggest a framework that we expect to enable the much wider deployment of functionalities, and a heavily increased involvement in their creation by a wider user community.

It is obvious that there will never be such a big community developing functionalities as there are communities developing textual content, uploading pictures, or videos. But there are strong hints that the current high barriers in publishing functionalities and in collaborating on them hampers their introduction and wide-spread creation at an unnecessary rate. There are many unresolved issues – Which is a good surface syntax? How can we ensure security? How do we handle trust in this environment? How to economically deal with the costs of executing these services? How to avoid versioning hell when executing the services?, and many others. But the simplicity of the idea and the expected quick pay off with even early prototypes of this architecture point to a possibly active and fruitful research project.

References

1. Chris Goad. Describing computation within rdf. In Isabel Cruz, Stefan Decker, Jérôme Euzenat, and Deborah McGuiness, editors, *The emerging semantic web: selected papers from the first Semantic Web Working Symposium*, pages 60–78. IOS Press Amsterdam, 2002.
2. Reto Krummenacher, Barry Norton, and Adrian Runte. Towards linked open services and processes. In *Third Future Internet Sympoisum*. Berlin, Germany, 2010.
3. Reuven M. Lerner. At the forge: introducing coffeescript. *Linux J.*, 2011, August 2011.
4. Dennis Quan, David Huynh, and David Karger. Haystack: A platform for authoring end user semantic web applications. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 738–753. Springer Berlin / Heidelberg, 2003.

[1] `http://scraperwiki.com`
[2] `http://c9.io/`