IndeGx: A Model and a Framework for Indexing RDF Knowledge Graphs with SPARQL-based Test Suits

Pierre Maillot[a,*], Olivier Corby[a], Catherine Faron[a], Fabien Gandon[a], Franck Michel[a]

[a]*University Cote d'Azur, Inria, CNRS, I3S, Nice, France*

Abstract

In recent years, a large number of RDF datasets have been built and published on the Web in fields as diverse as linguistics or life sciences, as well as general datasets such as DBpedia or Wikidata. The joint exploitation of these datasets requires specific knowledge about their content, access points, and commonalities. However, not all datasets contain a self-description, and not all access points can handle the complex queries used to generate such a description.

In this article, we provide a standard-based approach to generate the description of a dataset. The generated descriptions as well as the process of their computation are expressed using standard vocabularies and languages. We implemented our approach into a framework, called IndeGx, where each indexing feature and its computation is collaboratively and declaratively defined in a GitHub repository. We have experimented IndeGx on a set of 339 RDF datasets with endpoints listed in public catalogs, over 8 months. The results show that we can collect, as much as possible, important characteristics of the datasets depending on their availability and capacities. The resulting index captures the commonalities, variety and disparity in the offered content and services and it provides an important support to any application designed to query RDF datasets.

Keywords: semantic index, metadata extraction, dataset description, endpoint description, knowledge graph

1. Introduction: the Need for Dataset Descriptions

Since the initial proposal of standard practices for publishing linked data on the Web, the number of datasets published according to these principles has increased every year. These datasets are growing in number, size, and heterogeneity, covering fields ranging from biology to bibliography, geography, government statistics, and more.

Any application that combines several of these datasets relies for its development on some knowledge of their contents and the links existing between them, whether through shared vocabularies or common topics. Furthermore, the rise of the FAIR principles [27] in recent years increases the need for detailed metadata. However, it is usually difficult to have such a clear and up-to-date description of a dataset. Yet these descriptions are necessary to enable the selection of relevant datasets and their joint interrogation in many use cases.

In practice, and as a result, the use of a dataset accessible on the semantic web via a SPARQL access point currently requires prior exploration of its content by the application designers. This manual exploration makes it possible to decide on the relevance and to detect certain compromises to be made in its use to avoid overloading its endpoint or reaching its limits. This exploration is tedious and requires a good knowledge of semantic web technologies.

In most cases, the data providers would be responsible for describing their datasets. But this also requires specific efforts, costs, and skills that some providers, who are not necessarily experts in semantic web technologies, do not have or cannot afford. In particular, these descriptions rely on a deep understanding and joint use of specialized vocabularies and there is no standard model or tool for generating and updating these descriptions.

Depending on use cases and applications, the needs in terms of dataset descriptions may be different. Catalogs, such as the data portals based on the CKAN or OpenDataSoft technologies, present human-readable entries for each dataset. The content of those entries is based on metadata such as the provenance and the general themes of a dataset. Exploration approaches, such as the dataset dashboard [18] and LOUPE [20], present statistics about the content of a dataset to users. Some federation approaches, such as HiBiSCus [24], also rely on statistics on the content of a dataset. They generally are more extensive than the ones used in exploration approaches. Monitoring approaches, such as SPARQLES [25] and Yummy Data [30], present measurements of different features of datasets and

*Corresponding author

Email addresses: `pierre.maillot@inria.fr` (Pierre Maillot), `olivier.corby@inria.fr` (Olivier Corby), `catherine.faron@inria.fr` (Catherine Faron), `fabien.gandon@inria.fr` (Fabien Gandon), `franck.michel@inria.fr` (Franck Michel)

endpoints and their evolution through time.

Considering the current situation of linked open data, the main research question we address in this paper is: *is it possible to use only open technologies and standards to effectively generate a useful index of datasets exposed through a public SPARQL endpoint?* In the next sections, we will break down this broad question into several incremental sub-questions : (i) what should be the model and vocabularies to represent the metadata of an open index? (ii) how can we formalize the metadata generation process itself in a declarative yet operational way? (iii) what is the quality and coverage of the metadata we obtain in practice? and (vi) does it effectively support other applications?

Our answer to these questions is IndeGx, a declarative and extensible framework to generate remote dataset descriptions. We only target the datasets accessible through a SPARQL endpoint and in the rest of this paper, we call a Knowledge Base (KB) the pair of an RDF dataset and its SPARQL endpoint. Other approaches have proposed methods to generate dataset descriptions by interacting with the SPARQL endpoint of a KB. The novelty of IndeGx is the set of declarative rules in RDF it is based on to generate metadata in RDF. The interactions between our framework and a KB rely only on standard SPARQL queries. The usage of declarative rules and existing standards makes IndeGx both usable with the greatest number of KBs and easily extensible by the modification and the addition of new rules on a GitHub repository. Part of the rules used by IndeGx cover the metadata features found in other approaches. We also proposed original rules to generate new kinds of metadata, such as the statistics on the usage of namespaces in the KB.

We experimented IndeGx on 339 endpoints over 8 months to test its capacity to index KBs through time and to evaluate our result over an extensive period of time and panel of KBs.

This paper is organized as follows. In Section 2 we first present and discuss the related works. In Section 3, we present our proposed model for the description of an RDF dataset. In Section 4 we present the IndeGx metadata generation framework and the formal representation of its processes. In Section 5 we present the performance of IndeGx during our experimentation on a large set of KBs available on the web. Finally, conclusions and future works are presented in Section 6.

2. Related Works and Positioning

IndeGx is a *general* framework to generate a KB's description containing data for use in multiple cases, including federated querying, monitoring, catalog building. Other approaches consider the description of a KB for some of these usages. The survey on dataset profiling by Ben Ellefi et al. [2] presents a typology of possible descriptions for a KB according to which our IndeGx framework generates a machine-readable semantic index of KBs, where each KB description contains extracted information about the domain, links, licensing, and provenance of the KB and computed statistical metadata. Additionally, IndeGx gives the detailed provenance of the content it generates, which is not a feature found in the above-mentioned survey. In this section we provide a brief overview of approaches that are related to IndeGx.

2.1. Vocabularies and standards for KB metadata

In the state of the art, there is not just one single vocabulary used to describe a KB but several standard vocabularies that co-exist to describe either the KB content, its endpoint, or other features. The VoID vocabulary [8] is the most well-established vocabulary to describe datasets. It is described in a W3C interest group note standardizing the description of the content of a KB and its relations with others. It is used by many semantic web applications like the LOD Laundromat [1] and SPORTAL [15], and several other approaches surveyed by Ben Ellefi et al. [2]. The SPARQL-SD vocabulary [28] is a W3C recommendation to describe SPARQL endpoints together with their dataset's default and named graphs. The DCAT vocabulary [29] is a W3C recommendation to describe datasets, data services and data catalogs. These three vocabularies are complementary and are commonly used together. Moreover, they rely on other more general metadata vocabularies such as Dublin Core Terms and FOAF to complete their descriptions.

There are different ways to automatically retrieve the descriptions created with the VoID and SPARQL-SD vocabularies, such as the "well-known uris" mechanism and the SPARQL protocol. Collective efforts to make available the descriptions of KBs via a SPARQL endpoint exist, e.g. Wikidata [26], but are only on a voluntary basis. The Google Dataset Search service relies on an index of Web data sources automatically created: it crawls the web searching for schema.org metadata in web pages to identify web pages describing a dataset. While it does not require index maintainers to volunteer, it still requires the web pages to be properly annotated.

2.2. KB catalog

SPORTAL [15] proposes a catalog of KBs providing a description of the content of each KB. It can be exploited to select a KB to send queries to. The different statistical measures about the content of the KB are extracted with SPARQL queries to create a VoID description. Some metadata like authorship or vocabulary are not considered, nor SPARQL-SD descriptions as it was a recent technology at the time. The experimentation conducted to construct SPORTAL showed that the capacity of each SPARQL endpoint to answer queries limits the construction of the catalog. In the IndeGx framework, We reuse the queries used in SPORTAL to extract statistical measures of the content of each KB. When compared to SPORTAL, IndeGx provides more information, such as the endpoint description or the authorship. More importantly, it overcomes some of the limitations of SPORTAL through a series of queries with different levels of complexity and accuracy to get the same information, and it unloads parts of the computation of the result of the queries by the targeted KB through the use of a federation server configured to allow queries whose evaluation is particularly long, and for which most public endpoints would simply time out.

LODLaundromat [1] adopts a centralized approach as it crawls the semantic web for datasets, cleans them by removing non-standard resources and triples and by enforcing community guidelines, then hosts the clean version and offers VoID metadata about the new version through a SPARQL endpoint.

SpEnD [31] relies on a "search keyword" discovery process to discover SPARQL endpoints with relevant keywords. It then evaluates their performances and features, including their SPARQL coverage, similarly to SPARQLES (see Section 2.3).

2.3. KB monitoring

Our proposal can also be compared to monitoring frameworks that offer access to the metadata extracted from each base, like SPARQLES [25] or Yummy Data [30]. SPARQLES sends SPARQL queries and HTTP requests to assess the availability, performance, SPARQL compatibility, and discoverability of a set of 557 endpoints taken from datahub.io. IndeGx reuses the SPARQLES' SPARQL queries to assess the availability and SPARQL coverage of an endpoint. As it relies on a SPARQL-based test suite, it cannot assess features whose computation do not or not only rely on SPARQL queries: it is the case for the discoverability assessed by looking for server names and `.well-known` file, and the performance assessed by comparing the results of a batch of queries with or without cache.

Yummy Data is a catalog of KBs for biomedical science that uses different measures, among which those of SPARQLES, to rank the KBs in its catalog according to a score defined by the authors. This "Umaka score" of a KB is based on six aspects: availability, freshness, operation, usefulness, validity, and performance. IndeGx reuses Yummy Data's tests for freshness, usefulness, and validity. It cannot implement some Yummy Data's indicators that do not rely on SPARQL queries, but on low-level operations, e.g. the "operation score" relying on simple HTTP queries.

Luzzu [9] is a framework enabling experts to define quality metrics about KBs and to evaluate them automatically. Like IndeGx, Luzzu is extensible and is based on existing vocabularies to define the measures and their results. When compared to Luzzu, IndeGx can provide quality indicators similar to the ones in Luzzu if they can be expressed using SPARQL queries.

2.4. KB exploration

Dataset Dashboard [18] is a tool[1] for the exploration of over 200 KBs. It gathers the statistics about classes and properties that can be expressed using VoID. It also displays a graphical representation of a summary of the schema of a KB. Metadata on a KB is also used to support users during KB exploration and query writing. Approaches such as SPARKLIS [11] and LOUPE [20] present dataset statistics for the exploration of a KB. They are similar to the ones used in SPORTAL and in IndeGx. LODAtlas [22] is a website[2] aiming to keep available the metadata from the catalog datahub.io in its previous version, as it was in July 2018, and to offer further visualisation of the content of the dump files of KBs.

2.5. Query federation

The statistics obtained by IndeGx are used in an experimental federated query engine[3] implemented with Corese [7]. It first queries the SPARQL endpoint containing the metadata generated by IndeGx to carry source selection, it rewrites the user query accordingly, and returns the results.

HiBiSCus [24] also uses an RDF index for query federation. The metadata stored in the index represents the so-called "capabilities" of each KB, i.e. the description of the predicates occurring in the KB and the authority of the URIs that appear as subjects and objects of the predicates. The process to generate the index is not described. It is used in a pruning algorithm to compute the minimal set of KBs that must be queried to get the results of a federated query. Harth et al. [13] propose to use a Q-Tree [16] for the source selection for federated queries. A Q-Tree is an index data structure adapted for multidimensional sparse data. Catania et al. [4] go further by enriching Q-Trees with quality and context metadata. Those summaries are optimized for query federation, but unlike IndeGx they do not give other information such as provenance or human-readable information.

2.6. Comparison of related approaches

In this section we compare the closest approaches in the state of the art to our own. We defined a series of various features and we evaluated the approaches based on the articles describing them. Table 1, presents a synthesis of our analysis.

2.6.1. Target users and accessibility of data

Approaches targeting human users present information about the content of their KBs through a web site. All the exploration approaches provide a website used as UI. Approaches targeting software agents generally provide access

[1]Available at `https://onto.fel.cvut.cz/dataset-dashboard` (Last access: 28/06/2022)

[2]Available at `http://purl.org/lodatlas` (Last access: 28/06/2022)

[3]Available at `http://corese.inria.fr/federate/sparql`.

	Indexing		Monitoring			Exploration			Federation		
	SPORTAL [15]	LOD Laundromat [1]	SpEnD [31]	SPARQLES [25]	Yummy Data [30]	Dataset Dashboard [18]	SPARKLis [11]	LOUPE [20]	HiBiSCus [24]	Catania et al. [4]	IndeGx
Domain	General	General	General	General	Biomedical	General	General	General	General	General	General
Target user:											
- Software agent	✓	✓			✓				✓	✓	✓
- Human agent	✓		✓	✓	✓	✓	✓	✓			
Accessibility of data:											
- RDF dump		✓									
- Website	✓	✓	✓	✓	✓	✓	✓	✓			✓
- SPARQL endpoint	✓	✓									✓
Uses only standard SW techno:	✓					✓	✓	✓			✓
Traceability of process:	✓			✓	✓			✓			✓
Quality indicators:	✓	✓	✓	✓	✓				*	✓	✓
KB content representation:											
- Class/properties pop.	✓	✓	✓		✓	✓	✓	✓		✓	✓
- Data summary									✓	✓	
Nb of KB indexed:	557		658	307	67	204	N/A	35	N/A	N/A	339
Linked Data		✓									✓
Update frequency	Bi-weekly	N/A	N/A	Hourly*	Daily	N/A	N/A	N/A	N/A	N/A	Bi-weekly
Active:				✓	✓	✓	✓	✓			✓
Last update:				2022-10-03	2022-10-03	2022-10-03	2022-10-03	2016-07-01			2022-10-03

Table 1: Comparison of the features of related approaches. Checked cells indicate the presence of a feature in the approach. Asterisks indicate the presence of a feature close to the one described. N/A indicates that the feature is not applicable for this approach.

to their metadata through a SPARQL endpoint or RDF dumps. HiBiSCus extends an existing federation engine and offers no RDF dump, website, or SPARQL endpoint. A few approaches target both human or software agents. IndeGx gives access to the metadata it generates during its periodic indexing through a SPARQL endpoint. Additionally, the KartoGraphi [19] website[4] exploits IndeGx to offer various visualizations of the generated metadata, including evolution of this metadata through time.

2.6.2. Only standard semantic web technologies

Approaches that rely only on standard semantic web technologies are technologies that use SPARQL queries to interact with KBs and that store their results in RDF. This makes them easily integrable into existing semantic web ecosystems. This is the case of all the exploration tools listed in Table 1 (SPARQKLIS, LOUPE and Dataset-Dashboard), that use SPARQL queries to generate the content displayed on their website. It is also the case of SPORTAL and IndeGx. HiBiSCus uses SPARQL ASK queries to identify relevant sources during query federation but does not precise the method used to retrieve the content of its index. SPARQLES and Yummy Data both make extensive use of SPARQL queries in their monitoring but they also use other technologies to extract data from endpoints.

2.6.3. Traceability

For the tools evaluated in Table 1 *traceability* means the possibility to retrace the data generation process. Like SPORTAL and Yummy Data, IndeGx records reports for each query sent to an endpoint as part of the metadata of the index they generate. SPARQLES records a report of the SPARQL features supported by endpoints, associated to the query used to evaluate the feature. LOUPE describes the queries used to generate statistics on its website.

2.6.4. Quality indicators

The notion of *quality* of a KB is relative to its intended usage. The approaches listed in Table 1 test KBs according to quality indicators adapted to their usages. For example, the availability of a KB is a general indicator used in all indexing and monitoring systems. Monitoring approaches like Yummy Data or SPARQLES are specialized in evaluating KBs. Yummy Data computes a quality score for a KB combining several features, among which the average response time of an endpoint, the usage of classes and properties in the graphs of the KB, and the formatting of the URIs. SPARQLES computes the response time of SPARQL endpoints in different situations and the list of the SPARQL features that they support through time. IndeGx implements the tests used in SPARQLES and some of the features used by Yummy data, among which the presence of provenance data or the format of URIs as "cool URIs". Like Yummy Data it also computes the list of the namespaces of the properties and classes appearing in a KB to represent the vocabularies used by a KB. This may give an indication about the theme of a KB and help the user write her first queries if she does not know the KB's model. SPORTAL and HiBiSCus [24] adopt a similar but more limited approach. They only extract the authority of the resources of a KB, i.e. the domain of the URL. HiBiSCus shows that using the authority of URIs is effective to determine the necessary sources for a federated query.

2.6.5. KB content representation

Almost all approaches for the indexing, monitoring and exploration of KBs compute a representation of the population of each class and property of a KB. In most cases, including IndeGx, this knowledge is represented using the VoID vocabulary. For instance, SPORTAL uses this representation of the content of a KB as input to its source selection algorithm.

Federation approaches use other representations in their algorithms. For instance, HiBiSCus uses a summary of the authorities used in the domains and ranges of the properties in a KB. IndeGx generates statistics on the usage of the namespaces of entities in a KB, in a manner similar to the property and class partitions defined in the VoID vocabulary.

2.6.6. Update frequency

A lot of KBs are regularly updated and evolve with their domain of knowledge. To keep with this evolution, indexing and monitoring systems must regularly update their content. Of course this is not an issue for the exploration systems since they all query KBs when exploring them. IndeGx, like SPORTAL, updates its content on a bi-weekly basis to avoid surcharging the endpoints it indexes with its complex queries.

2.6.7. Activity

The approaches listed in Table 1 are considered as active if their source code or website is accessible. As it can be seen in the table, half of them are not active any more.

2.6.8. Synthesis

Our review of the approaches related to IndeGx shows that there is a growing interest for the extraction of metadata from KBs for various usages. We designed IndeGx as a transparent and extensible declarative tool to extract metadata based on SPARQL queries, among which all the metadata extracted by SPORTAL, SPARQLES, and LOUPE. It also provides original metadata, among which several statistics on the usage of the namespaces in resource URIs and literal datatypes, the extraction of the timezone of the server and a few SPARQL features not covered by SPARQLES.

By the reproduction, the combination and the extension of previous initiatives, the capacity of IndeGx to smoothly

[4]Available at `http://prod-dekalog.inria.fr`.

integrate new features makes it suitable for different uses: indexing, monitoring, and exploration. This ability to integrate new features and the transparency of its generation process are key features that distinguish IndeGx from other related approaches.

3. Dataset Metadata Model

As we highlighted it in Section 2, KB metadata are used for different purposes: presentation of a human-readable description in a catalog, for instance to foster adoption and trust, query federation, or query rewriting. Our method proposes to generate metadata of a KB to support different usages for both humans and machines, by collecting, computing and publishing this metadata as a KB itself, following the Linked Data principles. We distinguish between four types of metadata:

Definition 3.1 (Asserted metadata). *We define* ***asserted metadata*** *as the assertions about a KB or its content, typically made by their authors or by other human agents.*

Example asserted metadata are the authors, the license, or a human-readable description of the KB. Usually, they cannot be automatically generated from the KB. Those metadata are necessary to describe a KB in a catalog as they constitute the core of its catalog entry. They can also be useful for result ranking in query federation systems that annotate results with their sources, such as Corese [5] and BioFed [14], to apply trust-based policies [23] or to privilege the freshness of a source.

Definition 3.2 (Computable descriptive metadata). *We define* ***computable descriptive metadata*** *as the assertions computable from the content of a KB and describing this content.*

Example computable descriptive metadata are the number of triples or the list of named graphs in a KB, the numbers of classes, of properties, and their distributions. This type of metadata can easily be extracted from any KB with an active endpoint. Many applications rely on it for query federation [15, 24], KB exploration [11, 18, 20], or KB monitoring [25, 30, 31].

Definition 3.3 (Computable quality metadata). *We define* ***computable quality metadata*** *as the metadata that measures or provides indications on the quality of a KB.*

Computable quality metadata originate from the best practices in the semantic web community. Example of such metadata are the presence of license and provenance information, or the readability of a KB for humans, evaluated by the proportion of resources with a human-readable label in the KB. Catalogs like Yummy Data can use computable quality metadata to rank KBs. Monitoring application can be interested in the evolution of quality indicators over time. For instance SPARQLES provides the evolution of the availability of each endpoint it monitors.

Definition 3.4 (Generation traces). *We define* ***generation traces*** *as the description of the interactions with a KB in order to retrieve or generate the three other types of metadata on this KB.*

Generation traces contain the response to each SPARQL query sent to the endpoint of a KB with the date and time of its start and end of execution, as well as the error message returned by the execution, if any. For example, when an availability test fails, its trace will generally indicate that it failed due to a “timeout” error. Generation traces ensure the traceability of the values generated by IndeGx and are a potential source of technical metadata about the endpoint of a KB.

The rationale behind the categorisation we propose for the KB metadata is to distinguish (1) between asserted and computable KB metadata, and (2) between metadata describing the KB content and metadata providing insights into the quality of a KB and its content. Additionally, generation traces are (meta)metadata on the generation of the metadata of the three other types.

By definition, asserted metadata, computable descriptive metadata, and computable quality metadata cover all the metadata that can be expressed using standard vocabularies for KG descriptions (see Section 2.1) and retrieved or computed using SPARQL queries, thus including most of the metadata on which the state of the art approaches are based on (see Sections 2.2 to 2.5), notably, SPORTAL [15] and HiBiscUs [24] for query federation, LOUPE [20] and [22] for KB exploration, SPARQLES [25] and Yummy Data [30] for KB monitoring.

In the rest of this section, we detail each category of metadata. Our process to retrieve or generate them is described in Section 4. To simplify the examples in the rest of the article, we will use the prefixes defined in Listing 1.

Listing 1: Namespace declarations used in our listings.

```
bnb:    http://bnb.data.bl.uk/
dbo:    http://dbpedia.org/ontology/
dbp:    http://dbpedia.org/
dbps:   http://dbpedia.org/schema/
dbpr:   http://dbpedia.org/resource/
dbv:    http://dbpedia.org/void/
dcat:   http://www.w3.org/ns/dcat#
dct:    http://purl.org/dc/terms/
dqv:    http://www.w3.org/ns/dqv#
earl:   http://www.w3.org/ns/earl#
ex:     http://e.g/
formats: http://www.w3.org/ns/formats/
kgi:    http://ns.inria.fr/kg/index#
mf:     http://www.w3.org/2001/sw/DataAccess/
                tests/test-manifest#
owl:    http://www.w3.org/2002/07/owl#
prov:   http://www.w3.org/ns/prov#
rdf:    http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:   http://www.w3.org/2000/01/rdf-schema#
sd:     http://www.w3.org/ns/sparql-
               service-description#
```

```
void:   http://rdfs.org/ns/void#
xsd:    http://www.w3.org/2001/XMLSchema#
```

3.1. Asserted Metadata

In general, metadata about a KB can be obtained either from the KB itself or from an existing catalog. Only the metadata embedded in the KB is guaranteed to be machine-readable and written by the author of the KB. It is also the one most likely to be updated whenever changes are made to the KB. Still, embedding a KB description within the KB itself is not a widespread practice. As of the writing of this article, only 14% of the 564 endpoints monitored by SPARQLES include a VoID description [25].

We defined the *asserted metadata* as the metadata that we usually cannot derive from the content of the KB or by interacting with it. It can be declared in the KB itself or in an external source. It generally contains a description of the dataset including its provenance, authors, date of creation, license, etc. This kind of description is often used to create a human-readable entry in a catalog and to support keywords search for datasets.

There are many ways to structure this description and many vocabularies can be used. We propose to centralize and format this description using well-known vocabularies. We expect that a detailed description of a KB uses VoID and DCAT. Other vocabularies such as Dublin Core, Schema.org[5], or PROV-O can be used to complement the description of a KB. Listing 2 gives a minimalistic dataset description. Parts of the dataset description of DBpedia and the British National Bibliography KBs are shown in Listings 6, 7, 8 and 10.

Listing 2: Minimal dataset description.

```
ex:dataset a void:Dataset, dcat:Dataset ;
  rdfs:label "Example dataset"@en ;
  void:sparqlEndpoint ex:sparql .
```

3.2. Computable Descriptive Metadata

A KB description may contain elements that are derived from its content. Those elements differ in their nature from the asserted metadata as they are the part of the KB description anyone can compute. This category of metadata is more likely to change with the updates of the content of the KB. In the following we propose a categorization of computable descriptive metadata, that should be represented using the VoID and SPARQL-SD vocabularies.

3.2.1. Endpoint description

The documentation on the features of an endpoint guides the interactions with a KB. In practice, the SPARQL-SD vocabulary states that a SPARQL service description must contain a resource that is the subject of a property `sd:endpoint`, with the endpoint URL as the object of the property. Following this recommendation, we can infer that this resource identifies the SPARQL service, of type `sd:Service`, associated to the endpoint. Listing 3 gives an example of DBpedia's service description refering to DBpedia's dataset `dbv:Dataset` and DBpedia's SPARQL endpoint `dbp:sparql`.

Listing 3: Minimal endpoint description.

```
dbp:endpoint a sd:Service, dcat:DataService ;
  sd:endpoint dbp:sparql ;
  dcat:servesDataset dbv:Dataset .
```

Version 1.1 of SPARQL introduced several useful and powerful features for advanced querying, and SPARQL engine implementations have progressively supported such important features. Yet, SPARQLES shows that among the 500 endpoints it surveys, while 99% are compliant with some SPARQL 1.1 features, only 27% are compliant with all of them. Other practical aspects of SPARQL querying such as the query results format used by the endpoint also vary from one engine to the other. These are obstacles to the creation of generic Semantic Web applications, and a description of the SPARQL engine features should appear in the KB description to inform users about their specificity. As an example, as of July 2022, the description of DBpedia's endpoint contains the triples in Listing 4. They describe the supported versions of SPARQL, the supported input and output formats, as well as the non-standard property `bif:contains` that is characteristic of Openlink Virtuoso implementations for optimized full text search.

Listing 4: Excerpt from the description of DBpedia's endpoint on its supported query languages and query result formats.

```
dbp:endpoint
  sd:supportedLanguage
    sd:SPARQL11Update, sd:SPARQL11Query,
    sd:SPARQL10Query ;
  sd:resultFormat
    formats:N-triples, formats:RDF_XML,
    formats:RDFa, formats:N3, formats:Turtle;
  sd:inputFormat
    formats:N3, formats:RDFa ,
    formats:RDF_XML, formats:N-triples ;
  sd:propertyFeature bif:contains .
```

Beyond the default graph of a dataset, SPARQL allows querying specific named graphs identifying subsets of data. RDF named graphs have many potential usages: reification, provenance, subdivision of content, access control, etc. As they change the way we should interact with the KB, the list of the named graphs of a KB and their usage policy should appear in the description of a KB endpoint. Listing 5 provides the example description of the SPARQL engine features of DBpedia's endpoint: `sd:RequiresDataset` indicates that the endpoint requires the usage of a `FROM` clause to specify a graph; `sd:UnionDefaultGraph` indicates that querying the default graph returns the same results as querying the union (the merge) of all the named graphs (thus allowing the user to query any graph without specifying it in their

[5] `https://schema.org/`

query); `sd:BasicFederatedQuery` indicates that DBpedia's endpoint supports the SERVICE keyword for basic query federation (but in practice it rejects queries with a SERVICE clause); `sd:EmptyGraphs` indicates that graphs can be empty and are not automatically removed.

Listing 5: Excerpt from the description of DBpedia's endpoint on its SPARQL engine features.

```
dbp:endpoint sd:feature sd:RequiresDataset ,
    sd:UnionDefaultGraph ,
    sd:BasicFederatedQuery ,
    sd:EmptyGraphs ;
```

3.2.2. Dataset statistics

An overview of the content of a KB typically includes the list or hierarchy of its classes, properties, and their statistics. Also the count of triples, classes, and instances is a common measure of the size of a KB. For instance, these are description elements provided by SPARKLIS [11] or LOUPE [20]. Listing 6 is an excerpt from the description of DBpedia's dataset including such statistics[6]:

Listing 6: Excerpt from the description of DBpedia's dataset on basic statistics.

```
dbv:Dataset rdf:type void:Dataset ;
  void:triples 1099079630 ;
  void:classes 483628 ;
  void:properties 54422 ;
  void:entities 39830270 .
```

The list of classes and properties instantiated at least once are interesting complementary information since different KBs pertaining to similar domains can have different coverage of the same ontology. This information is hardly readable by humans for large ontologies, e.g. more than 483,628 classes in DBpedia but it can easily be processed by machines. As an example, SPORTAL uses such dataset statistics expressed with the VoID and VoIDext vocabularies to optimize query federation. Listing 7 provides an excerpt from the description of DBpedia's dataset on class statistics that we computed.

Listing 7: Excerpt from the description of DBpedia's dataset on class statistics.

```
dbv:Dataset
  void:classPartition
    [ void:entities 88646 ;
      void:class dbo:Agent ] ,
    [ void:entities 64036 ;
      void:class dbo:Person ] ,
    [ void:entities 62944 ;
      void:class dbo:Place ] .
```

3.2.3. Interlinking

The existence of interconnections between KBs is the key to the Web of Linked Data. KBs are either interconnected by their common vocabularies, by common resources, or by explicit equivalence relations between resources. In addition to the dataset statistics, the list of vocabularies used has been shown by both [21] and [24] to be a powerful tool to optimize source selection for federated queries.

Mapping relations between different resources are commonly expressed in the content of a KB, e.g. using the `owl:sameAs` property. The VoID vocabulary provides a dedicated structure, called linkset, to describe a set of `owl:sameAs` relations. Listing 8 shows one such linkset asserted by DBpedia.

Listing 8: Excerpt from the description of DBpedia's dataset on its `owl:sameAs` linkset.

```
dbv:sameAsLinks a void:Linkset ;
  void:inDataset dbv:Dataset ;
  void:linkPredicate owl:sameAs ;
  void:triples 49127465 .
```

3.2.4. SPARQL Compatibility

The computation of the above described features rely on SPARQL 1.1. Nevertheless, depending on the size of the base and the SPARQL coverage of the SPARQL engine, the generation of the dataset statistics may be refused by the endpoint. SPARQL engines generally implement only a subset of SPARQL 1.1 features and the possibility of using SPARQL 1.1 to query a KB is important both for the support of our operations and as a piece of information to be mentioned in the endpoint description. SPARQL-SD enables to describe specific SPARQL features. It also specifies the resources `sd:SPARQL10Query`, `sd:SPARQL11Update` and `sd:SPARQL11Query` to describe the SPARQL coverage of the endpoint. An example description is shown in Listing 4.

3.2.5. Default and Named Graphs

As shown in Listing 5 describing DBpedia's features, some endpoints expect from SPARQL queries that the graph that must be queried be indicated. It is thus necessary to first retrieve the list of the available graphs in a KB. Moreover, when used as a subdivision of content, named graphs can be seen as "separate" KBs. SPARQL-SD allows the description of the different graphs present as part of the description of the KB's endpoint. As an example, Listing 9 represents the list of named graphs in DBpedia, and their names in the DBpedia URI namespace.

Listing 9: Excerpt from the description of DBpedia's endpoint listing DBpedia's named graphs.

```
dbp:endpoint sd:namedGraph
  [ a sd:NamedGraph ;
    sd:name dbp:sparql-sd ]
  [ a sd:NamedGraph ;
    sd:name dbp: ]
  [ a sd:NamedGraph ;
    sd:name dbpr:classes# ]
  [ a sd:NamedGraph ;
    sd:name dbps:property_rules# ]
  [ a sd:NamedGraph ;
    sd:name dbv: ]
```

[6]Extracted the 29th of October 2021

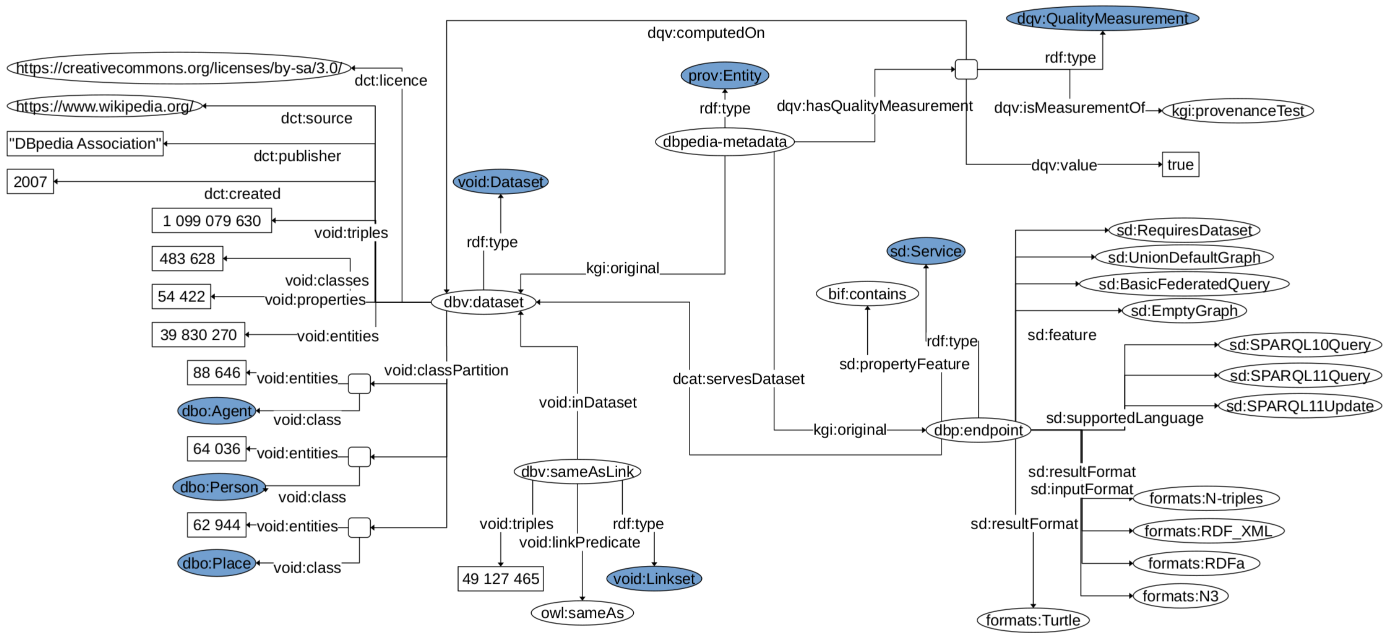

Figure 1: Schema of the part of the description of the dataset of DBpedia as presented in Listings 4, 5, 6, 7, 8 and 9

3.3. Computable Quality Metadata

Computable quality metadata, like computable descriptive metadata, are computed on the content of the KB. They provide metadata quality indicators for both the endpoint and the dataset of the KB. Quality is generally defined as the *fitness to use*, a broad expression that may cover very different indicators. Below we present the set of quality indicators that we consider and represent using the Data Quality Vocabulary. We consider basic quality features such as the availability of a KB's endpoint and the respect of well-known semantic web best practices. We do not aim at providing a comprehensive quality assessment test suite; we rather provide a framework equipped with basic indicators, that users may extend with any additional quality indicators matching their specific requirements.

3.3.1. Availability

As SPARQLES and others have shown over the years, the availability of SPARQL endpoints is variable through time. The availability of the endpoint of a KB is a measure of its quality and reliability. It can be estimated by logging the successful and failed access attempts made at regular intervals by the index updating system. This measure also gives an indication of the likeliness of obtaining complete results in a federated context. In practice, an access attempt can rely on a simple SPARQL query "`ASK {?s ?p ?o.}`".

3.3.2. Compliance with best practices

The semantic web community has defined several best practices for the creation and the distribution of KBs. We propose to check the adherence to some of them. The first one is that entities should be labeled. As some KBs, such as Wikidata [26], use URIs that bear no human-readable semantics – e.g. "Cat" is `wd:Q146` – the labeling of resources is even more necessary to ensure that the data is exploitable by humans. Studies of the quality of semantic web data and of labels in particular, such as [10, 17], have shown that too often this practice is not followed. Besides, several different properties can be used for labeling, e.g. `foaf:name` for persons, `dct:title` for books, `skos:prefLabel` for SKOS concepts, `rdfs:label` or `schema:name` for everything else.

Another best practice is to include provenance information in the the asserted metadata on a KB, like the data source it is derived from. These provenance metadata are the keystone for trusting the KB and share it in accordance with its license. Different standard vocabularies such as the Dublin Core, Schema.org and PROV-O can be used to describe provenance information. As an example, Listing 10 presents provenance metadata on DBpedia using the Dublin Core.

Listing 10: Provenance metadata collected from DBpedia.

```
dbv:Dataset
  dct:created  "2007"^^xsd:date ;
  dct:contributor "Soren Auer",
    "Sebastian Hellmann", "Chris Bizer",
    "Christopher Sahnwaldt", "Jens Lehmann",
    "Robert Isele", "Claus Stadler" ;
  dct:creator "Soren Auer", "Jens Lehmann",
    "Christian Bizer" ;
  dct:license
    <https://creativecommons.org/licenses/
      by-sa/3.0/> ,
    <http://www.gnu.org/licenses/fdl.html>;
  dct:publisher "DBpedia Association" ;
  dct:source <https://www.wikipedia.org/> .
```

Our method consists in looking for expected assertions made with well-known vocabularies in the asserted metadata of a KB (authorship, licence and date of release), and representing an indication of whether or not this information is available as a quality measurement using the Data Quality Vocabulary. Listing 11 presents a quality measurement of DBpedia asserted metadata.

Listing 11: Excerpt from the description of the British National Bibliography's dataset on the existence of provenance metadata.

```
:dbpedia-metadata dqv:hasQualityMeasurement [
  a dqv:QualityMeasurement ;
  rdfs:label "Provenance information is
        present in the metadata"@en ;
  dqv:computedOn dbp:sparql ;
  dqv:isMeasurementOf kgi:provenanceTest ;
  dqv:value "true"^^xsd:boolean
] .
```

There are other well-known possible quality indicators like the consistency of the KB's ontology, or its update frequency, that we do not detail here.

3.4. Generation Traces

In the best case, asserted metadata on a KB include the provenance of the KB's content but very rarely provide the provenance of the metadata itself. We propose to keep track and describe in RDF each step of the generation of a KB description. This allows for the description of the reason for missing values in the metadata, e.g because of a timeout during query evaluation, and it gives a history of the availability of the KB endpoint. We use PROV-O and the Evaluation and report vocabulary (EARL) to describe this process of metadata production. The trace is linked to the KB by property `earl:subject`, to the generation assets declaring the SPARQL queries needed for the generation of the targeted metadata with property `earl:test`, and to the result of the test with property `earl:result`. As an example, Listing 12 presents the description of the report of a failed test of the availability of DBpedia.

Listing 12: Excerpt from the description of DBpedia on the result of a reachability test on it.

```
[
  a earl:Assertion , prov:Activity ;
  earl:subject kgi:DBpediaMetadata ;
  earl:test kgi:availabilityTest ;
  earl:result [
   earl:info "Availability test" ;
   earl:outcome earl:failed ;
   prov:generatedAtTime "11-05-2021T17:58:43"
  ] .
```

These execution traces can also be used to compute some quality metadata on the KB. As an example, the traces of different SPARQL queries can be used to determine the coverage of SPARQL 1.1 by the KB endpoint.

3.5. Structure of the KB Index

Figure 2 depicts the metadata provided by our index for each registered KB. It consists of:

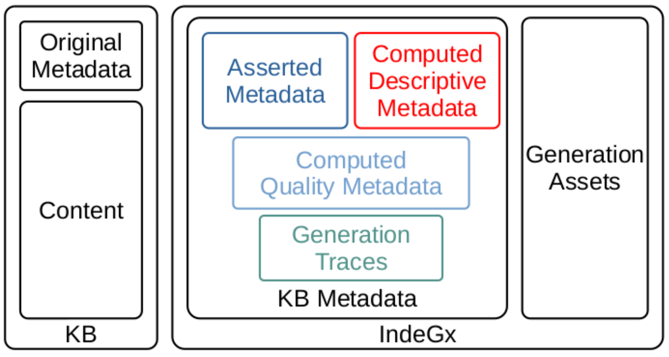

Figure 2: Schema of metadata distribution in our approach.

- the KB metadata extracted or computed by IndeGx, comprising:
  - the extracted asserted metadata,
  - the computed descriptive metadata
  - the computed quality metadata,
  - the generation traces.

  If some computable descriptive or quality metadata is present in the KB as asserted metadata, in IndeGx these assertions are updated with the results of the computations.
- the KB of generation assets describing all the steps for building the index. It is shared by every KB's metadata. They are presented in Section 4.

4. Formalization of Metadata Generation

In this section, we formalize the computation of the metadata of KBs from a catalog of KBs by IndeGx. We call the resources representing the dataset, endpoint, the metadata of a KB the *subjects* of IndeGx. Each entry of IndeGx annotating a subject is called a *subject profile*. Subject profiles are composed of the asserted and computed metadata of a KB's metadata. We present the formalization of the assets used by IndeGx to generate metadata in section 4.1 and the structure of the subject profiles in section 4.2.

4.1. Generation assets: tests and actions

From the computable metadata we identified previously, we detail the ones effectively computed in our approach using the generation assets. We formalize the generation assets as a series of *tests* and *actions* applied to each KB, according to a test process.

We call *tests* the queries that we send to characterize the dataset and the endpoint of a KB. Example tests are checking the availability of an endpoint, or checking if a given function is supported by the endpoint (e.g. supports the keyword `VALUES` to bind a variable to a set of values) or confirming advertised statistics about the base (e.g. the number of triples). We also use a test to check if the dataset of a KB contains the data needed to extract its description and if the endpoint supports the features of the SPARQL language necessary for IndeGx to generate elements of the KB's description.

We call *actions* the queries used to generate the description of a KB. Actions can include the extraction of existing data from the dataset of a KB, the creation of new data according to the result of a test, or the update of an existing description. Example actions are the extraction of the license asserted in the dataset of a KB, or the computation of the number of instances for each class in the dataset.

One test can trigger several actions and the actions triggered by a test can depend on the result of that test. For instance, if an endpoint supports a specific SPARQL feature (e.g. the `SERVICE` clause), several actions based on that feature may be triggered (e.g. attempts to query known endpoints).

In the following sections we explain how tests are designed to characterize a KB and how actions are then triggered to generate metadata in the index using in both cases SPARQL queries. We then specify the ontology and data structures to formalize the resulting index.

4.1.1. Tests: characterizing a KB using SPARQL queries

In the process for building IndeGx, tests are the preliminary interactions with the KBs. The role of a test is to check the functionalities of the endpoint of a KB and the content of its dataset before extracting and computing metadata for this KB in IndeGx.

A test description contains the SPARQL query it is based on. The query is either of the form `ASK` or `SELECT`. If a test consists of an `ASK` query, it is considered successful if the query returns true. If a test consists of a `SELECT` query, it is successful if the endpoint executes the query properly, returns a result – possibly empty – and returns no error. The values retrieved from the KB do not matter for the test itself, it is the actions triggered by the test that will retrieve the values to generate the appropriate metadata. As an example, the very first test launched in our implementation checks the reachability of the KB's endpoint. It consists of a simple `SELECT` query:

```
SELECT * { ?s ?p ?o } LIMIT 1
```

If the KB's endpoint accepts this query, regardless of its results, the action it triggers will initialize the endpoint and dataset description.

A wide range of tests are used to check the quality of computable metadata by comparing the values extracted from the dataset of a KB with the ones computed on the dataset. If needed, tests trigger actions to refine the metadata included in IndeGx. As an example, if statistics on the classes and properties in the dataset are present in the collected metadata, we verify that these values are consistent with the content of the KB, and we correct them in IndeGx if necessary. For instance Listing 6 shows that the current metadata present in DBpedia indicates 1,099,079,630 triples, while the result of the query used to count the number of triples is 859,801,816.

Another type of tests enables to determine the SPARQL 1.1 coverage of the KB endpoint. To this aim, SPARQL queries making increasing use of SPARQL 1.1 feature are sent in sequence to the endpoint. As an example, IndeGx tests the support of the `VALUES` keyword in a `SELECT` SPARQL query using the query:

```
SELECT ?o { <http://nonsensical.com/1> ?p ?o }
VALUES ?o { <http://example.org/thing> }
```

If an endpoint accepts such a query, regardless of its results, the support of this SPARQL 1.1 feature is confirmed and added to the endpoint's description by actions triggered by the test. If the server returns an error, the test is considered to be failed and the feature not supported. For instance the DBpedia dataset states that its SPARQL endpoint accepts basic federated queries (see Listing 5). Yet, in practice, it refuses queries containing the `SERVICE` keyword. Our approach confronts the features indicated in the original metadata against the actual coverage of the SPARQL endpoint.

Occasionally, some tests fail because their execution reaches the limits of the endpoint (e.g. time quota or maximum number of results) or due to the technical specificity of the test and the capabilities of the KB. Our approach makes it possible to define a chain of tests such that a test is launched only if the previous test succeeded or failed. This way, if a test sends a query too complex for the endpoint, alternative methods or less precise tests can be performed. As an example, a common mistake in endpoint descriptions is to use a local address, e.g. starting with `http://localhost`, instead of the actual endpoint URL. At the time of writing this article, DBpedia is in this situation. For this reason, if our approach does not find an endpoint description linked to the endpoint URL, it tests the presence of an endpoint description linked to a local address as a fall-back position. Conversely, if a KB is able to return the results to a test, we then try more complex tests. For instance, if IndeGx can retrieve the list of classes in a dataset, then, it tries to retrieve the list of properties used with class instances as subject or object. If this test is also successful, IndeGx will then compute for each pair of a class and a property the number of triples where an instance of the class occurs as the subject or object of the property.

4.1.2. Actions: generating a description using SPARQL queries

Actions are responsible for the generation of the asserted metadata, computed descriptive metadata and computed quality metadata. Tests can be seen as preliminaries to actions. Indeed, a test will often try a limited version of the actions on the KB, to check if these actions have a chance to succeed and generate a specific kind of information. Thus, actions are generally associated to more complex queries than the tests. The test results in the building process traces determine which action must be applied to build the index. There are three types of actions: *extraction* actions, *generation*, and *modification*. An *extraction action* adds new metadata by copying the metadata that the KB provides about itself to the description resources in the metadata. A *generation action* can create metadata not present in the KB, like statistics on the dataset classes and properties, as part of the computed descriptive metadata. A *modification action* is used to correct the values present in the metadata in the case where the computed descriptive metadata generates values for some KB characteristics already present in the asserted metadata.

An example modification action updates the list of vocabularies used in a KB to reflect the reality of its dataset. For instance, at the time of writing this article, the metadata contained in the WASABI [3] KB on music-related topics indicates 12 vocabularies used in the dataset, whereas the extraction of the namespaces actually used returns only 10 values: the Audio Features Ontology and Wikidata vocabulary are mentioned in the asserted KB description although there are not actually used in the dataset.

The behavior of the tests and actions submitted to a KB can be modified based on the metadata. For instance, if an endpoint description indicates that the endpoint has `sd:RequiresDataset` as a feature, then any SPARQL query should use `FROM` or `FROM NAMED` clauses. In this case, if the endpoint description does not indicate the feature `sd:UnionDefaultGraph`, then the list of named graphs must be added in `FROM` clauses to any query submitted to the KB.

4.2. IndeGx model

The IndeGx ontology has for namespace `http://ns.inria.fr/kg/index#` and its preferred prefix is `kgi:`. It extends the W3C Manifest, the Evaluation and Report Language (EARL), VoID and PROV-O vocabularies to describe tests, actions, and traces. In the following we describe the most important concepts and Table 2 gives a short summary of the ontology.

4.2.1. KB description generated by the IndeGx framework

Each KB description in IndeGx describes three core resources: the dataset subject, the endpoint subject and the metadata subject. It combines the content of their respective *profiles*. The dataset and endpoint subject profiles gather the extracted asserted metadata and the computed (descriptive and quality) metadata, while the metadata subject profile gathers computed quality metadata and generation traces. In IndeGx's current implementation, for the special case of a KB that does not enable IndeGx to actually compute the computable metadata, if some asserted metadata is available, then it is included in the profile in the place of the missing computed metadata. Property `dcat:dataset` links the dataset subject to the root of the index. The dataset and endpoint subjects are linked together with properties from VoID and DCAT vocabularies. The metadata subject is linked to the two other subjects using property `kgi:curated`.

Figure 3 presents the description of DBpedia in IndeGx. The *entry point* of the index is `kgi:catalogRoot`, instance of `dcat:Catalog`. The resources of type `dbv:Dataset` and `dbv:sparql-sd` are extracted from DBpedia (see Listings 4, 5, 7 and 8).

As an additional example, a simple profile, `ex:meta`, linked to an endpoint subject `ex:endpoint` and a dataset subject `ex:dataset`, would be represented as follows:

Listing 13: Example of a simple metadata description resource.

```
ex:meta a prov:Entity ;
  kgi:curated ex:endpoint, ex:dataset ;
  prov:wasAttributedTo ex:sparql ;
  prov:generatedAtTime "2021-07-15"^^xsd:date .
```

The generation traces are linked to the metadata subject with property `kgi:trace`. Each trace is written as an EARL report. Most queries used in tests are sent as they are to the endpoint of a KB. Some queries are templates where placeholders such as the endpoint URL must be replaced. The rewritten query is logged by linking it to the result of the report with property `kgi:sentQuery`. As an example, after a successful test of the presence of asserted metadata in a KB, with a default graph made of three named graphs, the following report is generated:

Listing 14: Example of EARL report.

```
ex:datasetDescResExtract
  a earl:Assertion, prov:Activity ;
  dct:title "Extraction of the
    description resource from the example
    endpoint." ;
  earl:subject ex:endpoint ;
  earl:test kgi:datasetDescResourceExtract ;
  kgi:sentQuery """
  SELECT DISTINCT ?res
    FROM :graph1
    FROM :graph2
    FROM :graph3
    WHERE {  { ?res a dcat:Dataset }
             UNION
             { ?res a void:Dataset }  }""" ;
  earl:result [
     earl:outcome earl:passed ;
     prov:generatedAtTime "2021-07-15"^^xsd:date ;
     earl:info "The server returned an
               answer for this query"@en
    ] .
```

4.2.2. Test and action description

Tests and actions form the *generation assets* used by IndeGx to generate metadata by interacting with a KB. Each test is associated with at least one action. Generation assets are declared and organized following the structure of a test suite using the Manifest vocabulary, that has been extended in the IndeGx ontology (see Table 2). Class `earl:TestCase` is specialized by two new classes:

Resource	**Description**
`kgi:catalogRoot`	Default resource used as the root of the index.
`kgi:federation`	Used as object of `kgi:endpoint` to indicate that a query must be sent to a federation server.

Class	**Description**
`kgi:TestQuery`	Main class for the tests.
`kgi:TestTrue`	A placeholder class used for tests that always succeed, to apply their actions, generally for the maintenance of IndeGx.

Property	**Description**	**Domain**	**Range**
	KB description		
`kgi:endpoint`	Indicates the endpoint to send a query to if it is not the currently described KB.	`mf:ManifestEntry`	`rdfs:Resource`
`kgi:original`	Links the metadata description resource to the original description resources from the described KB.	`prov:Entity`	`prov:Entity`
`kgi:curated`	Links the metadata description resource to the description resources generated by the framework.	`prov:Entity`	`prov:Entity`
	Trace description		
`kgi:trace`	Links the metadata description resource to the traces generated by the operations of the framework.	`prov:Entity`	`earl:Assertion`
`kgi:sentQuery`	Links the result of a test to the query sent if it has been parameterized from its original template during operations.	`earl:TestResult`	`xsd:string`
	Test description		
`kgi:onSuccess`	Links a test to the list of actions to be performed if it succeeds.	`mf:ManifestEntry`	`rdf:List`
`kgi:onFailure`	Links a test to the list of actions to be performed if it failed.	`mf:ManifestEntry`	`rdf:List`

Table 2: Short summary of the IndeGx ontology.

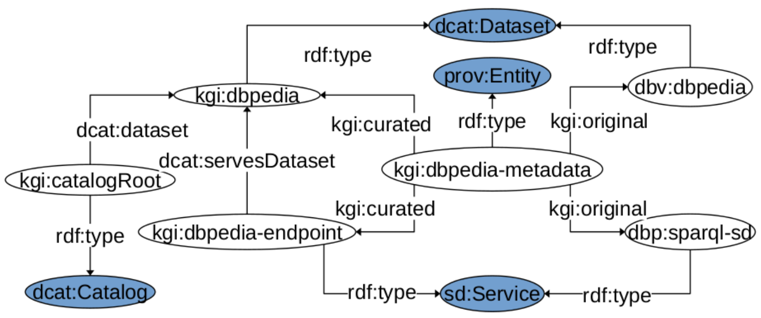

Figure 3: Model of the description of DBpedia in IndeGx

`kgi:TestQuery` is the class of the tests whose success depends on the successful execution of a SPARQL query regardless of its result, e.g. availability tests; this is the case for most of the tests in IndeGx. `kgi:TestTrue` is the class of the tests whose actions are applied automatically for the maintenance of the index, e.g. the update of the date of last modification of the index, or the addition of a set of example queries for IndeGx users. Since the Manifest vocabulary does not contain any property to distinguish the output of an action, we introduced properties `kgi:onSuccess` and `kgi:onFailure` to declare the list of actions to apply when a test succeeds or fails.

A generation asset consists of both an RDF graph describing its test and the RDF description of the actions that must be performed depending on the result of the test in a manifest file. Listing 15 presents the description of an example test used to compute the list of vocabularies occurring in the KB. The associated SPARQL query retrieves the namespaces of the properties and classes.

Listing 15: Example test for the extraction of vocabularies from `vocabularyList.ttl`.

```
<vocabularyList.ttl>
  a kgi:TestQuery ;
  dct:title "Extraction of the namespaces
       of properties and classes."@en ;
  kgi:query """SELECT DISTINCT ?ns WHERE {
   {
     SELECT DISTINCT ?elem {
        { ?s ?elem ?o . }
        UNION { ?s a ?elem . }
        }
    }
    BIND(REPLACE( str(?elem),
        "(#|/)[^#/]*$", "$1")
        AS ?ns) .
  }""" .
```

Listing 16 presents an example manifest containing the description of an action that must be triggered in the case where the test described in Listing 15 is successful: CONSTRUCT queries are sent to the KB to create new triples from the KB's content and SPARQL UPDATE queries are applied to the current content of the index.

Listing 16: Example action for the extraction of list of vocabularies, written in `manifest.ttl`

```
<manifest.ttl> a mf:Manifest ;
  mf:entries ( <vocabularyList.ttl> ) .

<vocabularyList.ttl> a mf:ManifestEntry ;
 kgi:onSuccess (
 [ mf:action """CONSTRUCT {
  kgi:exampleMetadataDescription
   void:vocabulary ?ns .
  }
  WHERE {
    {
    SELECT DISTINCT ?elem {
       { ?s ?elem ?o . }
       UNION { ?s a ?elem . }
    }
    }
    BIND(IRI(REPLACE( str(?elem),
        "(#|/)[^#/]*$", "$1"))
        AS ?ns) .
  }"""
)
```

IndeGx's test suite contains 85 generation assets. There are on average 3.3 actions per successful test and 0.24 actions per failing test.

4.2.3. Off-loading computational load

Some of the generation assets provided in IndeGx rely on an additional, private SPARQL endpoint (in our case an instance of Corese [7]) that we configured to allow any kind of federated query without quotas in terms of number of results or execution time. The usage of an external resource is meant to off-load the CPU- and memory-intensive computations of some generation assets from the KB to the additional endpoint under our control. Using this endpoint also opens up the possibility to use SPARQL extensions such as the LDScript [6] language implemented in Corese.

We have developed an implementation of the approach described in this section, that is publicly available on a Github repository under the GNU General Public License.

5. Experimentation and Evaluation

In this section, we summarize several months of experimentation and evaluation. To present the results we have separated the computable descriptive metadata generation assets into three categories, depending on the origin of the definition of the generation assets used to produce them. The first category corresponds to the list of generation assets that we have defined for IndeGx, such as the extraction of the list of vocabularies used in a source. The second and third categories essentially correspond to, respectively, generation assets adapted from SPARQLES and the ones adapted from SPORTAL. We augmented the last two categories with a few new generation assets but they remain in the same spirit as the others in their categories.

5.1. Experimental Setup

All the experiments were done using a virtual server with 16Go RAM memory and 4 cores Intel(R) Xeon(R) @ 2.60GHz. The framework was launched periodically over a period of 8 months between November the 18th, 2021 and July the 16th, 2022. Over that period, each endpoint and its dataset was subject to at least 72 tests and at most 98 tests. We started our framework with a set of 197 endpoints extracted from the active endpoints listed by SPARQLES, and the lists of endpoints given by LinkedWiki, YummyData, and WikiData. During the experimentation campaigns, we added the endpoints listed on the LOD Cloud website and the list of the chapters of DBpedia. After 6 months, we removed from our lists the endpoints that never answered our queries or whose URL in their source catalog did not point to an active SPARQL endpoint. Starting with 470 KBs extracted from different sources, at the end of our 8 months of experimentation, we counted 339 endpoints available during at least one of our indexations (meaning that 28% never answered), among which 80 were available during all our indexations. To evaluate the coverage of our catalog we performed advanced Google queries leveraging the `allinurl` operator to find SPARQL endpoints[7] and discovered only 5 additional endpoints. The complete catalog of endpoints is available in our GitHub repository.

5.2. Execution Times

In our current implementation, we sequentially evaluate each endpoint of the catalog. Doing so, we obtain the run-time of the evaluation of each endpoint that can then be queried from the generation traces. Over the period of our experiment, IndeGx performed 2947 indexations of datasets. On average each endpoint was indexed 4 times over a period of 8 months. It takes around 13 minutes on average to index a KB with a standard deviation of 7 minutes. Occasionally, we observed that the indexation of all the KB of a catalog took more than 5 days to complete. Therefore, to avoid the launch of an indexation on a catalog while another indexation of the same catalog is still running we empirically chose to update the index every two weeks.

[7]`https://www.google.com/search?q=allinurl%3Asparql`

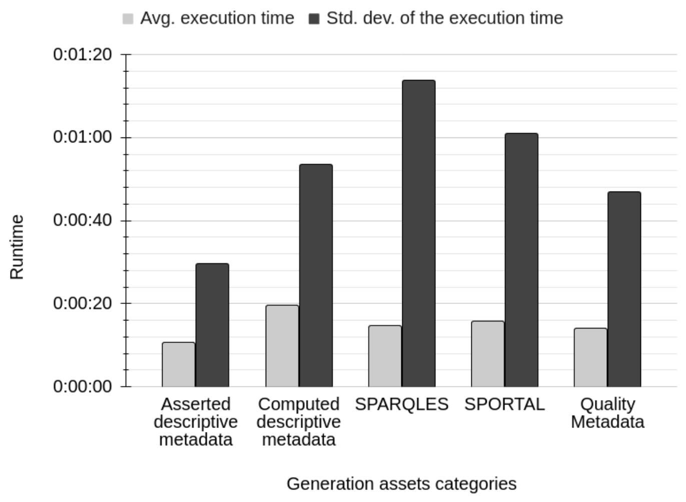

Figure 4: Average runtime of the tests in each category.

Figure 4 shows the distribution of the runtimes against each category of test. It shows that at least 75% of the tests are performed in less than a minute. The standard deviation of the execution shows the differences between the categories. The 4 computed categories have the longest execution time due to the computational complexity of their queries. The asserted metadata and computed quality metadata categories have similar runtimes, as they mainly check the presence of elements in the KB. On average, the five tests with the longest average execution time are:

1. The count of the number of properties;
2. The count of the number of classes;
3. The check for the presence of provenance information;
4. The count of the number of instances of each class;
5. The extraction of the list of vocabularies.

The time taken by the first, second and fourth tests of this list is due to the usage of counting in the query, which is computationally costly. IndeGx checks the presence of provenance information (third test) using a query that contains 10 UNION operations in order to check several well-known combinations of properties for provenance. The extraction of the list of vocabularies (fifth test), as IndeGx does it, uses a REGEX to get the beginning of the URIs of every class or property used in a dataset. The cost of this operation explains the position of this test in the list. Some SPARQL engines, such as Virtuoso, use a runtime evaluation system to filter the incoming queries that are too complex. In practice, these five tests with the longest average execution time are not considered as too complex by these engines, despite their observed response times.

5.3. Metadata Obtained

During the experiment period, IndeGx has generated a total of 3084 descriptions of KB. On average, the description of a KB contains 13,193 triples. Half of the KBs, in the interquartile range of all the descriptions, contain between 2733 and 23,693 triples. In this section, we describe what IndeGx was able to retrieve in terms of provenance information, classes and properties statistics, information about the SPARQL norm coverage and quality feature evaluations.

5.3.1. Provenance metadata

IndeGx looks for provenance information that are stored in datasets. As IndeGx is limited to the metadata accessible using standard SPARQL queries, it cannot retrieve other sources such as the `.well-known/void` files. However, an attempt to retrieve the `.well-known/void` files of the KBs we index showed that only 10 KBs offer such metadata. The test used to look for provenance information searches for an instance of `void:Dataset` or `dcat:Dataset` linked by any appropriate property from a well-established vocabulary, i.e. DCTerms and PROV-O, to any authorship, time-related, licensing or sourcing information. Among the 339 KBs, 33 contained some kind of provenance information. Of those 33, all contained information about the source of their data, 26 KBs contained information about their creators, contributors, or publishers, 18 KBs contained information about the time they were created or released, 16 KB contained information about their licenses.

The proportion of KB with provenance metadata suitable for a KB catalog is very small. Moreover, this lack of information hinders the traceability of data when datasets are reused in external projects. It is possible that our limitations in technology and our choice of vocabulary in the queries made us miss properly described datasets. Yet, from our experience and after numerous manual explorations of KBs, we think that the proportion of KBs annotated with provenance information that IndeGx has identified is representative of the state of the publicly accessible KBs.

5.3.2. Classes and properties statistics

SPORTAL and other approaches have demonstrated the feasibility of a federated query answering method based on the statistics about the classes and properties that can be expressed using the VoID vocabulary. IndeGx uses its system of chained generation assets to incrementally extract as many statistics from each KB as possible. Among the possible statistics, the description of the usage of properties is of particular interest to determine the possible joins between KBs. Among the 339 KBs, the computed descriptive metadata of 207 KBs contains the number of distinct objects and subjects for each property, and the computed descriptive metadata of 184 KBs contains the list of properties appearing around the instances of each class. Those two statistics alone are the base of federation and exploration approaches. Yet, the computation of those statistics is hindered by technical elements, such as the processing power of the endpoint and, sometimes, hidden parameters of an endpoint that limit the number of results retrievable by sub-queries. As an example, by default, an instance of Virtuoso limits the number of results of a query to 10,000, even when an explicit higher limit is specified in the query.

5.3.3. SPARQL features coverage

IndeGx uses the queries used by SPARQLES, with 2 additions, to test 43 SPARQL features. Each query is written not to retrieve results but to test if it receives an error due to the presence of the tested feature. On average, the endpoints support 85% of the tested features. The five least supported features are, in descending order:

1. The usage of the `SERVICE` keyword for query federation,
2. The usage of the `VALUES` keyword,
3. The usage of a sub-query with the `GRAPH` keyword,
4. The usage of the `FROM` keyword,
5. The usage of the filter `NOT IN` in an `ASK` query.

Note that most of the features from SPARQL 1.0 are supported by all endpoints, and most of the features that are not supported are specific to SPARQL 1.1. This can impact all potential use-cases of a KB.

5.3.4. Quality features

IndeGx uses different quality features to evaluate a KB. Each quality feature corresponds to a readability criteria for the content of the dataset or a usability criteria of the KB in different use cases. The measurement are presented either as a Boolean, a sort of flag indicating the presence of a feature in the KB, or as a percentage indicating the proportion of the KBs with the feature. There is no judgment associated with a value of 0 or 1, as a feature can be more or less desirable depending on its usage.

Readable labels. The presence of a label in natural language to describe the entities of a dataset is an important part of the readability of a KB. This feature is highlighted in the survey by Zaveri et al [32]. IndeGx measures the proportion of resources with a label in a dataset. This measure is done in one query, using two sub-queries to count the number of resources in the dataset and the number of labels linked to those resources. Due to this complex structure, the query is not answered by all endpoints. Among the 339 endpoints, 99 have answered IndeGx' test. Of those, 35 have a measure above 0. If there are any labels in a dataset, on average 53% (±38%) of the resources are labeled. This shows that the labelling of every resource is not a widespread practice, which indicates poor readability of the publicly available KBs in general.

Short URIs. In IndeGx, short URIs are URIs without parameters, with less than 80 characters. The shortness of URIs is linked to the readability of the dataset for humans. This is related to the concept of "Cool URIs". IndeGx measures the proportion of URIs in a dataset that are considered as short URIs in one query that assesses all URIs in the dataset. As the query applies different filters on every URI before counting them, it is not answered by all endpoints. Among the 339 endpoints, only 58 answered IndeGx's test. Of those, 89% (±21%) are short URIs.

Blank node usage. Blank nodes are used differently in a KB, depending on its intended usage. In KBs containing schemata, blank nodes are used for the definition of class restrictions and other data structures, as well as in KBs containing SHACL shapes. Conversely, in some KBs containing resource descriptions, blank nodes may have been used where URIs would have been preferable to facilitate their processing. As a result, the presence of blank nodes in a KB is an important indicator which interpretation depends on the usage scenario of the KB. Among the 339 endpoints, 102 endpoints answered IndeGx's generation assets evaluating the proportion of resources that are blank nodes in a dataset. The distribution of the measure in this group shows that, on average, 84% (±22%) of the entities in a KB are URIs, and in 75% of the KBs at least 69% of the entities are URIs.

Language tags. The usage of language tags on literal values improves the readability of a KB and allows the creation of multilingual applications exploiting them. IndeGx measures the proportion of literals with a language tag. As part of the computed descriptive metadata, it also extracts the list of the language tags used in the literals of a dataset. Among the 69 KBs using language tags, the five most used language tags are, in descending order, "en" in 64 KBs, "fr" in 24 KBs, "de" in 19 KBs, "es" in 14 KBs, and "it" in 11 KBs. Note that, due to technical constraints, we do not count the regional variations of some tags, such as en-US or en-UK in this count. Apart from 3 KBs that contain more than 100 language tags, most of the KBs use between 1 and 3 language tags. The measure of the proportion of literals with a language tag in a dataset shows that in the 73 KBs that answered this test, on average 25% (±36%) of the literals have a language tag. Those statistics show that multilingual applications should not expect to find translations for string literals in most publicly available KBs.

Presence of the KB's URI pattern. The VoID vocabulary offers two properties, void:uriSpace and void:uriRegexPattern, to state how the URIs of a KB are constructed. This information can be used in different use-cases, in catalogs entries and in query federation for source selection. A link detection algorithm may also use it to identify the resources coming from another source that are in a KB, and the resources coming from a KB that appear elsewhere. IndeGx evaluates the presence of one of the two given properties in a dataset description. This simple test has been answered by all the 339 endpoints. Only 48 KBs contain the triples describing the URIs of their resources using VoID.

Vocabulary list. The list of vocabularies used in a KB is useful for catalogs and federated query solving approaches. IndeGx computes this list for each KB. Among the 339 endpoints, only 3 of them answered that their KB contains such a list of used vocabularies as metadata. In contrast, IndeGx managed to compute this list of vocabularies for 225 KBs as part of the computed descriptive metadata. Yet, IndeGx computes it by extracting the domain and path of each class or property's URI. While this guarantees to get all the vocabulary namespaces, all the URIs obtained by IndeGx are not guaranteed to be vocabularies.

5.4. Errors Obtained During the Building Process

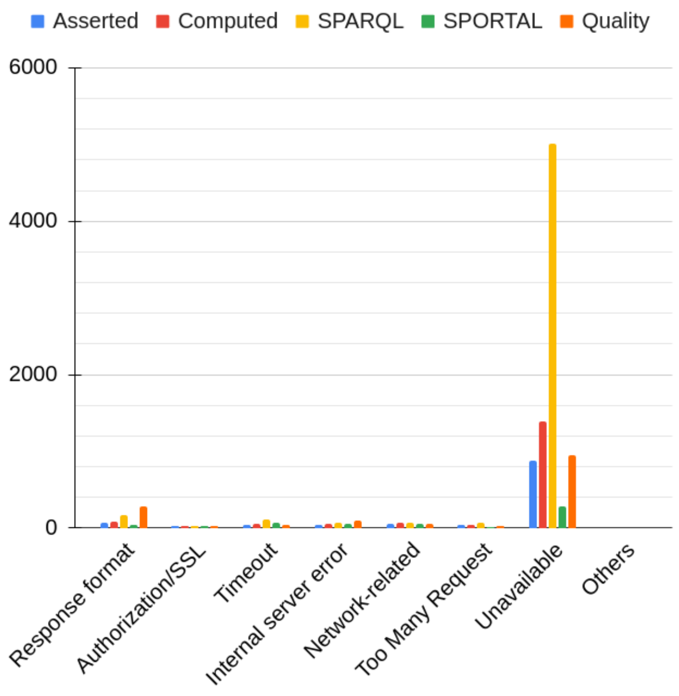

Figure 5: Total number of failing tests in each category, for each type of error during the experiment.

Figure 5 shows the number of tests in each category that failed because of an error during the experimentation period. We have classified the different error traces according to 8 categories corresponding to the message kept by IndeGx in the trace of each test. From the figure we can see that the most common source of failure for the different tests is the unavailability of the endpoints at the moment of querying them. These fluctuations of availability are also a quality feature that is part of the description of a KB. This shows that any application based on a catalog of KB should take into account the probable unavailability of part of their catalog.

5.5. Generation Assets Usage

In this section, we distinguish between explicit computable descriptive metadata and non-explicit computable descriptive metadata. The explicit computable descriptive metadata are computed using straightforward queries whose results are either stored in the dataset or given by the SPARQL engine without consulting the dataset. They include the extraction of the `sameAs` relations, the list of graphs, and the timezone of the SPARQL endpoint. The non-explicit computable descriptive metadata needs a transformation of the entities from the dataset to be computed. This is generally done using a regular expression or another filter applied to a large number of entities. They include the computation of the list of vocabularies, languages, datatypes, hostnames, and the detection of linksets toward other KBs.

The SPORTAL generation assets generate data structures describing statistics about the content of the dataset. We distinguish between simple and complex generation assets. Simple generation assets each only generates one triple, describing the number of a specific type of entities in the dataset of a KB, e.g., the number of classes and properties. Complex generation assets use more complex queries and generate more complex data structures, mostly statistics about classes, like the number of instances per class or the list of properties used in the description of the instances of each class.

During the experimentation, more than 3.4 million triples were generated. Table 3 presents statistics on all the runs of our experiments combined. It provides the number of generation assets and the proportion of generated triples for each category of generation assets. In Figure 6 the black bars show for each category of generation assets the average proportion of KBs that passed at least one test of that category. The doted line shows the maximum proportion of KBs that passed at least one test of a category in all runs combined. A remarkable fact shown by Table 3 and Figure 6 combined is that 88% of the triples were generated by the complex SPORTAL generation assets applied to about 20% of the KBs. To generate metadata, a generation assets needs from the KB both the support of the SPARQL features it uses and the content and the capacity to compute the results of the queries. The generation assets of the SPARQLES category are the simplest as they only check the support of SPARQL features by an endpoint.

5.6. Supporting Human and Machine Interactions with IndeGx

IndeGx proposes a declarative indexing framework which results are accessible to machines as linked data and through a SPARQL endpoint. To make this output accessible to humans too, we provided, on top of this index service, a visualization Web application called KartoGraphI[8]. It generates an overview of important characteristics and a number of the results presented in this paper can also be visualized and browsed on KartoGraphI [19].

[8] `http://prod-dekalog.inria.fr`

	Generation assets categories						
	Asserted metadata	Computable metadata					Computable quality metadata
		Computable descriptive metadata		SPARQLES	SPORTAL		
Color used in figures	🟦	🟥		🟨	🟩		🟧
Sub-category		Explicit	Non-explicit		Simple	Complex	
Number of available tests	5	3	5	45	14	14	13
% of the total size of all the descriptions	1,22%	5,64%	2,94%	1,54%	0,12%	88,0%	0,53%

Table 3: Statistics about the generation assets grouped by categories.

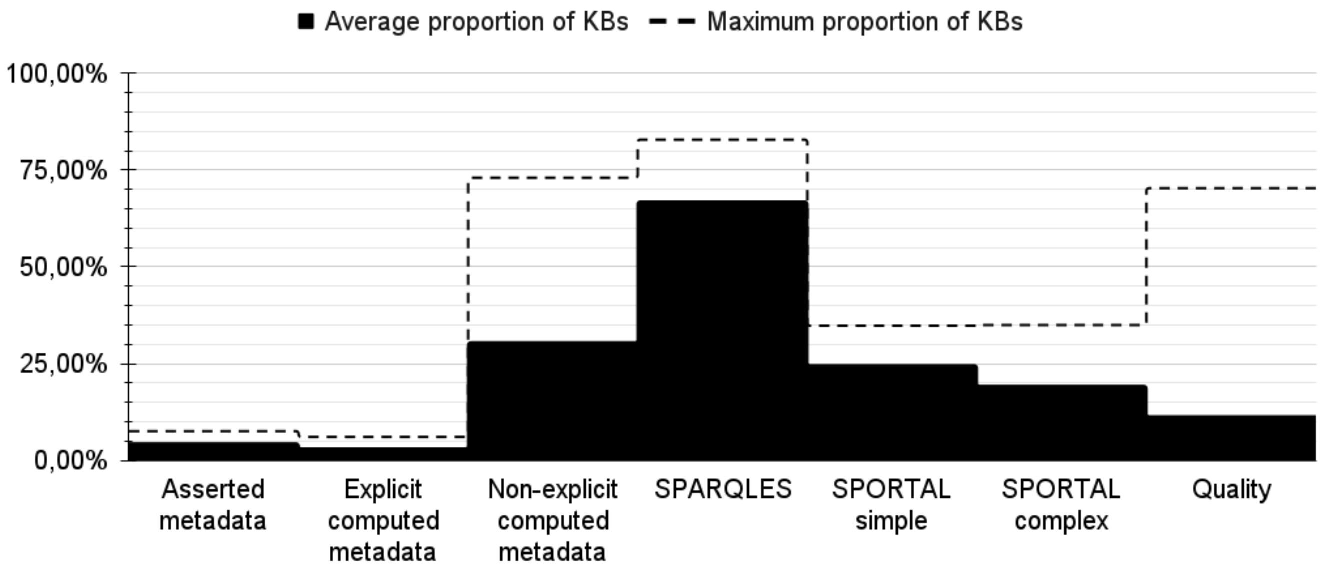

Figure 6: Histogram of the proportion of KBs that passed at least a test of a generation assets in each category, during all runs.

To demonstrate the use of the index in other applications we also provided a demonstrator of a federated query engine exploiting the index. In this prototype, the VoID class and property partitions generated by IndeGx are used by the source selection algorithm of the Corese federated query engine's endpoint to automatically select and distribute the processing of a SPARQL query over several relevant sources.

Finally, to assist and encourage KB providers to include good quality metadata in their KBs, we have created an online service called Metadatamatic, that simplifies the creation of metadata for a dataset through a dynamic form and scripts that reuse the actions defined for IndeGx to extract statistics on the dataset.

6. Conclusion

In this paper, we have presented IndeGx, an approach for the sustainable remote generation of a standard-based description of a KB. Our approach has been implemented in a framework in which every operation of the generation of a description is specified using standard technologies. It can include any description feature of a dataset that can be generated using one or more SPARQL queries. Our implementation of IndeGx relies on a GitHub repository where each indexing feature is collaboratively and declaratively defined. Anyone can add new features to the index by contributing the corresponding actions to the repository. IndeGx is also designed to incrementally assess the endpoints' capabilities in increasing order of complexity, and as much as possible to work around their limits.

IndeGx needed on average 16 minutes to process an available KB, with a standard deviation of 12 minutes. Our experimentation showed that IndeGx can generate metadata from any available KB, the quantity of metadata generated from a KB depends on its dataset's content and its endpoint's capacities. Great disparities were observed in the content and the capacities of the KBs available on the semantic web. This indicates that applications trying to exploit many KBs must be prepared to handle large differences between their sources. In other words, the results of IndeGx show that any linked data consuming application must be conceived as an extremophile agent and that the provided index can help in this regard.

We also showed that IndeGx can support the search for useful KBs and promote the integration of human-readable descriptions in KBs and the upgrade of SPARQL endpoints toward better SPARQL compliance. We demonstrated how visualization portals for humans as well as distribution plans for federated query solving could effectively be built on top of such an index. More generally, IndeGx provides metadata supporting the findability, understanding, and reuse of KBs, thus improving the overall FAIRness of the semantic web.

This first experiment is also opening a lot of perspectives and future works. The current list of KBs used in IndeGx comes from snapshots of catalogs. In the future, we hope to apply a method to automatically discover new KBs, using services such as Google dataset search or using traversal crawling [12] and add them to our index. We will also consider methods to reuse the previous versions of the index as alternate metadata sources to describe a KB. This would also make it possible to survey the evolution of datasets and avoid the loss of data in case of the temporary unavailability of a KB.

We observed that some standard SPARQL queries meant to compute somehow straightforward results cannot be computed by some endpoints, e.g. count the number of triples for each property. Hence, it appears that more subtle strategies will have to be considered in such cases. To extend the capacities of IndeGx, we are investigating the use of SPARQL extensions such as the LDScript language [6], and the use of URI parameters to parameterize SPARQL services according to their capabilities [5]. We are also considering using RDF-Star to augment the traces and the provenance information in the metadata generated by IndeGx.

Acknowledgment

This work is supported by the ANR DeKaloG (Decentralized Knowledge Graphs) project, ANR-19-CE23-0014, and by the ANR D2KAB (Data to Knowledge in Agriculture and Biodiversity) project, ANR-18-CE23-0017, both projects from CE23 - Intelligence artificielle. The work is also supported by the 3IA Côte d'Azur ANR-19-P3IA-0002.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Wouter Beek, Laurens Rietveld, Hamid R Bazoobandi, Jan Wielemaker, and Stefan Schlobach. 2014. LOD laundromat: a uniform way of publishing other people's dirty data. In *International semantic web conference*. Springer, 213–228.

[2] Mohamed Ben Ellefi, Zohra Bellahsene, John G Breslin, Elena Demidova, Stefan Dietze, Julian Szymański, and Konstantin Todorov. 2018. RDF dataset profiling–a survey of features, methods, vocabularies and applications. *Semantic Web* 9, 5 (2018), 677–705.

[3] Michel Buffa, Elena Cabrio, Michael Fell, Fabien Gandon, Alain Giboin, Romain Hennequin, Franck Michel, Johan Pauwels, Guillaume Pellerin, Maroua Tikat, and Marco Winckler. 2021. The WASABI Dataset: Cultural, Lyrics and Audio Analysis Metadata About 2 Million Popular Commercially Released Songs. In *The Semantic Web. ESWC 2021. Lecture Notes in Computer Science, vol 12731.* 515–531. https://doi.org/10.1007/978-3-030-77385-4_31

[4] Barbara Catania, Giovanna Guerrini, and Beyza Yaman. 2019. Exploiting context and quality for linked data source selection. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. 2251–2258.

[5] Olivier Corby, Catherine Faron, Fabien Gandon, Damien Graux, and Franck Michel. 2021. Beyond Classical SERVICE Clause in Federated SPARQL Queries: Leveraging the Full Potential of URI Parameters. In *International Conference on Web Information Systems and Technologies*. Online, Portugal.

[6] Olivier Corby, Catherine Faron-Zucker, and Fabien Gandon. 2017. LDScript: a linked data script language. In *International Semantic Web Conference*. Springer, 208–224.

[7] Olivier Corby and Catherine Faron Faron-Zucker. 2010. The KGRAM Abstract Machine for Knowledge Graph Querying. In *Proceedings of the International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)* (Toronto, Canada). IEEE, 338–341.

[8] Richard Cyganiak, Jun Zhao, Michael Hausenblas, and Keith Alexander. 2011. *Describing Linked Datasets with the VoID Vocabulary.* W3C Note. W3C. https://www.w3.org/TR/void/

[9] Jeremy Debattista, Sören Auer, and Christoph Lange. 2016. Luzzu—a methodology and framework for linked data quality assessment. *Journal of Data and Information Quality (JDIQ)* 8, 1 (2016), 1–32.

[10] Basil Ell, Denny Vrandečić, and Elena Simperl. 2011. Labels in the Web of Data. In *The Semantic Web – ISWC 2011 (Lecture Notes in Computer Science)*. Springer, Berlin, Heidelberg, 162–176. https://doi.org/10.1007/978-3-642-25073-6_11

[11] Sébastien Ferré. 2017. Sparklis: An expressive query builder for SPARQL endpoints with guidance in natural language. *Semantic Web* 8, 3 (Jan. 2017), 405–418. https://doi.org/10.3233/SW-150208

[12] Raphaël Gazzotti and Fabien Gandon. 2021. When owl:sameAs is the Same: Experimenting Online Resolution of Identity with SPARQL queries to Linked Open Data Sources. In *WEBIST 2021 - 17th International Conference on Web Information Systems and Technologies*. Virtual, France. https://hal.archives-ouvertes.fr/hal-03301330

[13] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. 2010. Data summaries for on-demand queries over linked data. 411–420. https://doi.org/10.1145/1772690.1772733

[14] Ali Hasnain, Qaiser Mehmood, Syeda Sana e Zainab, Muhammad Saleem, Claude Warren, Durre Zehra, Stefan Decker, and Dietrich Rebholz-Schuhmann. 2017. Biofed: federated query

processing over life sciences linked open data. *Journal of biomedical semantics* 8, 1 (2017), 1–19.

[15] Ali Hasnain, Qaiser Mehmood, Syeda Sana e Zainab, and Aidan Hogan. 2016. SPORTAL: Profiling the Content of Public SPARQL Endpoints. *International Journal on Semantic Web and Information Systems (IJSWIS)* (July 2016), 134–163. https://doi.org/10.4018/IJSWIS.2016070105

[16] Katja Hose, Daniel Klan, and Kai-Uwe Sattler. 2006. Distributed data summaries for approximate query processing in PDMS. In *2006 10th International Database Engineering and Applications Symposium (IDEAS'06)*. IEEE, 37–44.

[17] Lucie-Aimée Kaffee and Elena Simperl. 2018. The Human Face of the Web of Data: A Cross-sectional Study of Labels. *Procedia Computer Science* 137 (Jan. 2018), 66–77. https://doi.org/10.1016/j.procs.2018.09.007

[18] Petr Kremen, Lama Saeeda, Miroslav Blasko, and Michal Med. 2018. Dataset Dashboard - a SPARQL Endpoint Explorer. In *Proc. of VOILA@ISWC 2018, Monterey (CEUR Workshop Proceedings, Vol. 2187)*. CEUR-WS.org, 70–77.

[19] Pierre Maillot, Olivier Corby, Catherine Faron, Fabien Gandon, and Franck Michel. 2022. KartoGraphI: Drawing a Map of Linked Data. In *Extended Semantic Web Conference*. Springer.

[20] Nandana Mihindukulasooriya, Maria Poveda-Villalón, Raúl García-Castro, and Asunci??n Gómez-Pérez. 2015. Loupe - An online tool for inspecting datasets in the linked data cloud. In *CEUR Workshop Proceedings*, Vol. 1486.

[21] Pascal Molli, Hala Skaf-Molli, and Arnaud Grall. 2020. *SemCat: Source Selection Services for Linked Data*. Research Report. université de Nantes. https://hal.archives-ouvertes.fr/hal-02931367

[22] Emmanuel Pietriga, Hande Gözükan, Caroline Appert, Marie Destandau, Šejla Čebirić, François Goasdoué, and Ioana Manolescu. 2018. Browsing linked data catalogs with LODAtlas. In *International Semantic Web Conference*. Springer, 137–153.

[23] Sam Rahimzadeh Holagh and Keyvan Mohebbi. 2019. A glimpse of Semantic Web trust. *SN Applied Sciences* 1, 12 (2019), 1–10.

[24] Muhammad Saleem and Axel-Cyrille Ngonga Ngomo. 2014. HiBISCuS: Hypergraph-Based Source Selection for SPARQL Endpoint Federation, Vol. 8465. https://doi.org/10.1007/978-3-319-07443-6_13

[25] Pierre-Yves Vandenbussche, Jürgen Umbrich, Luca Matteis, Aidan Hogan, and Carlos Buil-Aranda. 2017. SPARQLES: Monitoring public SPARQL endpoints. *Semantic Web* 8, 6 (Aug. 2017), 1049–1065. https://doi.org/10.3233/SW-170254

[26] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: A Free Collaborative Knowledgebase. *Commun. ACM* 57, 10 (2014). https://doi.org/10.1145/2629489

[27] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data* 3, 1 (2016), 1–9.

[28] Gregory Williams. 2013. *SPARQL 1.1 Service Description*. W3C Recommendation. W3C. https://www.w3.org/TR/2013/REC-sparql11-service-description-20130321/.

[29] Peter Winstanley, David Browning, Riccardo Albertoni, Alejandra Gonzalez Beltran, Simon Cox, and Andrea Perego. 2020. *Data Catalog Vocabulary (DCAT) - Version 2*. W3C Recommendation. W3C. https://www.w3.org/TR/2020/REC-vocab-dcat-2-20200204/.

[30] Yasunori Yamamoto, Atsuko Yamaguchi, and Andrea Splendiani. 2018. YummyData: providing high-quality open life science data. *Database: The Journal of Biological Databases & Curation* 2018 (2018).

[31] Semih Yumusak, Erdogan Dogdu, Halife Kodaz, Andreas Kamilaris, and Pierre-Yves Vandenbussche. 2017. SpEnD: Linked data SPARQL endpoints discovery using search engines. *IEICE TRANSACTIONS on Information and Systems* 100, 4 (2017), 758–767.

[32] Amrapali Zaveri, Anisa Rula, Andrea Maurino, Ricardo Pietrobon, Jens Lehmann, and Sören Auer. 2016. Quality assessment for Linked Data: A Survey. *Semantic Web* 7, 1 (Jan. 2016), 63–93. https://doi.org/10.3233/SW-150175