SemanGit: Adding a semantic layer to the Git protocol

A first application on top of GitHub

Dennis Kubitza
University of Bonn
denn_kubi@freenet.de ✉

Matthias Böckmann
University of Bonn
s6maboec@uni-bonn.de ✉

MA-INF 4314 - Lab Semantic Data Web Technologies

October 18, 2018

Abstract. Under the background of increasing capabilities of machine learning and web crawling it is important to add a meaningful structure behind data. In this project report, we present a way to add semantics to data provided by the version control system Git and the provider GitHub. We extract publicly available data from GitHub in the form of a knowledge graph and provide a corresponding ontology, which is compatible and expendable with other implementations of the Git-protocol. We discuss the benefits of creating such a structure, the ideas behind optimizing our knowledge graph and show how the resulting data can be utilized in multiple application sectors.

Contents

1 Introduction

Using a version control system (VCS) brings many benefits to developers who work on a project. It tracks file changes and is mainly used for collaborative programming. This helps in scaling the development. The most commonly used VCS is called Git [1] and was initially released in 2005. The Git protocol has been integrated into many integrated development environments. To keep track of file changes, Git requires a repository to which code can be submitted. Today, the largest Git repository hoster is GitHub with more than 28 million users and 85 million repositories in June 2018 [2]. This shows the enormous scale of the usage of VCSs. Regarding GitHub, there is abundant information available about the users, companies, projects, known issues of a project, bugs that have been found and fixed and so on. Github even provides components of social media, like the possibility to follow other authors and projects, form organizations and distribute development by sharing code and include code-fragments of other uses. All of this information can be queried from the GitHub REST API [3]. However, taking the direct route and querying the GitHub API has several drawbacks. Most importantly, there is a limitation of 5 000 queries per hour for a registered user. It would take many years to extract all every information via API calls, at least 23 months if every information is only queried once.

The GHTorrent project [4] has been extracting meta data from GitHub since 2013 with a multitude of API tokens to exceed the query limitations. They provide this dataset in the form of daily and monthly dump files. To get around the query limitations of GitHub, we have chosen to utilize the GHTorrent dataset. The information is provided in a relational structure, which we used as a basis for creating our knowledge graph.

Our task was to add semantics to this data. As a first step, we have created an ontology for Git and extended it to also take the social features of GitHub into account. An interactive visualization of our ontology can be found on WebVOWL [5]. We have then written routines for automatically extracting the data from GHTorrent and translating it into a knowledge graph fitting our ontology. All programs are published on GitHub [6]. Our main focus has been to reduce the storage size as much as possible while using the RDF format Turtle. This is required to have a queriable dataset, which still is multiple hundreds of Gigabytes large. In the end we managed to create a knowledge graph optimized to the point where we even beat the storage size of the original .csv format.

First of all, we will give an overview of related work in this field in section 2. In section 3, we will provide some background information about our work and the related fields. Afterwards, we give more detailed information about our approach and our implementation in section 4. The results of our optimization attempts are presented in section 5. Section 6 shows the value of the resulting data and how it can be utilized in real life scenarios. Finally, a conclusion is provided in section 7 with reference to future work on this field.

Everything that we created during this lab is already published on the Internet and can be accessed with following links:

Homepage (under construction) `www.semangit.de`

Github Page `https://github.com/DennisKubitza/MA-INF-4314_SemanGit`

Ontology `https://github.com/DennisKubitza/MA-INF-4314_SemanGit/blob/master/Theory/ontology/semangitontology.ttl`

Ontology Live Preview `http://visualdataweb.de/webvowl_editor/#iri=https://raw.githubusercontent.com/DennisKubitza/MA-INF-4314_SemanGit/master/Theory/ontology/semangitontology.ttl`

2 Related Work

While there has been a significant amount of research on ontologies and version control systems, we are, to the best of our knowledge, the first to apply semantic concepts to a VCS. While different contributions [7] [8] showed and discussed how VCSs can be utilized for the versioning of ontologies or even described new VCSs based on semantic data [9], no one attempted to create an Ontology for the Git-protocol.
Despite our effort to create an Ontology, we also intend to create and share the resulting Graph, extracted from GitHub. In the context of Data-extraction we already mentioned the established systems and most importantly Ghtorrent [4] and the Rest API [3]. The big favor of GHTorrent is, that they already agglomerated all necessary data of the query-per-hour limited Rest API through donated API-tokens. As their Data is open source, directly down-loadable and well-documented, Ghtorrent forms the backbone of our project. An alternative we considered, which was operating in a similar way is gh-archive [10]. Unfortunately, the data is also partitioned into daily, monthly and yearly dumps, while being only accessible with SQL-Queries (instead of raw data), together with a limit on payment-free data-queries.

3 Background

In itself, Git is simply a protocol for tracking file changes, such as insertions, deletions or alterations of lines of code, additions or deletions of entire files and more. These changes can then be submitted by *committing* the changes and *pushing* them to a repository. During the process of pushing the commit, Git will make sure that the changes are not in conflict with other changes that have been made, such as when two developers alter the same line of code. In the case of a conflict, Git offers multiple methods for resolving them. The commits also act as versions of the source code to which one can go back to. The repository does not need to be on a local machine. Remote Git repositories can therefore make the process of programming collaboratively significantly simpler.
The largest online provider[11] for remote Git repositories is GitHub. In addition to simply providing Git repositories, GitHub also adds a number of social features that are not part of the Git protocol, such as following other users,

watching project changes, creating release versions of a project and pull requests – a request for the contributors of a repository to adapt provided source code changes.

This project is called *SemanGit*, which is short for semantic Git. Our task was to add semantics to the Git protocol and its related fields. The backbone and initial step of our project was the design of an ontology in the web ontology language OWL, in order to help with logical inference. For example, if we define the predicate (relation) *committed_by* to be a link from a commit to a Git user, then we know that for every occurrence of this predicate, of what class the head entity and the tail entity is.
RDF [12] documents using an ontology are more machine-readable than most "traditional" documents, as, for example, the RDF document clearly distinguishes of what type entities are. Due to the vast size of GitHub, we chose to extract our data from this provider for creating our knowledge graph. They provide a REST API [3] as a query point from which one can gather data. This is however impractical as an approach for us due to the limited resources and time scope of the project. With its limitation of 5 000 queries per hour per token, it would take over 740 days to query all 89 million repositories just once, not providing us with nearly all the information included in our ontology.
The GHTorrent project [4] however provides large amounts of GitHub data which they have gathered using a multitude of tokens over several years. They provide a complete database dump every month and also daily incremental updates. So we adopted their dataset as input instead. While they already gathered data from GitHub, the GHTorrent data is simply stored in a relational model. This is not well suited for semantic web purposes, where a graph database is preferred instead. As the data provided GHTorrent already comprises several Terra byte of Data, we were forced to optimize our Graph Database as much as possible, while still ensuring valid Turtle syntax [13]. Beyond the documented standard we had to cope with additional restrictions of several query and validation tools [14, 15, 16] , that are necessary to ensure that our created data is still usable in practice by the most common tools.

4 Proposed Approach

In this section we will give more detailed information about the posed problem, our proposed solution and the implementation of it. For better readability, we split parts of this section into two subcategories: Ontology and software. The ontology parts contain information on the ontology development, whereas the software part covers the more practical aspects, such as the actual conversion and control flow of the resulting program.

4.1 Problem Statement

For our project, our task was to create an ontology that

- can encapsulate as much information as possible regarding Git and GitHub;
- is reusable by other researchers and extensible to other providers than GitHub and

- clearly distinguishes between what is part of the Git protocol and what has been added by GitHub, such as social features.

Using this ontology, we were supposed to create a knowledge graph with it, by feeding as much data to the ontology as time allows. We have also made it our task to work on the quality of the resulting RDF file by reducing its size as much as possible without breaking the RDF format. We also wanted to ensure the maintainability of our program, by creating fully automated scripts to process the monthly releases of GHTorrent [4].

4.2 Solution Strategy

Ontology

Due to the fact that we grab all our data from GHTorrent [4], we have decided to take their structure into account when developing our ontology, an illustration if which is displayed in figure 1. All relevant information provided by them has been included in our structure, too. It is computationally prohibitive for us to extract further information on large scale from the GitHub REST API [3] due to its query limitation.

We have ensured to clearly distinguish between Git properties and properties that do not belong to the Git protocol, but which are provided by GitHub. In this case, we have created a subclass that can hold all the additional information and make the basic Git class superclass of it. As an example, a "Git user" only consists of an email address. That is to say, when creating a commit, one can specify the author of the commit by providing an email address.
A GitHub user on the other hand is a lot more complex. It has a nickname, avatar, a creation date, a location etc. So in our ontology, there is a class called *User* which only has the *user_email* data field, and a *github_user* subclass that has 9 more fields, including those mentioned above.
On GitHub, users can leave comments on commits, pull requests and on reported issues. Seeing that those are very similar, we have grouped those three classes as *commentable*, such that a comment can be made for any class that is subclass of *commentable*.
Our ontology comprises of 22 classes and 80 properties. During the creation process, we used the WebVOWL editor tool [17] for building the ontology and the *OOPS! OntOlogy Pitfall Scanner!* [14] for checking the integrity of it.

Software

We will now describe the process of generating our knowledge graph according to the ontology described above.
The GHTorrent project provides updates on a monthly basis and on a daily basis. The monthly updates are full updates, whereas the daily ones are incremental. To have more resilient results, we have opted to process the monthly database dumps. A bash script handles the process flow, coordinating what needs to be downloaded, extracted, parsed, cleaned or read into a triple store. Also, should an error occur during one of these stages, the script automatically

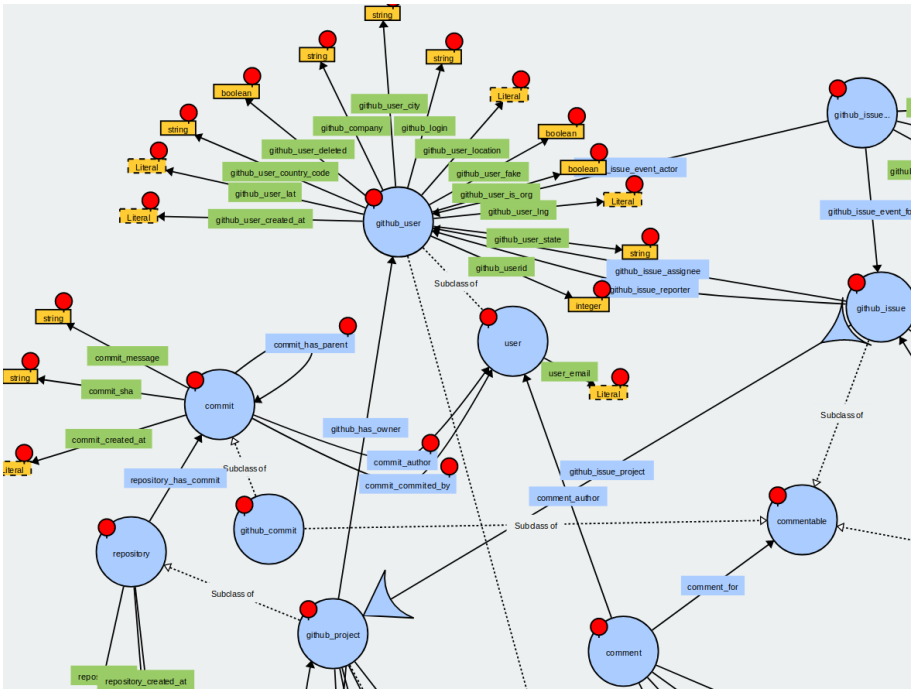

Figure 1: A small section of a simplified version of the ontology

detects at which stage the error occurred and can resume the process at a suitable point, without redoing work that has been finished successfully. Resource usage is also measured during all stages, see section *Evaluation*. The actual parsing is handled by a Java program, taking the extracted monthly database dump files, which are in .csv format, as input and transforming each file into RDF. To reduce the output size, we have chosen Turtle as output format. The Turtle abbreviations are particularly efficient, as many of the input files are presorted by some column.
Further improvements are achieved by using prefixes and by changing the integer representation of all IDs.

4.3 Implementation

We will now describe many of the points mentioned in the previous section in depth, providing some implementation details about the most important methods used by our approach.
First of all, our bash script scans the GHTorrent website automatically in a predefined interval to check for a new monthly dump file. In case of an update, we begin the download process. Once this is done, we use the pigz [18] library for extracting the .tar.gz archive with multiple processor cores. After extraction finished, the Java program for generating the RDF output is called. It also works in a multi threaded way, handling each file in a separate process. The bash script surveilles every started process for errors and takes care of automatic clean ups, to the last successful step of processing. This is also

automatically done, for any outdated or irrelevant data, like the extracted .csv files after a successful translation.
Our output format is Turtle, which we have chosen to be able to shorten the output. It allows the usage of prefixes to shorten frequently occurring URIs or frequent URI parts. Also, it allows us to abbreviate the subject of triples in case of consecutive triples sharing the same subject, and even both the subject and predicate can be abbreviated, in case of only the object being different in consecutive triples.
Due to the fact that the database dumps of GHTorrent are presorted, we can abbreviate many triples – sometimes a few hundred at a time. Regarding the prefixes, we are also able to excessively use this feature of Turtle to great effect. Originally inspired by [19], we used statistics to determine an optimal prefixing to lower our RDF size. Seeing that we are working with a fixed ontology, we simply added one prefix for every URI-type that comes up in it. The prefix names are also very short, as we use the alphabet $[a-zA-Z]$ for enumerating them, i.e. an integer representation to the logarithmic base 52. We have chosen the empty prefix for the most commonly used URI to save further space.
Another feature that we added for saving space is to change the representation of any integers that occur as part of URIs or as integer literal. We are using a Base64URL-style representation excluding the minus character, as it is illegal as leading character.

5 Evaluation

In this part we will evaluate the different optimization steps and alternatives with respect to storage space improvement and compare also the query time of queries with varying complexity on samples of the data to prove that our usage of abbreviations and base - conversion imposes no significant disadvantage. In fact, we can show that reducing the file size can improve the query performance. We restrict our analysis on datasets consisting of a certain fixed percentage of the newest monthly dump 2018-08-01 in order to boost the computation time while maintaining a representative sample. To limit all computation times to a treatable amount, our datasets for testing are composed of roughly $0.5, 1, 2$ and 4 GiB of data. Roughly, as we need still to maintain the integrity of each row. This is a limitation to about 1.35% of the original data of a single monthly dump.

5.1 Storage Saving through Intelligent Encoding

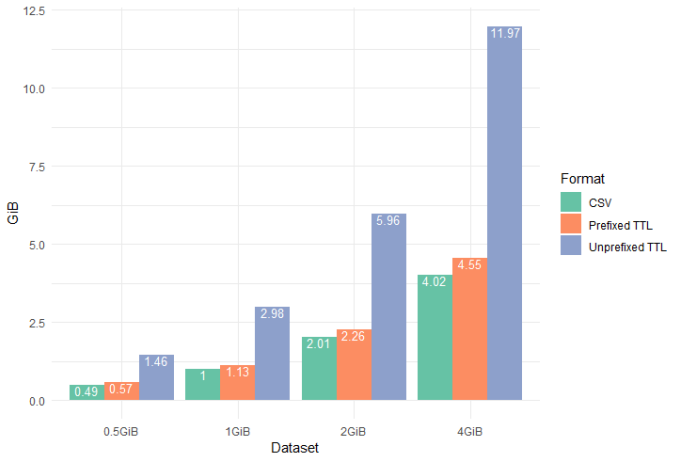

Figure 2: Format size comparison

In the first place we make use of Turtle's prefixing possibilities to transform each URI's non numeric content and each predicate to a string of maximum 2 characters. In figure 2, we see how the original .csv data compares to the .ttl format with and without making use of prefixing. We see that using prefixing as the first optimization step reduces the size of our data from an initial rate of 197.5% overhead on average compared to the original .csv files to 25.4% overhead using prefixing. For a single monthly dump of 284 GiB, this would result to an estimated improvement of 480 GiB in comparison to a file format like n-Triples, even without considering the use of abbreviations.

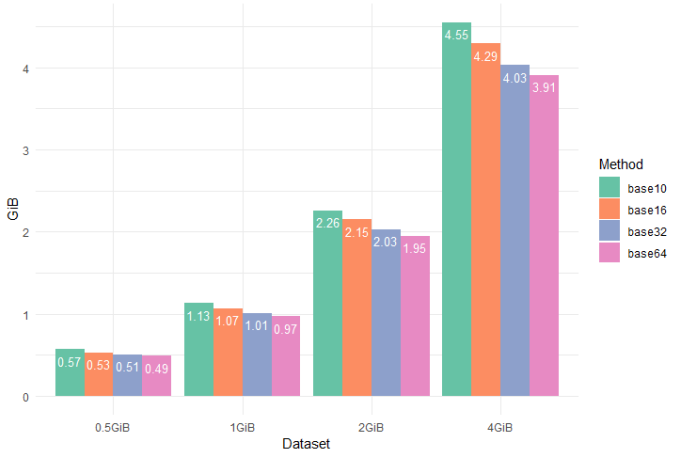

Figure 3: RDF size comparison of different ID encodings

As the second major improvement, we chose a base 64 encoding for all stored SQL-IDs, that we reused for generating the URIs. The resulting storage im-

provement is demonstrated in figure 3. A base 10 conversion denounces the same values of Prefixed TTL in figure 2. With a base 64 conversion, we managed to save 14% additional storage on average. Although the average ratio of 1.03 between base 32 and base 64 is still high enough to justify the attempt of using a base 128 conversion, it is not possible to improve further to base 128 due to the limitation set by ASCII.
In conclusion, we managed for now to store the RDF using only 97.8% of the required disk space from the original *.csv* files. Possible further improvements are discussed in the conclusion.

5.2 Run-Time analysis

We want to cover two different aspects of runtime analysis, but unfortunately can only provide the first one in this paper. We show that the base conversion and prefixing imposes no draw-back on the run-time on the translation process. We measured the whole data transformation process starting from already decompressed .csv data in table 1. As both extraction and conversion scale linearly in file size the results times 75 estimate the run-time of processing a monthly dump.

Table 1: Absolute Conversion Run-Time

No-Prefix	base10	base16	base32	base64
413	216	228	237	244

Runtime in seconds, 4GiB of Data
AMD Athlon X4 860K (4x4.0GHZ), 16GB DDR3-1866, 2TB HDD

We see that the subroutine with no prefixing takes almost twice as long. The bottleneck is definitely the I/O speed of the hard disk. After the application of prefixing, we see a slightly longer runtime for higher bases.
We assume that the improved file size will also lead to improved query time. Such an analysis would have been the second part of this subsection. We need to postpone this analysis to the point where we have a server with an adequate triple store.

6 Use Cases

While the general use cases of the SemanGit RDF do not differ from the one of the GitHub REST API [3] or GHTorrent [4], the underlying structure of semantic data imposes some favorable advantages to the already in section 3 mentioned flaws of both systems. Especially in cases where we want to find and analyses certain local structures in the global database, both API and SQL-database soon reach their theoretic capacities. The traditional way of defining use-cases as a sequence of interactions to achieve a goal is not usable for our case, as we don't provide a software system, but data. Nevertheless, we want to mention some interesting goals that different agents may exemplary target while using SemanGit.

The Headhuntress

Suppose that a user operates as headhuntress. Despite the public information concerning the required programming skills of a target person, the headhuntress also want to measure the quality of these skills to select the best programmer for her company or client. While it is easy to determine how high-quality programming is reflected on GitHub (no major issues reported, positive comments on projects, no infrequent commits), it is more problematic to actually measure and compare these traits. With SemanGit, she can find a representative subpopulation and generate benchmark results either by doing analysis manually or applying machine-learning.

The economists

One topic of economics is the analysis of driving forces, structures and institutions of an economy. While the behavior of agents in traditional scenarios are well-documented, the analysis of Open-Software-Projects and the motivation behind contribution is subject to notable current research [20]. For such research interests, the use of linked data offers new opportunities as it is tailored for the analysis of local models and offers direct access to empirical data on individual level.

Checking for Plagiarism

The re-usage of existing software solutions is common, popular and even intended in the area of Open-Source Software, but requires proper referencing and citing in most cases. Organizations like universities or software foundations could be interested in similarity analysis of other projects commit-structure, issues, comments, and history to determine plagiarism while avoiding an extensive search for similar code structure over software repositories. Especially repeated parallel occurrence of commits concerning bug-fixes over different projects could be an evidence for such plagiarism attempts.

7 Conclusions and Future Work

Within the course of the lab, we have created a tool for gathering vast amounts of data for Git and GitHub and present them in a semantic, but lightweight way. All in all we gathered semantic data consisting of approximately 12.6 billion triples and managed to store them with less space than our raw data in .csv format. During the work process, we have experienced much. We will talk about some key experiences in the Lessons Learned section.
In the Future Work section, we will finally also present some more ideas on how the tools could be improved further or how the result can be made more usable.

Limitations

Despite all our efforts to reduce the output size to a minimum, the amount of data is still vast. Many tools are unable to deal with files that are as large as

ours. Distributed software is required to provide enough computation power for even simple queries on the entire dataset.

Lessons Learned

During the development stage, we mostly used older dump files of GHTorrent, as they are smaller in size and easier to handle. In fact, it has been difficult to work on current dump files on personal hardware. Towards the end of the project, we then moved on to more recent input files, which then had a bunch of undocumented changes, breaking our conversion entirely. It has been a time consuming task to update our tools for newer files. We will therefore keep in mind not to rely on structure to remain unchanged as time progresses.
We have also attempted to use a binary format for our RDF output called Header Dictionary Triples (HDT) [21], [22] which is still under research. While it definitely did improve storage aspects, we encountered multiple issues and bugs, including it simply freezing when facing too many triples in the input. We have learned to be cautious when dealing with current research tools and also not to put too much effort into trying to make them run properly.

Future Work

At this point in time, our results only data available on GitHub. While GitHub is the largest provider for remote Git repositories, there are still more. Using the SANSA Stack [23], which should be able to deal with data of a scale like ours, one could apply certain machine learning algorithms to the data. For example linking entities of various Git repository providers or to link companies on GitHub with another database, such as DBpedia.
We have shown that increasing the base for the integer representation yields improved storage space. One could try to take this further and use much larger bases. To do so however, one would have to use non-ASCII characters, which require more space for storing in UTF8 format.

Acknowledgement

We would like to show our gratitude to our supervisor Damien Graux, Fraunhofer IAIS, for proposing this topic and supporting us with good advices, recommendations. positive and negative feedback at any time that we were in need of it.

References

[1] git - A free and open source distributed version control system. https://git-scm.com/about. Accessed: October 18, 2018.

[2] GitHub About. https://github.com/about. Accessed: October 18, 2018.

[3] GitHub API v3. https://developer.github.com/v3/. Accessed: October 18, 2018.

[4] G. Gousios and D. Spinellis. Ghtorrent: Github's data from a firehose. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pages 12–21, June 2012.

[5] SemanGit Ontology on WebVOWL. http://visualdataweb.de/webvowl_editor/#iri=https://raw.githubusercontent.com/DennisKubitza/MA-INF-4314_SemanGit/master/Theory/ontology/semangitontology.ttl. Accessed: October 18, 2018.

[6] SemanGit on GitHub. https://github.com/DennisKubitza/MA-INF-4314_SemanGit. Accessed: October 18, 2018.

[7] Tania Tudorache, Natalya F. Noy, Samson Tu, and Mark A. Musen. Supporting collaborative ontology development in protégé. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, and Krishnaprasad Thirunarayan, editors, *The Semantic Web - ISWC 2008*, pages 17–32, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[8] Timothy Redmond, Michael Smith, Nick Drummond, and Tania Tudorache. Managing change: An ontology version control system. In *OWLED*, 2008.

[9] Lik Cheung. Semantic version control system for source code, July 23 2013. US Patent 8,495,100.

[10] GH Archive . https://www.gharchive.org/. Accessed: October 18, 2018.

[11] Comparison of Git Providers. https://en.wikipedia.org/wiki/Comparison_of_source_code_hosting_facilities#Popularity. Accessed: October 18, 2018.

[12] Resource Description Framework . https://www.w3.org/TR/rdf11-primer/. Accessed: October 18, 2018.

[13] RDF 1.1 Turtle. https://www.w3.org/TR/turtle/. Accessed: October 18, 2018.

[14] OOPS! – OntOlogy Pitfall Scanner! http://oops.linkeddata.es/. Accessed: October 18, 2018.

[15] Blazegraph - a ultra high-performance graph database supporting Apache TinkerPop and RDF/SPARQL APIs. https://www.blazegraph.com. Accessed: October 18, 2018.

[16] Apache Jena - A free and open source Java framework for building Semantic Web and Linked Data applications. https://jena.apache.org/index.html.

[17] Steffen Lohmann, Vincent Link, Eduard Marbach, and Stefan Negru. Webvowl: Web-based visualization of ontologies. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 154–158. Springer, 2014.

[18] pigz – A parallel implementation of gzip for modern multi-processor, multi-core machines . `https://zlib.net/pigz/`. Accessed: October 18, 2018.

[19] Damien Graux, Louis Jachiet, Pierre Genevès, and Nabil Layaïda. The SPARQLGX System for Distributed Evaluation of SPARQL Queries. working paper or preprint, October 2017.

[20] Jean Tirole, Josh Lerner, and Jean Triole. The simple economics of open source. 2000.

[21] Javier D. Fernandez, Miguel A. Martinez-Prieto, Claudio Gutierrez, Axel Polleres, and Mario Arias. Binary rdf representation for publication and exchange (hdt). *Web Semantics: Science, Services and Agents on the World Wide Web*, 19:22 – 41, 2013.

[22] Miguel A. Martinez-Prieto, Mario Arias, and Javier D. Fernandez. Exchange and consumption of huge rdf data. In *The Semantic Web: Research and Applications*, pages 437 – 452. Springer, 2012.

[23] Jens Lehmann, Gezim Sejdiu, Lorenz Bühmann, Patrick Westphal, Claus Stadler, Ivan Ermilov, Simon Bin, Nilesh Chakraborty, Muhammad Saleem, Axel-Cyrille Ngomo Ngonga, and Hajira Jabeen. Distributed semantic analytics using the sansa stack. In *Proceedings of 16th International Semantic Web Conference - Resources Track (ISWC'2017)*, pages 147–155. Springer, 2017.