Semantic Web Approach to the GitHub Database Processing

Piotr Czekalski and Paweł Micek

Abstract—Despite the vast choice of RDFS (resource description framework schema) vocabularies being published in projects like LOV (Linked Open Vocabularies), there are still areas for which proper ontology has to be developed. GitHub, being both a reliable source of open Git repositories and a social platform, introduces new terms for which no proper description by means of the Semantic Web has been made yet. This paper regards every aspect of introducing a new set of semantic data: extracting the data, creating or refining the vocabulary, and possible ways to operate on properly integrated data set using SPARQL (SPARQL protocol and RDF query language). Authors of the paper would like to present a methodology of developing a unique RDFS vocabulary. Having this vocabulary, data extracted from GitHub can be described by the use of new terms as well as those from well-established ontologies, such as SIOC (semantically-interlinked online communities), DOAP (description of a project) or DC (dublin core).

Index Terms—GitHub, RDF, semantic web, semantic, SPARQL, vocabulary, WEB 3.0.

I. INTRODUCTION

The amount of data stored and provided to users through the WEB has exceeded the reasonable size years ago. The repositories are the case as well, because they are used not only in ICT market but more and more often for document editing and teamwork on a daily basis. Whether a user seeks particular content, browsers related social data, manages the projects or searches for relation among various properties and files in the repository or even estimates programmer’s productivity, processing the huge amount of the repository data with ease is essential. Here is where semantic presentation and semantic approach comes in handy.

One of the most popular and available systems present currently over the WEB is Git. A distributed revision control and source code management system. What makes GitHub special is that it provides a web user interface for browsing its database. It provides several functionalities, such as a user activity overview, follower system, or organizations that also make it a social platform of its own. Basic relations between data gathered by GitHub are presented by the platform itself. On the other hand, GitHub API provides a means of extracting the data and presenting it in a different, perhaps semantically cleaner, way.

RDF is a standardized way to do so. It is intended for situations in which information on the WEB needs to be processed and thoroughly understood by applications [1]. The format itself yields large overhead of information; however, semantic correctness of the dataset should be considered more important nowadays. Whenever the amount of sent metadata becomes a problem, another means of sending and processing it by third-party applications, e.g. JSON (JavaScript Object Notation), can be used.

Currently, no vocabulary describing unique ideas of GitHub has been published in open dictionary databases. However, there exist some classes and descriptions which may be reused. DOAP (Description of a Project) vocabulary describes terms strictly connected with version control layer: programming languages, repositories, project names, project pages, whereas SIOC (Semantically-Interlinked Online Community) and FOAF (Friend of a Friend) describe social platform terms, such as: user account, user ID, e-mail [2], [3].

II. EXTRACTING STRUCTURED DATA FROM GITHUB

A. Structured Data Model

Data is extracted from the GitHub database by means of the GitHub API with the use of an open source C# library - Octokit.net. The API itself is accessed over HTTPS and the data is both sent and received as JSON [4]. The number of requests per hour is limited; however, it is sufficiently high to obtain a representative dataset. The amount of gathered data can be further increased with the use of conditional requests and queries optimization.

The exemplary HTTP response from the API server is shown in the code snippet below.

```
$ curl -i https://api.GitHub.com/users/octocat/orgs
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 12 Oct 2012 23:33:14 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Status: 200 OK
ETag: "a00049ba79152d03380c34652f2cb612"
X-GitHub-Media-Type: GitHub.v3
X-RateLimit-Limit: 5000
X-RateLimit-Remaining: 4987
X-RateLimit-Reset: 1350085394
Content-Length: 5
Cache-Control: max-age=0, private, must-revalidate
X-Content-Type-Options: nosniff

Objects in JSON represent four main concepts of GitHub
```
system, namely:

- **Repositories** – every repository has its name (obligatory, shortened one), ID, and owner, being a user or an organization. Additional attributes (see Fig. 1), such as description or homepage URL, can also be specified.
- **Commits** – every commit to a repository is identified by its unique SHA token. Additionally, the object contains information about the author and the date of the commit.
- **Users** – the user object contains information about a user’s account, i.e., their login and ID. Further information about their activity, especially the list of repositories a user participates in, allows creating connections between users.
- **Organizations** – GitHub allows the creation of organization accounts, which are, similarly to user accounts, described by login and ID. Moreover, an organization can be the owner of a repository.

Fig. 1. The structure of an exemplary received repository object in JSON.

Access to the database is also limited by the way it is achieved. The list of all open repositories cannot be simply iterated, thus the dataset is accessible only on specific demands, e.g., by querying specific repository ID or searching for a given repository description. However, semantic data approach is suitable for an instantly and non-linearly growing dataset.

**B. Internal Structure of the Program**

Octokit.net library, while being still in early stages of development, provides almost a full reflection of functionalities provided by the API [5]. The structure of classes and methods resembles documentation of the GitHub API presented on the webpage. After a connection is set, data can be extracted from the JSON structured response.

The response is interpreted by Octokit.net internal routines (see Fig. 2) and then stored in memory for the time of operation of the program. Such data can be used to create RDF structures:

- **Triples**, in the form: `Subject + Predicate + Object`
- **Quadruples** (being, in fact, triples embedded in a specified context to avoid duplicates), in the form: `Context + Subject + Predicate + Object`
- **Graphs** – storing lists of triples. Triples can be freely added and removed. Moreover, basic set operations, such as unions and intersections between two graphs, are allowed.
- **Stores** – analogously to graphs, stores contain lists of quadruples.

- **Containers** – storing a group of homogeneous items (RDF literals or RDF resources) with the possibility of adding further elements to it [6].
- **Collections** – similarly to containers, storing a group of items, without the possibility of further changes.

Fig. 2. Technologies used to extract structured GitHub data and serialize it as semantic dataset.

Semantic data, originally saved as triples with use of RDFSharp library, can be exported to text files [7]. Although the library provides several types of syntax, we use RDF/XML, as it is a widely supported format [8]. Files with serialized triples grouped into collections describing repositories can be further processed. To ensure correctness of the content, data types of every field are explicitly declared and checked by RDFSharp internal routines. Exemplary serialized triples are shown in the snippet below.

```xml
<rdf:Description
    rdf:about="https://GitHub.com/12meses12katas/Enero-String-Calculator"
    dcterms:created
dateTime="2011-01-14T09:31:27"/>
<sioc:id rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">1253909</sioc:id>
<doap:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Enero-String-Calculator</doap:name>
</rdf:Description>
```

### III. DEVELOPING SPECIFIC ONTOLOGY

The ontology designed for describing data mined from GitHub incorporates existing vocabularies, such as DOAP,
SIOC or FOAF (see Fig. 3). It makes the semantic data accessible and understandable for a wider choice of third-party agents [9]. Moreover, new elements of vocabulary are introduced to express connections and objects not described earlier. The new ontology describes ideas strictly connected to GitHub, especially the system being its heart, Git.

The main goal is to create a vocabulary specific enough to describe the dataset properly and minimalistic enough, using vocabulary imports and extensions, to avoid duplication of attributes and classes [10].

### TABLE I: SCHEMA METADATA FIELDS

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>voaf:Vocabulary</td>
<td>Type of the resource which describes the vocabulary</td>
</tr>
<tr>
<td>vann:preferredNamespacePrefix</td>
<td>Gh</td>
</tr>
<tr>
<td>vann:preferredNamespaceUri</td>
<td>Location of the namespace: <a href="http://www.piotrczekalski.pl/ns/gh">http://www.piotrczekalski.pl/ns/gh</a></td>
</tr>
<tr>
<td>dc:title</td>
<td>Main vocabulary label: GitHub vocabulary (gh)</td>
</tr>
<tr>
<td>dc:description</td>
<td>The GitHub (gh) vocabulary, described using W3C RDF Schema and the Web Ontology Language. (note: descriptions in other languages may be added)</td>
</tr>
<tr>
<td>dc:issued</td>
<td>Date of the first vocabulary publication: 2014-06-19</td>
</tr>
<tr>
<td>dc:modified</td>
<td>Last vocabulary modification date</td>
</tr>
<tr>
<td>owl:versionInfo</td>
<td>The current version of the vocabulary: 1.0 (note: additional comment with rdfs:comment tag can be added to explain last changes)</td>
</tr>
<tr>
<td>dc:rights</td>
<td>The property rights of the vocabulary – additional cc:license property may come in handy</td>
</tr>
<tr>
<td>dc:creator</td>
<td>Creator(s) of the vocabulary: Piotr Czekalski, Paweł Micek (note: each creator can be a foaf:Person entity with all personal information included)</td>
</tr>
</tbody>
</table>

The FOAF vocabulary is designed to establish a decentralized language for describing personal and professional interests and social network linkage. We focused on the first part particularly, using FOAF to describe only basic personal attributes, e.g. name or homepage of a person. A special distinction between a person and a user account has to be made, as – theoretically – one person can create multiple accounts. This is where SIOC vocabulary comes in handy. Using SIOC, we can describe terms strictly related to GitHub as a social platform or, more precisely, online community. A user is explicitly defined as an entity distinct from a person, having an avatar, associated e-mail, ID etc.

To describe attributes related to repositories, DOAP vocabulary is used. It contains classes and attributes suitable for a description of a project in general; however, it lacks terms specific to GitHub as an open repository platform.

Aforementioned vocabularies are the base for the new RDF vocabulary. Although some attributes may be duplicated, the creator of a new vocabulary has the right to do so. Semantic data must be specific in the given context after all.

A new vocabulary should contain metadata describing the vocabulary itself [11]. A list of recommended properties is presented below (see Table I). Note the usage of various external vocabularies.

Having all necessary metadata of the vocabulary specified, we could add what is the main content of the schema – definitions of classes and properties (see Table II).

### TABLE II: SCHEMA DEFINITION FIELDS

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:label</td>
<td>Element main title, e.g. gh:Commit (note: title may be specified for more languages)</td>
</tr>
<tr>
<td>rdfs:comment</td>
<td>Role of the element (also may be specified for more languages)</td>
</tr>
<tr>
<td>rdfs:isDefinedBy</td>
<td>Explicit link between an element and the namespace it belongs to</td>
</tr>
<tr>
<td>vs:term_status</td>
<td>Element status, e.g. stable or testing (note: in case of deprecated elements dc:isReplacedBy may be used to identify the new element to replace the deprecated one)</td>
</tr>
</tbody>
</table>

### IV. ACCESSING THE DATASET

When serializing large amount of data, further optimizations concerning the size of the output files can be done. To reduce the overhead, several namespaces can be introduced with the use of xmlns attribute, so that the addresses of the vocabularies are not repeated. Special attention has to be paid to the risk of occurring naming conflicts.

In our particular case, a dataset is described using several vocabularies, including a new one. When accessing the dataset, prefixes have to be specified.

```sparql
PREFIX doap: <http://usefulinc.com/doap/ns/doap>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX gh: <http://www.piotrczekalski.pl/ns/gh>
PREFIX sioc: <http://rdfs.org/sioc/ns>
SELECT * WHERE {
  ?x doap:name ?name .
  ?x doap:homepage ?homepage .
  ?x sioc:id ?id .
}
```

The above example shows a basic SPARQL query listing...
all repositories having a homepage address specified.

V. CONCLUSIONS AND FUTURE WORKS

As the Web of data is growing, more attention has to be paid to organizing the data not only at the level of a single database, but also at a higher level of metadata. This is where Linked Open Data project comes in handy [12]. The quality of such data is improved by connecting related data that was not previously linked. However, to make this project possible, every creator of vocabulary has to follow recommendations specified by vocabulary aggregates (such as LOV). Another important aspect is the maintenance of the vocabulary: it is recommended that the author hosts the vocabulary schema and updates it to fit changing properties of the dataset.

Authors of the paper would like to extend or update created vocabulary, according to possible changes to the structure of GitHub services in the future. GitHub is a living project, thus certainly such changes will be needed. That is clearly one of the challenges a creator of a vocabulary has to face.

REFERENCES


Piotr Czekalski was born in Zabrze on 8th March 1975. He graduated from 1st Comprehensive High School in Gliwice and started engineering and master degree studies at the Silesian University of Technology in Gliwice, Poland. In 1999 he received his master of science, engineer titled on computer science (graduated with distinction) and then continued doctorate degree course in the same place. In 2004, Piotr finished his dissertation on automatic knowledge learning using hybrid fuzzy-evolutionary systems in Silesian University of Technology, Institute of Computer Science, Gliwice, Poland.

By the research, he continued his educational and commercial career, cooperating actively with corporate partners including Sygnity (formerly Computerland), Kamsoft, Tauron, Ernst and Young and others. The main research activity related to the AI hybrid technologies, particularly fuzzy and evolutionary ones, innovative user interfaces and automation control resulted in the following papers (among others): Evolution-fuzzy rule based system with parameterized consequences (Zielona Góra, AMCS, 2006), Touch-Less Java Based Hand Tracking System to Control an Artificial Arm (Switzerland, Trans Tech Publications, 2014), SMAC - Dataflow and Storage Modeling for Remote Personnel Identification in Restricted Areas (Kuala Lumpur, Springer, 2013).

Paweł Micek was born in Ruda Śląska on 14th October 1992. He graduated from Comprehensive High School in Ruda Śląska and started engineering studies at the Silesian University of Technology in Gliwice, Poland. He is a student of the macrocourse on automatic control and robotics, electronics and telecommunication, and computer science. His research interests include software engineering, data analysis and web applications development. Currently, he is working on his final year engineering project of a webcam time-lapse video observation website.