Working with Open Data

Materializing ALL the Internets

Robert Lehmann

robert.lehmann@student.hpi.uni-potsdam.de

overLODe

July 2, 2012
Bachelor thesis

CONTENTS

1 Background 1
2 Vision 2
3 Schema 4
3.1 Dataset Vocabulary Primer 4
3.2 Ontology 5
4 Architecture 6
4.1 Two of a Kind 7
4.2 Data providers 8
4.3 Generating statistics 9
5 Fault tolerance 10
6 Optimizations 11
6.1 Concurrency 11
6.2 Mimetype guessing 12
6.3 Data sparsity 12
6.4 Rate limiting 13
6.5 Chunking 13
6.6 Scraping 14
6.7 Data staleness 14
7 Results 15
8 Closing remarks 15

Hiermit erkläre ich, dass ich die vorliegende Arbeit in allen Teilen selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe. Alle Stellen der Arbeit, die ich aus diesen Quellen und Hilfsmitteln dem Wortlaut oder dem Sinne nach entnommen habe, sind kenntlich gemacht.

Robert Lehmann

ABSTRACT

Open Data has crossed the chasm of technical feasibility, with Semantic Web technologies well matured by now. However, harnessing it can still be an impervious adventure for users not intimately familiar with the surrounding processes and technologies.

We present a lifecycle management framework which glues together the formalities of the Semantic Web with the loose ends of Open Data. From a mere description of data sources, we collect and integrate heterogeneous data sets into a Semantic Web technology stack.

1 BACKGROUND

The Open Data movement has proven its point sufficiently and reached the Peak of Inflated Expectations in the Gartner Hype Cycle [1]. The scientific community around the technologies that power Open Data is thriving and well-established [2]: The *Extensible Markup Language* (XML) has seen widespread adoption,[1] the *Resource Description Framework* (RDF) is adopted as a Standards Specification,[2] ontologies spring up like mushrooms.[3]

Still, the community repeatedly identifies not technical matters but processes to be lacking[4] for end users— citizens, journalists, policy makers, or scientists —looking to utilize the huge amount of information buried in Open Data. Existing solutions frequently end up producing information silos; the solution space is more often than not difficult to explore for those unfamiliar with Semantic Web technologies. We have identified a number of problem domains which, currently, are tedious to instrumentalize:

DISCOVERY For exploration of meta data catalogs, the community currently sees uptake of the *Comprehensive Knowledge Archive Network* (CKAN). Ignoring one-off solutions,[5] CKAN seems to be the only contender. Software as a Service offerings such as *Socrata* are emerging.

[1] Research suggests that it is the most popular content type on the Web right after images and HTML pages. [3]

[2] Jeremy J. Carroll and Graham Klyne. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation. World Wide Web Consortium, Feb. 2004. URL: http://w3.org/TR/2004/REC-rdf-concepts-20040210.

[3] In 2011, there have been at least 1,500 ontologies indexed by the Semantic Web search engine *Swoogle*. [4, p. 7]

[4] The action items proposed at *Berlin Open Data Day 2012* [5] are hardly technical at all.

[5] A common misconception is that Open Data catalogs are, in itself, a novel idea. Public authorities keep complaining that they have built specialized information catalogs, such as *PortalU* (http://portalu.de/) for environmental monitoring, themselves long ago.

STORAGE Database layers are a completely different story — there are at least two large competing libraries: *OpenRDF Sesame* and *Apache Jena*. Underlying technologies, so-called triplestores, are aplenty and tend to build upon traditional SQL databases (e.g., *Sesame RDBMS Sail*, *Jena SDB*, or *Virtuoso*), file serializations (e.g., *N-Triples* and *RDF/XML*), or proprietary RDF representations (e.g., *OWLIM* and *Sesame Native*).

DELIVERY While the *SPARQL Protocol and RDF Query Language*[6] has its foot wide in the door, there are several implementations of it. Sesame ships with the *Sesame Server* which can be readily served by *Apache Tomcat*; *Virtuoso Universal Server* is another strong contestant in that area.

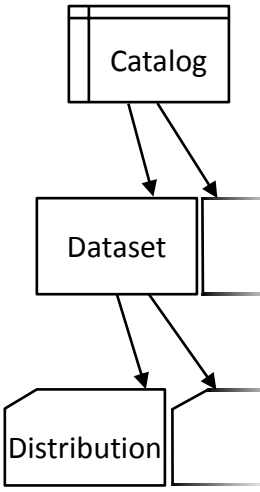
SENSEMAKING A very broad field in itself, we would not expect there to be a single answer to a multitude of tasks such as visualization, search, and analytics; such thing does and can not exist in the traditional database world either. For our work we will discuss the *Information Workbench* because it is (a) a Web platform and (b) supports collaborative authoring.

(For a list of project homepages, see Appendix A.)

⁂

In Open Data, the notion of **datasets** has established as a fundamental unit of measure. While there is still no common understanding what it entails, the consensus is basically that it is a defined, related, and closed set of entities and relationships.[7] As such, a dataset has no intrinsic representation. That's where **distributions** come into play: A distribution is a single serialization of a dataset.

A related set of datasets is commonly grouped into a **catalog**, which is an even fuzzier concept. Commonly, the source or the vendor of the dataset is used for catalogization.[8]



2 VISION

Switching contexts is hard. We strive to offer a one-stop solution which covers all of the aforementioned activities and integrates a full Semantic Web technology stack. While we think a stock solution for working with Open

[6] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF*. W3C Recommendation. World Wide Web Consortium, Jan. 2008. URL: http://w3.org/TR/2008/REC-rdf-sparql-query-20080115.

[7] Geographic or chronologic proximity, but sometimes pragmatic considerations such as filesize, are common bounds.

[8] For the Linked Open Data Cloud, grouping otherwise seemingly unrelated datasets into a single LOD catalog makes sense and is legit too.

Data is sorely needed we understand service providers might want to replace individual components and thus tried to build the framework as modular as feasible.

From the problem domains outlined in Section 1 we can derive tasks for a consolidated and automated workflow:

COLLECT (Discovery) In order to learn about which datasets are available we need to crawl catalogs and federate them in a central repository. We discuss a unified schema for metadata storage in Section 3.

DOWNLOAD (Storage) When a set of datasets has been selected, those need to be retrieved from external sources.

POPULATE (Storage) Working with data becomes much easier if it blends seamlessly into a Semantic Web technology stack. It must be transformed into RDF and loaded into a triplestore; we will use Sesame because it offers a pluggable backend mechanism —so-called *Sails*— which supports those triplestores which performed best in large deployments [6].

DIGEST (Sensemaking) With data replicated at our site, we can dissect it at our discretion (f.ex. to calculate histograms). Using the *Silk* framework[9] we link datasets with each other [7]; ad-hoc SPARQL queries can contribute vital statistics about the datasets.

SETUP (Delivery) Finally, the data needs to be served through SPARQL to be leveraged by other Semantic Web technologies. Sesame comes with its Sesame Server (and a user-facing administration tool called the *Sesame Workbench*) and can be deployed through standard Tomcat setups.

⁂

Tools like the *data package manager* (dpm) already handle significant portions of that very workflow (namely, it collects metadata from an index and fetches data packages through HTTP) but cease to offer whole lifecycle management solutions. A centralized solution like the *LOD Cache* is potentially harmful: Single points of failure do not contribute to the wellbeing of the overall system.

For our purposes we ignore data quality wholesale; we merely want to resolve integrative problems present in Open Data today.

USED DATASETS

For the purposes of this paper we have chosen some datasets which we believe should cover a representative range of different use cases and challenges.

[9] Julius Volz et al. "Silk – A Link Discovery Framework for the Web of Data". In: LDOW2009. Madrid, Spain, Apr. 2009. URL: http://ceur-ws.org/Vol-538/ldow2009_paper13.pdf.

Dataset	Homepage	Fetched on
Eurostat	http://estatwrap.ontologycentral.com/	2012/05/14
Worldbank	http://data.worldbank.org/	2012/05/15
Data.gov	http://explore.data.gov/	2012/05/15
LOD Cloud	http://thedatahub.org/group/lodcloud/	2012/05/21

Table 1: Sources used in our test runs

The Data Hub is a CKAN setup and the canonical host for the Linked Open Data Cloud[10]. The LOD Cloud comprises over 300 datasets covering media, geography, publications, government, life sciences, user-generated content, and cross-domain matters which are readily linked between each other. While CKAN is perfectly capable of hosting data sets as well, it only points to remote files from the LOD Cloud.

Eurostat, while freely available as comma-separated values (CSV) files, has been handily wrapped by OntologyCentral's Estatwrap to produce RDF output. It also adds links to the LOD Cloud to the 5,000 statistics on Europe.

Worldbank offers its own proprietary REST API with XML and JSON output for World Development Indicators on developing and high-income economies which it hosts itself. They are provided under a permissive license.

Data.gov is the US Federal Executive Branch's Open Data page hosted by Socrata. It encompasses about 4,500 datasets, in large parts hosted remotely.

3 SCHEMA

It is hard to come up with a universal schema for all catalogs as they all provide distinct sets of metadata. We need a least common denominator of metadata to fuel our tool and fetch datasets automatically, though. Every catalog is free to augment the standardized subset of metadata with its intricate information points itself. For interoperability with the Information Workbench, the metadata must be represented in RDF.

3.1 Dataset Vocabulary Primer

Fraunhofer FOKUS recommends a minimum set of required metadata: [5, p. 157f]

- title
- description
- publishing authority (author)
- format
- language
- license

[10] See http://lod-cloud.net/ for details.

CKAN established in its best practices [8, p. 11] that the *Data Catalog Vocabulary* (DCAT) should be used for general description of datasets; the Information Workbench has used that standard in the past, too.[11]

The types identified in Section 1 are consistent with DCAT's classes: Dataset, Distribution and Catalog. [9] It does not force any particular meaning upon them but defines a minimum subset of predicates useful for modeling their relationships. Additionally, it requires that Datasets must be "published or curated by a single source" and "available... in one or more formats." It defines certain types of distributions depending on their access method (namely Download, WebService and Feed.)

The *Vocabulary of Interlinked Datasets* (VoID) in turn requires that a dataset is a "set of RDF triples." [10] With that restriction in place it can partition, interlink and measure them.

3.2 Ontology

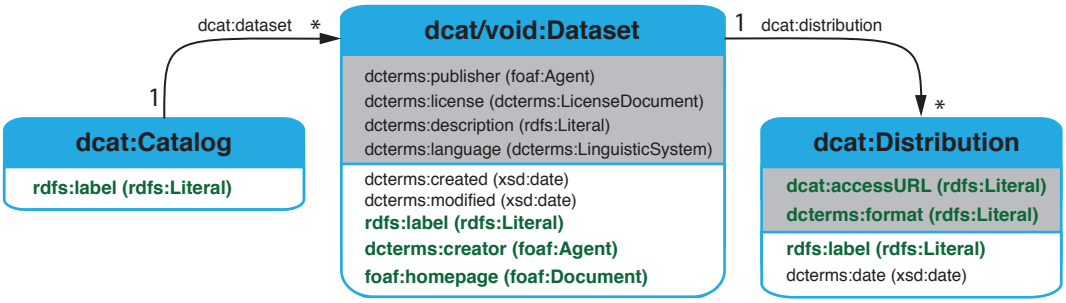


Figure 1: Entities and relationships in our Ontology [8, p. 12]. Required predicates are highlighted, those already present in the DCAT Specification are shaded.

In our representation, we basically use the datasets defined in DCAT with a grain of VoID; our schema is "DCAT conformant" as per the W3C Editor's Draft [9]. For better standards compliancy, we fall back to standard RDF predicates where possible and the widespread Dublin Core Terms[12] vocabulary for general information and source relationships between distributions.

For the purposes of automating the download and population of datasets, we distinguish three types of distributions: **Remotes** are distributions stored at third parties. These come directly from the metadata suppliers in the collection step. When we download them they become local **dumps** constituted by the `file` protocol; they should not differ from their originating remote with

[11] Check out `http://iwb.fluidops.com/resource/dcat:Dataset`.
[12] `http://dublincore.org/documents/dcmi-terms/`

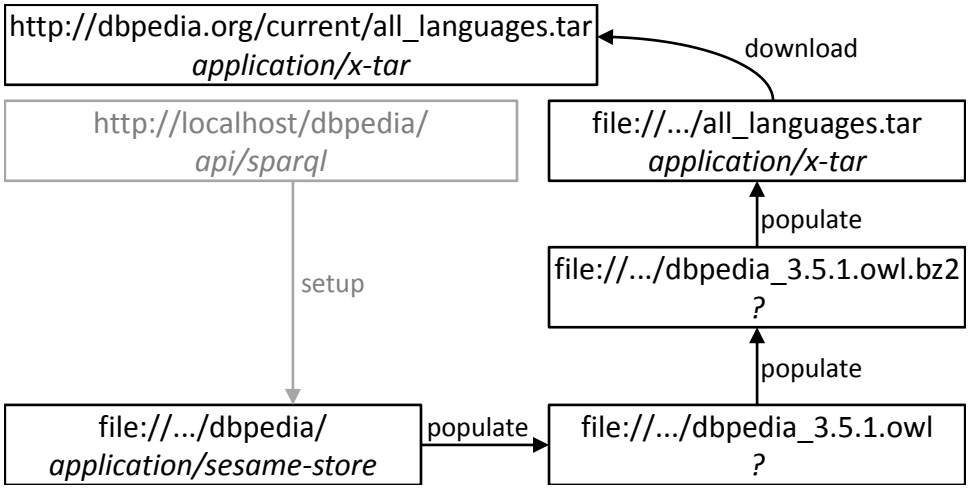


Figure 2: Different lifecycle stages of a distribution. The arrows are represented as dcterms:source relationships in our schema.

regard to contents. They are finally imported into Sesame **repositories** which are identified by the `application/sesame-store` MIME type.

We allow for easy provenance with explicit source links between each distribution.

The DCAT specification, as of the time of writing, specifies the dereferenceable access URL of a distribution as a literal instead of an URI. Not only does the name disagree with that stipulation but the Working Group is at odds about it as well.[13] We have tried to cope with both interpretations.[14]

We want to leave room for storing multiple datasets in a single repository. The *Service Description Vocabulary*[15] can be used to express in which named graph a distribution is saved in.

4 ARCHITECTURE

We have come up with a multi-layered architecture which (a) resembles our schema described in Section 3, (b) fulfills all tasks described in Section 2, and (c) satisfies both client- and API-style reuse (see 4.1). In this section, we

[13] `http://dvcs.w3.org/hg/gld/raw-file/27735149f5e9c/dcat/index.html#property--access-download`

[14] This dramatically complicates SPARQL queries: Both need to be cast into their lexical form using the SPARQL built-in `str`. In experiments with a simple equivalency query and a bogus 100,000 triple store, this seems to incur an 8% overhead.

[15] Gregory Todd Williams. *SPARQL 1.1 Service Description*. W3C Working Draft. World Wide Web Consortium, Jan. 2012. URL: `http://www.w3.org/TR/sparql11-service-description/`.

describe the absolute bare minimum effort we needed to pertain to guide a dataset through the whole lifecycle. For— partially critical —optimizations, see Section 6.

For retrieving the metadata catalogs we have employed the Information Workbench's on-board means. It already provided a CKAN harvester which uses the limited JSON interface. We have written providers for Semantic CKAN [8, ch. 4.2], Eurostat, Worldbank, and Socrata. They comply with our metadata schema and can be found in `com.fluidops.iwb.provider`.

Users are free to provide a minimal set of metadata manually as outlined in Figure 1 and our tool readily copes with downloading it too.

4.1 Two of a Kind

As described earlier, we feel the urge for a ready-to-use one-shot solution; professionals and experimenters might wish to use their favorite tool or library and only trigger certain aspects of the packaged workflow though. In order to cater to expert and novice users alike we have written both a Web-style and an Object-oriented programming interface.

The Web-style interface is a high-level way to command the framework. It generally provides non-trivial compound calls such as `downloadCatalog` or `populateAll` which return prepackaged statistics. For a full list of convenience methods, see `edu.overlode.util.ConvenienceClient`; for details on the provided statistics confer Section 4.3.

Underneath rests the Object-oriented interface which closely resembles the entities described by our ontology. It has a component for distributions and one for datasets[16].

Because we are usually confronted with large volumes of data, the framework is lazy by default: If a distribution has already been materialized we assert that data from a certain distribution is not a moving target — see Section 6.7 for why that is a bold assertion. We can thus go and run the tool incrementally. Restarting the tool and going from the last stable point (a metadata store commit) is, in practice, extremely useful to fix network hangups or mere programming errors on our or a third-party vendor's side.

Where possible we used bulk insertion instead of individual inserts.[17]

[16] Actually there are two parts to this: One component, `Dataset`, is capable of assessing a dataset in terms of our schema — it is a Dumb Data Object with the capability to fetch all related distributions. The `Loader` knows how to download and populate those.

[17] The `org.openrdf.repository.RepositoryConnection` method *add* supports streams and files natively.

4.2 Data providers

As per our schema we know which Internet media type a file is supposed to be in — this is, again, a bold assumption as we will discuss in Section 5 and 6.2. Based on that well-defined type we can dispatch program flow to a specially crafted component— a *provider* —and let it process the downloaded file.

Name	Capabilities[18]	Technologies used
RDFXML	rdf+xml	Rio
RDF	ntriples, turtle, n3, trix, trig	Sesame
Archive	tar, zip	Commons Compress
CompressedFile	gzip, bzip2	*ditto*
XML	xml	XPath, DOM [19]
JSON[20]	json	JSON in Java
CSV	csv	[11]

Table 2: Currently implemented providers in `edu.overlode.impl`

(For a list of project homepages, see Appendix A.)

Adding new providers is trivial: Providers just need to implement a method[21] accepting a local file and insert their generated triples into an open repository connection. For their convenience they are also handed a base URI (the document root) and a source distribution. If they wish to delegate new files, say extracted archive contents, back into the dispatching loop, they can yield a set of files back to the caller at their discretion.

Providers can then register for those MIME types they can operate on. The dispatcher orders providers by priority (first-come, first-served.) Distributions are handled in that order, too.

[18] MIME subtypes in the "application" type

[19] These are mashed up in `com.fluidops.iwb.provider.XMLProvider` to provide a manual mapping mechanism, a "mapping spec," from XML schemas to RDF.

[20] As a proof of concept we have written a dedicated provider for Socrata's JSON API, which spews out domain-specific vocabulary. JSON, as such, ends up in a very generic representation when transformed to RDF. We feel that special-purpose providers — or a mapping mechanism as used in the XML provider – is required to get anything remotely useful out of JSON.

[21] `List<File> handle(File dump, RepositoryConnection conn, String baseUri, Distribution dist)`

The Information Workbench has the notion of providers as well. The downside of its implementation is that it does not support direct streaming into a repository but needs to load all data into memory first.

4.3 Generating statistics

With over 9,000 individual datasets in our collective sample, comprehensive diagnostics are key. We can track consumed time, quantitative coverage, and occurred errors.

All statistics are stored in RDF[22], together with their parent distributions, for inspection purposes.

REAL-TIME STATISTICS The tool can take quite some time to run. We have implemented means to examine live runs as an extension to Java's `ThreadPoolExecutor`. It stores which dataset and distribution is handled by each thread and can present that information to callers.

Currently, the tool logs back a report every hour. An on-demand signal handler, implemented through proprietary Sun APIs, eventually had to go away as it did not fly with all JVM implementations. A possible optimization would be interaction with running threads through some sort of API.

REPOSITORY MEASUREMENTS VoID already developed SPARQL queries which can measure the size and extent of an endpoint.[23] Sesame's built-in hooks only allow for very limited measurements, namely the total number of triples.

SUCCESS METRICS While it is trivial to come up with a Definition of Done for distributions — did it run to completion without any exceptions? — it is hardly sensible to use that very same definition transitively on datasets. For datasets, success is no longer a binary but a fuzzy state where a single distribution is not fatal.

Currently, we just allow datasets to show up in both figures: Successful and failed datasets.

[22] as `rdf:comment` links

[23] http://code.google.com/p/void-impl/wiki/SPARQLQueriesForStatistics

5 FAULT TOLERANCE

In our experiments we have observed a lot of glitches[24] which can be categorized into the following error classes (roughly in order of appearance):

TRANSIENT NETWORK FAILURE Fetching a resource might fail due to the network being unreliable. A network link might not resolve, time out, or be down for whatever reason. If a problem persists, it might indeed be a permanent network failure. Sometimes, we hit a remote HTTP server but it replies with an error code.[25] Recovery from these errors is achieved through plain retrying.

PERMANENT NETWORK FAILURE More often than not, a resource is simply not persisted at the place where the catalog told us to look for it. HTTP has a special status code for that situation: 404 "Not Found." That is a sure-fire sign that this distribution is impossible to materialize.[26] Scraping can, under certain circumstances outlined in Section 6.6, rectify erroneous references.

CONTENT DISCREPANCIES Every distribution carries information about which format it is composed in. Human creativity seems umlimited in terms of their interpretation of RFC 2046[27]; files contain wholly different contents than advertised every now and then (f.ex. when they are transparently compressed). Mimetype guessing as described in Section 6.2 can be used to ameliorate that kind of failure.

RECOVERABLE ERRORS Partial corruption of a file such as encoding errors can destroy points of information. In simpler file formats skipping that particular area can restore a sane state.[28] Chunking is, despite other reasons outlined in 6.5, a good middleground between performance and coverage.

INCOMPATIBILITIES In the spirit of content discrepancies, there is a lot of leeway for interpretation in all areas of Semantic Web standards. Particular programs produce output which is not suitable for general consumption by other standards-adhering parties. The lacking popularity of SPARQL 1.1 (and the inevitable rise of proprietary extensions to SPARQL 1.0) contribute to the

[24] For specific accounts of which sources provoked those errors, see Petrick [8, ch. 5.1].

[25] In fact, Estatwrap often replies 503 "Service Unavailable" when it hits the Google App Engine quota after some 100 requests.

[26] For the LOD Cloud, we estimate that a third of all sources are affected.

[27] N. Freed and N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. IETF Draft Standard. RFC 2046. Internet Engineering Task Force, Nov. 1996. URL: http://www.ietf.org/rfc/rfc2046.txt.

[28] This works, for example, with the N-Triples format where every item is delimited by a newline character. Recovering a sane parser state after a hiccup in XML files in hardly possible. We conjecture only those file formats parseable by regular languages are truly automatically recoverable.

unlevel landscape. Contributing patches[29,30,31] or, for the time being, monkey-patching/forking help overcome these obstacles.

FATAL ERRORS For a certain class of errors there is just no sane technical solution. Some instances of content discrepancies fall in here: If no amount of fairy dust could ever detect a certain file format we are lost. Errors in XML files are often fatal as restoring parser state (or skipping— and detecting in the first place —faulty blocks) is incredibly hard if not impossible.

6 OPTIMIZATIONS

With the errors presented in the previous chapter in mind, we can come up with several optimizations to the minimal working prototype introduced in Section 4. Part of these optimizations are already implemented, others we propose for future work.

6.1 Concurrency

Waiting for a bunch of network operations to finish is an excellent fit for concurrency.

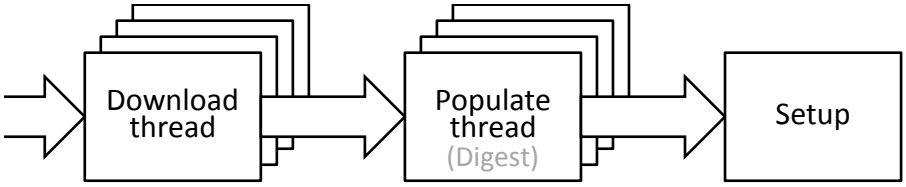


Figure 3: Program flow and synchronization points

We have parallelized the processing of individual datasets in the download and population steps. Intertwined concurrency — i.e., downloading one distribution, trying to populate it, downloading another one if it fails, and so forth — is hard because it requires jumping between the two steps. That would call for a task queue mechanism, dynamically enqueueing new operations as it receives success/failure acknowledgements.

[29] Sesame bug SES-959 "RDF/XML parser fails on relative URIs in rdf:datatype for empty nodes" (http://openrdf.org/issues/browse/SES-959)

[30] SES-1052 "HTTPRepository sends wrong binding prefix" (http://openrdf.org/issues/browse/SES-1052)

[31] Silk merge request #34733 "Support latest SPARQL 1.1 update syntax" (http://assembla.com/code/silk/git/reviews/34733)

Digestion could be separated from population as well. As statistics queries only takes up a fraction of time, that would not be a very worthwhile optimization and can well be done right when the repository has been populated.

6.2 Mimetype guessing

As described in Section 5, a third of all distributions carry a wrong format tag. Upon closer look, it turns out that they are only half-way wrong: The distributions do contain files in that format but are transparently archived and compressed.

In face of that discovery, we employ a mimetype guesser[32] by default. Under certain circumstances — i.e., when it fails entirely or does not recognize an XML vocabulary — that decision can be overridden as shown in Algorithm 1 below.

Algorithm 1 Logic employed to override the guessed mimetype

Given: $guessedformat, declaredformat$
Require: $declaredformat$ can be handled ▷ do not download at all otherwise
  if $declaredformat$ is not $guessedformat$ **then**
    if $guessedformat$ can not be handled **then** ▷ guessing failed, perhaps fall back to $declaredformat$
    end if
    if $guessedformat$ is XML and $declaredformat$ is subtype of XML **then**
      fall back to $declaredformat$ ▷ eg. RDF/XML
    end if
  end if

6.3 Data sparsity

If a dataset has multiple distributions, there are two possible semantics for them. They could be (a) different serializations of the same information or (b) multi-part uploads of distinct subsets from the pool of information[33]. In the wild, we also have observed (c) both at the same time, where some files are split into multiple distributions for volume reasons and others augment the dataset.

Separating one set of distributions is easy: Metadata. These are usually[34] tagged with the "meta" MIME type. We load these unconditionally.

[32] Such a program inspects the first few bytes of a file for *magic numbers* indicating the true file type. If that fails, they usually fall back to the file extension.
[33] 14% of all LOD Cloud datasets have multi-part downloads.
[34] There are exceptions, though, where ontologies are tagged as RDF/XML files.

We have tried to come up with sensible heuristics[35] to distinguish cases (a) and (b) but they all fall down at one dataset or the other. Erroneous metadata kills of those approaches entirely.

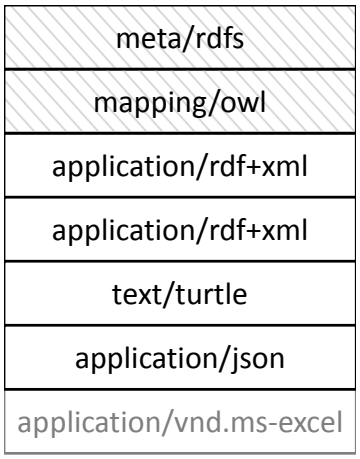


Figure 4: Example mime types. Metadata is grayed out, useless distributions are faded out.

6.4 Rate limiting

Services such as Estatwrap impose hard quota limits on third parties. While incremental runs of the tool are good enough to eventually fetch all of the datasets, techniques such as rate limiting and retrying could be employed to automate those multiple runs.

A distribution, if failed through a transient network error, could be appended to the queue of tasks again. For downloads from mixed sites that would eventually help to download all distributions. If the queue becomes very short (only few distributions left) or comprises only remotes from a single site, that would not help but anger the remote end because we are consistently hitting their quota limit.

For such situations, we could employ rate limiting to increase the time between two download attempts from a single site, f.ex. in exponential amounts of time.

6.5 Chunking

Data tends to break at seemingly unpredictable places. We could try to come up with contrived tactics trying to repair every possible bit of the files. We propose a simpler technique: Chunking.

For NTriples[36], we can accurately predict that every triple ends with a newline characters.[37] If the parser reports a fatal error, we can certainly contain

[35]
- Load distributions by priority, until one mimetype succeeds in its entirety.
- Select the majority mimetype.
- Peek into distributions to spot if they represent the same data.

[36] Jan Grant and Dave Beckett. *RDF Test Cases*. W3C Recommendation. World Wide Web Consortium, Feb. 2004. URL: http://www.w3.org/TR/rdf-testcases/#ntriples.

[37] Actually, every triple ends with a dot and a newline characters but the dot could already be part of the broken data. We did not observe any cases where newline characters where missing entirely.

the failure to any position between the last and the next newline character; all sibling triples are unaffected.

With that in mind there are now two different implementation strategies:

- ◇ Spoon-feed lines individually. This way, we can use the existing parser unmodified. For speed reasons, multiple lines could also be aggregated into larger blocks. That would save the repeated call overhead but loses more triples in case of an error. Blocks can also be distributed to multiple workers.
- ◇ Ignore erroneous lines. The parser needs to be changed to stop bailing at every single error, demoting syntax errors from fatal errors to normal errors. It drops only affected lines and neatly contains the error.

Sesame's Native Store is a pathological case in favor of the first approach: It queues up transactions until commit and keeps grinding the machine. Explicit commit points after every block, say 1K, help deflating that queue.

Chunking does not work for NTriples only. N3 and Turtle, as supersets of the NTriples format, could be made to work with slight modifications. A naïve implementation probably requires dot-newline as a separator because the subject context could be messed up otherwise.

6.6 Scraping

We have surveyed those distributions failing with a permanent network failure on how they could be recovered from existing metadata. In 28% of the cases, scraping the parent directory of the declared endpoint linked to the file in demand.

We did ultimately reject this optimization because it does not improve the situation much in absolute numbers and a clean solution involves the metadata maintainers getting their act together.

6.7 Data staleness

The tool is lazy by default in multiple places: Downloads do not override existing files, nor are they even attempted if a remote already has a corresponding dump. Population does not take place if it has already been successful, too.

This particular behaviour does not fit datasets with no versioning strategy such as Eurostat, Worldbank and Data.gov: Updates are just brought into existing distributions. Downloaded files might thus contain outdated information. We have provided hooks[38] which can force re-downloading and re-parsing of distributions.

[38] A `forceDownload` and a `forcePopulate` flag in the Object-oriented interface.

7 RESULTS

Eurostat and Worldbank are very homogeneous catalogs: If you succeed managing a single dataset's lifecycle, you can easily replicate that procedure for others — it is all or nothing. Except for deprecated datasets, metadata was accurate; content discrepancies do not occur if all you have is but one file format. In the limits of our quota constraints, we have full coverage for both catalogs.

As presented in Petrick [8], we can materialize roughly a third of the LOD Cloud. Mimetype guessing proved to be worth its salt and rectified all content discrepancies present. The LOD Cloud, to be fair, is already covered by two thirds with SPARQL endpoints from its vendors though.

Data.gov includes many distributions in non-machine-readable formats. We succeeded to load roughly 30G of data from almost 300 different datasets.

Especially on single-core machines those processes are painfully slow; a single dataset can take up to a day to be handled completely.

8 CLOSING REMARKS

Our findings match those surveyed by Socrata in 2010[39], where half of all developers found data to be in unusable formats and two thirds of all developers stated data is not easily accessible. Especially heterogeneous catalogs have difficulties maintaining metadata quality and would do good with enforcing some kind of quality assurance.

We have presented a tool which lays the foundation for working with Open Data at large. We can cope with homogeneous sources just fine and discussed strategies to enhance coverage for heterogeneous ones as well.

REFERENCES

[1] Steve Bittinger. *Hype Cycle on Government Transformation*. Tech. rep. Gartner, Inc., July 2011. URL: http://www.gartner.com/id=1745021.

[2] Tim Finin. *The Semantic web's place on the Hype Cycle*. Blog. University of Maryland, Baltimore County. Aug. 2005. URL: http://ebiquity.umbc.edu/blogger/2005/08/25/the-semantic-webs-place-on-the-hype-cycle/.

[39] https://benchmarkstudy.socrata.com/Developer-Survey/Developers-Rate-the-Current-State-of-Gov-Data-Acce/4j6z-qhyp

[3] Sunghwan Ihm and Vivek S. Pai. "Towards Understanding Modern Web Traffic". In: *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. IMC '11. Berlin, Germany: ACM, Nov. 2011, pp. 295–312. ISBN: 978-1-4503-1013-0. DOI: 10.1145/2068816.2068845. URL: http://doi.acm.org/10.1145/2068816.2068845.

[4] M. A. Sicilia et al. "Empirical Findings on Ontology Metrics". In: *Expert Syst. Appl* 39.8 (June 2012), pp. 6706–6711. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2011.11.094. URL: http://dx.doi.org/10.1016/j.eswa.2011.11.094.

[5] Wolfgang Both, Ina Schieferdecker, et al. *Berliner Open Data-Strategie*. German. Tech. rep. Fraunhofer FOKUS, June 2012. URL: http://berlin.opendataday.de/berlin-open-data-handlungsempfehlungen-fur-nachhaltigkeit/.

[6] Mike Personick, Kingsley Idehen, et al. *Large Triple Stores*. Wiki. World Wide Web Consortium. Aug. 2011. URL: http://www.w3.org/wiki/LargeTripleStores.

[7] Benjamin Reissaus and Christian Godde. "Dataset Exploration in the Information Workbench based on Links and Tags". June 2012.

[8] Dominic Petrick. "Experiences and challenges in working with the Linked Open Data Cloud". June 2012.

[9] Faadi Mali, John Erickson, and Phil Archer. *Data Catalog Vocabulary (DCAT)*. W3C Working Draft. World Wide Web Consortium, Apr. 2012. URL: http://www.w3.org/TR/vocab-dcat/.

[10] Keith Alexander et al. *Describing Linked Datasets with the VoID Vocabulary*. W3C Group Note. Semantic Web Interest Group, Mar. 2011. URL: http://www.w3.org/TR/void/.

[11] Magdalena Noffke. "From CSV to RDF — Converting CSV Documents to use them in the Information Workbench". June 2012.

A MENTIONED TECHNOLOGY

CKAN	http://ckan.org/
Socrata	http://socrata.com/
Sesame	http://openrdf.org/
Jena	http://jena.apache.org
Sesame RDBMS Sail	http://openrdf.org/doc/sesame2/api/org/openrdf/sail/rdbms/RdbmsProvider.html
Jena SDB	http://jena.apache.org/documentation/sdb/index.html
Virtuoso	http://virtuoso.openlinksw.com
N-Triples	http://w3.org/2001/sw/RDFCore/ntriples/
RDF/XML	http://w3.org/TR/REC-rdf-syntax/
Sesame Native	http://openrdf.org/doc/api/org/openrdf/sesame/sailimpl/nativerdf/NativeRdfRepository.html
OWLIM	http://ontotext.com/owlim/
Sesame Server	http://openrdf.org/doc/sesame2/users/ch06.html
Tomcat	http://tomcat.apache.org/
Information Workbench	http://fluidops.com/information-workbench/
dpm	http://okfn.org/projects/datapkg/
LOD Cache	http://lod.openlinksw.com
Silk	http://www4.wiwiss.fu-berlin.de/bizer/silk/
Rio	http://openrdf.org/doc/rio/api/overview-summary.html
Commons Compress	http://commons.apache.org/compress
JSON in Java	http://json.org/java/

B ONTOLOGY

```
@prefix dcat: <http://www.w3.org/ns/dcat#>.
@prefix void: <http://rdfs.org/ns/void#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix dcterms: <http://purl.org/dc/terms/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
```

```
@prefix : <http://data.fluidops.net/resource/>.

:Ontology
  rdf:type owl:Ontology;
  dcterms:title "Schema_for_Datasets_and_Distributions";
  dcterms:creator :Team;
  dcterms:modified "2012-02-29"^xsd:date;
  foaf:homepage <http://data.fluidops.net/>;
  owl:imports void:, dcat:.  # importing DCAT and VoID

dcat:Catalog owl:equivalentClass [
   rdf:type            owl:Class;
   rdfs:isDefinedBy    :Ontology;
   owl:intersectionOf  (
      [rdf:type owl:Restriction;
       owl:onProperty      rdfs:label;
       owl:someValuesFrom  rdf:Literal]  # at least one rdfs:label
      [rdf:type            owl:Restriction;
       owl:onProperty      dcat:dataset;
       owl:someValuesFrom  dcat:Dataset]  # at least one dcat:dataset
   )
].

dcat:Dataset owl:equivalentClass [
   rdf:type            owl:Class;
   rdfs:isDefinedBy    :Ontology;
   owl:intersectionOf  (
      [rdf:type owl:Restriction;
       owl:onProperty      rdfs:label;
       owl:someValuesFrom  rdf:Literal]  # at least one rdfs:label
      [rdf:type            owl:Restriction;
       owl:onProperty      foaf:homepage;
       owl:someValuesFrom  foaf:Document]  # at least one foaf:homepage
      [rdf:type            owl:Restriction;
       owl:onProperty      dcterms:creator;
       owl:someValuesFrom  foaf:Agent]  # at least one dcterms:creator
      [rdf:type            owl:Restriction;
       owl:onProperty      dcat:distribution;
       owl:someValuesFrom  dcat:Distribution]  # at least one dcat:distribution
      [rdf:type            owl:Restriction;
       owl:onProperty      dcat:distribution;
       owl:someValuesFrom  dcat:Distribution]  # at least one dcat:distribution
      [rdf:type            owl:Restriction;
       owl:onProperty     [owl:inverseOf dcat:dataset];
       owl:someValuesFrom  dcat:Catalog]  # at least one associated dcat:Catalog
   )
].

void:Dataset owl:equivalentClass dcat:Dataset.

dcat:Distribution owl:equivalentClass [
   rdf:type            owl:Class;
   rdfs:isDefinedBy    :Ontology;
   owl:intersectionOf  (
```

```
      [rdf:type            owl:Restriction;
       owl:onProperty      rdfs:label;
       owl:someValuesFrom  rdf:Literal]  # at least one rdfs:label
      [rdf:type            owl:Restriction;
       owl:onProperty      dcat:accessUrl;
       owl:someValuesFrom  rdfs:Resource]  # at least one dcat:accessUrl
         # overriding the spec'd rdfs:Literal
      [rdf:type            owl:Restriction;
       owl:onProperty      dcterms:format;
       owl:someValuesFrom  dcterms:MediaType]  # at least one dcterms:format
         # narrowing down the spec'd dcterms:MediaTypeOrExtent
  )
].
```