Whyis 2: An Open Source Framework for Knowledge Graph Development and Research

Jamie McCusker(✉) and Deborah L. McGuinness

Rensselaer Polytechnic Institute, Troy, NY 12180, USA
{mccusj2,dlm}@rpi.edu

Abstract. Whyis is the first open source framework for creating custom provenance-driven knowledge graph applications, or *KGApps*, supporting three principal tasks: knowledge curation, inference, and interaction. It has been used in knowledge graph projects in materials science, health informatics, and radio spectrum policy. All knowledge in Whyis graphs are encapsulated in nanopublications, which simplifies and standardizes the production of qualified knowledge in knowledge graphs. The architecture of Whyis enables what we consider to be essential requirements for knowledge graph construction, maintenance, and use. These requirements include support for automated and manual curation of knowledge from diverse sources, provenance traces of all knowledge, domain-specific user interaction, and generalized distributed knowledge inference. We coin the term "Nano-scale knowledge graph" to refer to nanopublication-driven knowledge graphs. Knowledge graph developers can use Whyis to configure custom sets of knowledge curation pipelines using custom data importers and semantic extract, transform, and load scripts. The flexible, nanopublication-based architecture of Whyis lets knowledge graph developers integrate, extend, and publish knowledge from heterogeneous sources on the web. Whyis KGApps and are easily developed locally, managed using source control, and deployable via continuous integration, server deployment scripts, and as docker containers.

Keywords: knowledge graphs · software framework · research framework

Resource type: Software framework
Python Package Index: whyis
License: Apache 2.0 License

This work was funded by the National Institute of Environmental Health Sciences (NIEHS) Award 0255-0236-4609/1U2CES026555-01, National Science Foundation (NSF) Award DMR-1310292, IBM Research AI through the AI Horizons Network, the National Spectrum Consortium (NSC) project number NSC-17-7030, and by the Gates Foundation through Healthy Birth, Growth, and Development knowledge integration (HBGDki). Any opinions, findings and conclusions or recommendations expressed in this material are those the authors and do not necessarily reflect the views of AFRL, IBM, or the Gates Foundation.

C. Pesquita et al. (Eds.): ESWC 2023, LNCS 13870, pp. 538–554, 2023.
https://doi.org/10.1007/978-3-031-33455-9_32

Documentation URL: https://whyis.readthedocs.io
Docker pull command: docker pull tetherlessworld/whyis
Source Code URL: https://github.com/tetherless-world/whyis
Example Project URL: https://github.com/whyiskg/les-mis-demo

1 Introduction

Knowledge graphs have become an important component of commercial and research applications on the Web. Google was one of the first to promote a semantic metadata organizational model described as a "knowledge graph," [33] and many other organizations have since used the term in the literature and in less formal communication. We believe that that successful knowledge graph construction requires more than simply storing and serving graph-oriented data, or even data in the Resource Description Format [6] or Linked Data [4]. Knowledge graphs need to be easily maintainable and usable in sometimes complex application settings. For instance, keeping a knowledge graph up to date can require developing a knowledge curation pipeline that either replaces the graph wholesale whenever updates are made, or requires detailed tracking of knowledge provenance across multiple data sources. Additionally, applying reasoning systems to graphs from diverse, potentially conflicting sources becomes very difficult, which has resulted in investigations of new kinds of reasoning paradigms [19].

Beyond this, it is becoming clear that other sorts of knowledge inference have become important to knowledge graph construction. NLP methods and other machine learning methods are commonly demonstrated as potential sources of knowledge graph construction. User interfaces are also key to the success of a knowledge graph, especially when supporting computational users. It is insufficient simply to provide a SPARQL endpoint or to list out the statements relevant to a single entity for most tasks. Google's knowledge graph, for instance, takes the semantic type of the entity into account when rendering information about that entity. Domain-specific APIs also help smooth the integration of knowledge graphs into existing systems. Finally, these challenges are dependent on high-quality knowledge provenance that is inherent in the design of any knowledge graph system, and not merely an afterthought.

The above challenges are not currently met by a reusable knowledge graph framework or architecture. We have therefore developed Whyis as a framework for developing knowledge graphs to support the above challenges. As shown in Fig. 1, Whyis provides a semantic analysis ecosystem: an environment that supports research and development of semantic analytics that we have previously had to build custom applications for [21,23]. Users interact through a suite of views into the knowledge graph, driven by the type of node and view requested in the URL. Knowledge is curated into the graph through knowledge curation methods, including Semantic ETL, external linked data mapping, and Natural Language Processing (NLP). Autonomous inference agents expand the available knowledge using traditional deductive reasoning as well as inductive methods that can include predictive models, statistical reasoners, and machine learning. We review case studies of projects that have used Whyis for knowledge graph development in materials science, health informatics, and radio spectrum policy.

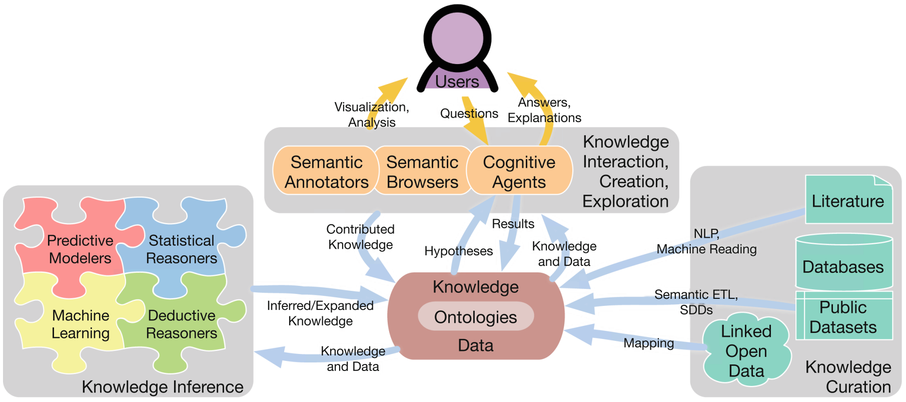


Fig. 1. The semantic ecosystem enabled by the Whyis framework for knowledge creation, interaction, and inference.

2 Approach

Whyis is a framework for developing nano-scale knowledge graph applications. It is a code-focused approach that enables software developers to create knowledge graph applications (KGApps) using minimal modifications, allowing for the use of code-oriented deployment and management tools like GitHub, Docker, and other DevOps tools. While it is possible and useful to use Whyis 2 to create KGApps without changing any code, much of the customization capabilities will only be useful if it is customized to some degree. We expect that most knowledge graph creators will use Javascript and Python to customize their KGApp for better user interaction, knowledge curation, and inference. In the software industry, this is called a low code/no code approach, allowing creators of knowledge graphs to do so without coding, but if they need to add code, it would be minimal.

Nano-scale knowledge graphs use *nanpublications* to encapsulate every piece of knowledge introduced into knowledge graphs it manages. Introduced in [24] and expanded on in [10], a nanopublication is composed of three named RDF graphs: an *Assertion* graph, knowledge encoded in RDF (which can be however many RDF statements as appropriate); a *Provenance* graph, which explains what the justification for the assertion is, and a *Publication Info* graph, which provides publication details, including attribution, of the nanopublication itself. We see knowledge graphs that include the level of granularity supported by nanopublications (thus nano-scale) as essential to fine-grained management of knowledge in knowledge graphs that are curated and inferred from diverse sources and can change on an ongoing basis. Other systems, like DBpedia [1] and Uniprot [28], have very rough grained management of knowledge using very large named graphs, which limit the ability to version, explain, infer from, and annotate the knowledge.

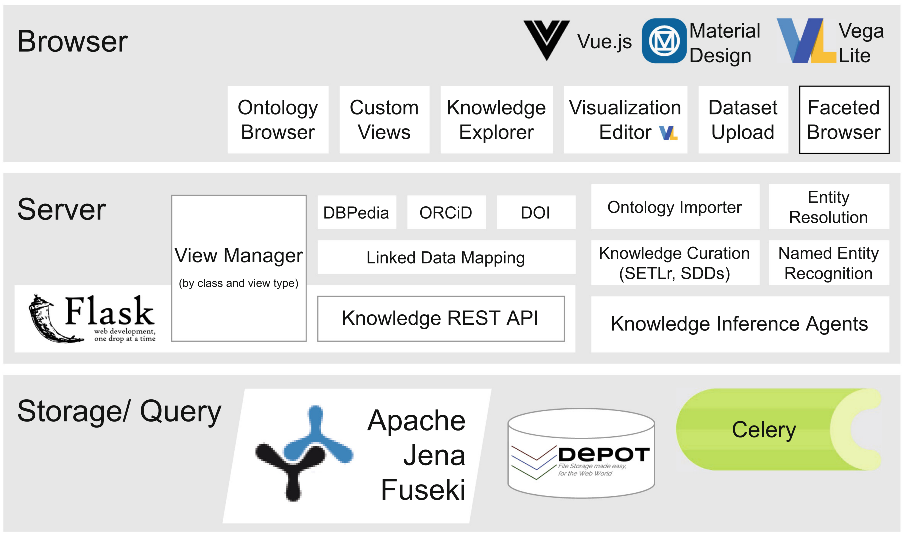


Fig. 2. The Whyis technology stack. Nanopublications are stored in the RDF database. Files can be uploaded and stored in a special File Depot instance as well. Celery is used to invoke and manage a set of autonomic inference agents, which listen for graph changes and respond with additional nanopublications. Users interact with the graph through a set of views that are configured by node type and are based on the Flask templating system Jinja2.

2.1 Whyis 2 Changes and Improvements

While Whyis 1 was only available as a Flask application, Whyis 2 is available as a python package through the Python Package Index as “whyis”, and can be installed using Python 3.7 or later. Whyis 2 provides a number of additional capabilities and modes of operation above Whyis 1. It is now possible to run Whyis in “embedded” mode, which does not require users to install system services for Celery, Redis, and Fuseki. It also provides a number of auto-generated deployment scripts to create production servers from initial KGApps with minimal customization. We also included the ability to back up and restore KGApp RDF databases and file repositories to improve deployment and maintenance of production knowledge graphs. In order to improve long-term support, we also migrated Whyis from Blazegraph, which hadn’t been recently maintained, to

Fuseki, which is still being actively improved. We also found Fuseki to be the only RDF database that could sustain ongoing edits while still providing performant read access to the database. Since it uses Apache Jena Fuseki, it also requires Java JDK 11 or later. Whyis 2 is also available as a Docker hub image using the tag "tetherlessworld/whyis", and has detailed documentation available at https://whyis.readthedocs.io.

2.2 Architecture

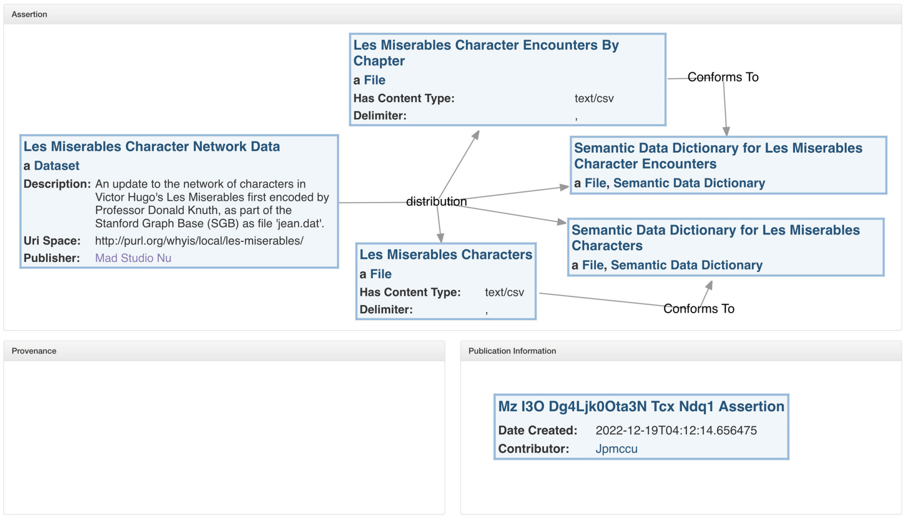
Whyis is written in Python using the Flask framework, and uses a number of existing infrastructure tools to work, as shown in Fig. 2. The RDF database used by default is Apache Jena Fuseki,[1] which provides for modular enhancement for additional capabilities. Whyis uses the SPARQL 1.1 Query [11], and Graph Store HTTP Protocols [25]. Any RDF database that supports those protocols can be a drop-in replacement for Fuseki. A read-only SPARQL endpoint is available via '/sparql', along with a Yet Another SPARQL GUI (YASGUI)-based UI [29].

Storage is provided using the FileDepot Python library[2] to provide file-based persistence of uploaded files. FileDepot abstracts the storage layer to configurable backends, handles storage of content type, file names, and other metadata, and provides durable identifiers for each file. It provides backends for a number of file storage methods, including local files, Amazon S3,[3] MongoDB GridFS,[4] and relational databases. Whyis also relies on a task queuing system called Celery[5] that can be scaled by adding more task workers on remote machines.

Knowledge graph developers create a new KGApp by simply running the "whyis" command in an empty directory. The system will generate a python module that contains the configuration, templates, and starter code files that allow developers to customize Whyis to their purposes. Views, templates, and code all live within this revision control-friendly Python project to better enable the management and staging of a production system.

To illustrate the capabilities and structure of Whyis, we created a demonstration knowledge graph of characters and their interactions from the novel *Les Miserables*, as originally created by Donald Knuth and maintained by Media and Design Studio[6] The demonstration graph is available at the Example Project URL. We have loaded the graph with an initial description of the network as a *dcat:Dataset*,[7], as shown in Fig. 3.

[1] https://jena.apache.org/documentation/fuseki2/.
[2] http://depot.readthedocs.io.
[3] https://aws.amazon.com/s3.
[4] https://docs.mongodb.com/manual/core/gridfs.
[5] http://www.celeryproject.org.
[6] Available at https://github.com/MADStudioNU/lesmiserables-character-network.
[7] https://github.com/whyiskg/les-mis-demo/blob/main/data/les-miserables.ttl.



Assertion

Les Miserables Character Encounters By Chapter
a File
Has Content Type: text/csv
Delimiter: ,

Les Miserables Character Network Data
a Dataset
Description: An update to the network of characters in Victor Hugo's Les Miserables first encoded by Professor Donald Knuth, as part of the Stanford Graph Base (SGB) as file 'jean.dat'.
Uri Space: http://purl.org/whyis/local/les-miserables/
Publisher: Mad Studio Nu

Semantic Data Dictionary for Les Miserables Character Encounters
a File, Semantic Data Dictionary

Semantic Data Dictionary for Les Miserables Characters
a File, Semantic Data Dictionary

Les Miserables Characters
a File
Has Content Type: text/csv
Delimiter: ,

Provenance

Publication Information

Mz I3O Dg4Ljk0Ota3N Tcx Ndq1 Assertion
Date Created: 2022-12-19T04:12:14.656475
Contributor: Jpmccu

Fig. 3. A rendering of the *dcat:Dataset* “Les Miserables Character Network Data” with its nanopublication. Metadata is uploaded to the system to allow for future discovery, download, and/or processing. The “Publication Info” graph traces that the assertion was created by the user ‘jpmccu’ on 2022-12-19, which was captured automatically. There is no other provenance of the graph.

2.3 Enabling Knowledge Curation

Whyis supports knowledge curation through several pipelines: direct user interaction, direct loading of RDF, semantic ETL (sETL) scripts, semantic data dictionaries, and on-demand loading of linked data from the linked data web. These approaches can be mixed and matched as needed by knowledge graph developers.

Users can interact with the default Whyis user interface to annotate graph entities with links between entities (using defined Object Properties), add attributes to entities (using Datatype Properties), and add types to entities (using OWL or RDFS Classes). Users can also upload pre-generated RDF using the web interface or KGApp developers can use the *whyis load* command available from the command line.

A commonly used approach to integrate knowledge into a KGApp is using the Semantic Extract, Transform, and Load-r (SETLr) [20] to support conversion of tabular data, tabular and non-tabular JSON, XML, HTML, and other custom formats (through embedded python) into RDF suitable for the knowledge graph, as well as transforming existing RDF into a better desired representation. By loading SETL scripts (written in RDF) into the knowledge graph, the SETLr inference agent is triggered, which runs the script and imports the generated RDF. SETLr itself is powerful enough to support the creation of named graphs, which lets users control not just nanopublication assertions (as would be the case if they were simply generating triples), but also provenance and publication info. SETLr in Whyis also supports the parameterization of SETL scripts by file type. Users can upload files to nodes by HTTP POSTing a file to a node's URI. The node then represents that file. When adding new metadata about that node, it can include *rdf:type*. If a file node has a type that matches one that is used in a SETL script, the file is converted using that script into RDF. This lets users (and developers) upload domain-specific file types to contribute knowledge.

Another way to import knowledge is for domain specialists to write Semantic Data Dictionaries (SDDs) [27], and link them to uploaded data files. A SDD is an Excel spreadsheet that abstracts away the particulars of RDF modeling to allow for domain scientists to describe data at a level of abstraction they are familiar with. The SDDAgent looks for SDDs attached to data files, and is able to compile a SETL script for the SETLr agent to process. The end result is that users can write high-level descriptions of their data, while gaining the scalability benefits of writing a SETL script by hand. The structure[8] needed to process the two SDDs used in the *Les Miserables* knowledge graph are shown in Fig. 3.

The last to import knowledge is to configure an on-demand Linked Data importer. Whyis provides a flexible Linked Data importer that can load RDF from remote Linked Data sources by URL prefix. We have successfully tested use of this importer with DOI [26], OBO Foundries [34], Uniprot [28], DBPedia [1], and other project-specific resources. It supports the insertion of API keys, content negotiation, and HTTP authentication using a netrc file. It tracks the last modified time of remote RDF to only update when remote data has changed and provides provenance indicating that the imported RDF *prov:wasQuotedFrom* the original URL. Examples are available in the default configuration file in the *NAMESPACES* entry.[9] Whyis also provides a file importer that, rather than parsing the remote file as RDF, loads the file into the file depot. This can be invoked on-demand, so that metadata can be loaded from one SETL script about a collection of files, then other SETL scripts can process those files based on the types added, and the files would be dynamically downloaded to Whyis for processing.

2.4 Enabling Knowledge Interaction

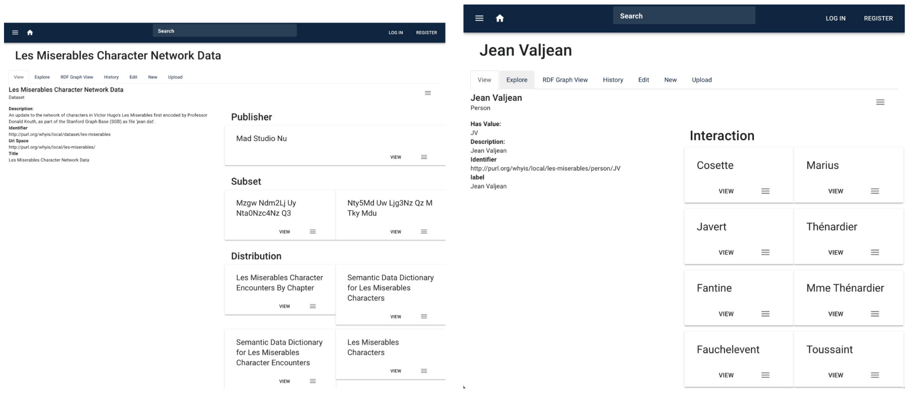
To support the Knowledge Interaction user stories, developers of Whyis knowledge graphs can create custom views for nodes by both the *rdf:type* of the node and the *view* URL parameter. These views are looked up as templates and rendered using the Jinja2 templating engine.[10] This is configured in a turtle file in the KGApp directory (vocab.ttl), where viewed classes and view properties are defined. For more details, please see the view documentation.[11]

Through the use of nanopublications, developers can provide explanation for all assertions made in the graph by accessing the linked provenance graph

[8] https://github.com/whyiskg/les-mis-demo/blob/main/data/les-miserables.ttl.
[9] https://github.com/tetherless-world/whyis/blob/main/whyis/config/default.py.
[10] http://jinja.pocoo.org/docs/2.10.
[11] https://whyis.readthedocs.io/en/latest/views.html.

when a user asks for more details. Search is also supported through an entity resolution-based autocomplete and a full text search page.

Whyis provides a number of built-in views that can be customized as needed. In Fig. 4, we see standard entity views (Figs. 4a and 4b). Whyis also provides the ability to create custom visualizations using a SPARQL query and Vega Lite [31], as explained in Deagen *et al.* Fig. 5 demonstrates how it can be used to generate custom, dynamic visualizations of data using many different approaches.



Les Miserables Character Network Data

Les Miserables Character Network Data
Dataset

Description:
An update to the network of characters in Victor Hugo's Les Miserables first encoded by Professor Donald Knuth, as part of the Stanford Graph Base (SGB) as file 'jean.dat'.
Identifier
http://purl.org/whyis/local/dataset/les-miserables
Uri Space
http://purl.org/whyis/local/les-miserables/
Title
Les Miserables Character Network Data

Publisher
Mad Studio Nu

Subset
Mzgw Ndm2Lj Uy Nta0Nzc4Nz Q3
Nty5Md Uw Ljg3Nz Qz M Tky Mdu

Distribution
Les Miserables Character Encounters By Chapter
Semantic Data Dictionary for Les Miserables Characters
Semantic Data Dictionary for Les Miserables Character Encounters
Les Miserables Characters

Jean Valjean

Jean Valjean
Person

Has Value:
JV
Description:
Jean Valjean
Identifier
http://purl.org/whyis/local/les-miserables/person/JV
label
Jean Valjean

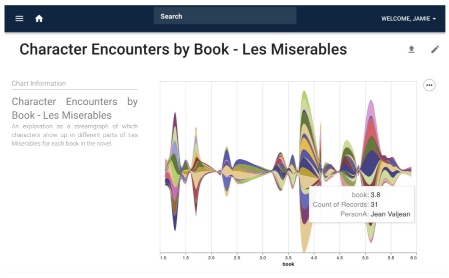
Interaction
Cosette
Marius
Javert
Thénardier
Fantine
Mme Thénardier
Fauchelevent
Toussaint

(a) The *Les Miserables* dataset. (b) Jean Valjean.

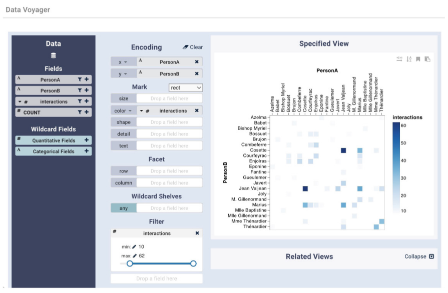
Fig. 4. Views for entities in knowledge graphs are fairly flexible, but also extensible.

2.5 Enabling Knowledge Inference

Knowledge Inference in Whyis is performed by a suite of inference agents, each performing the analogue to a single rule in traditional deductive inferencing. Inference agents support a much broader means of creating inference "rules" than simple if-then entailments. Inference agents identify entities of interest using a SPARQL query (effectively the antecedent) and a python function that generates new RDF as the consequent. Agent developers create these queries and functions as a way to expand the capabilities of their knowledge graphs, and use integration at the knowledge modeling level to provide knowledge processing workflows. The agent framework provides custom inference capability, and is composed of a SPARQL query that serves as the "rule" "body" and a python function that serves has the "head", which generates additional RDF. The agent is invoked when new nanopublications are added to the knowledge graph that match the SPARQL query defined by the agent. Developers can choose to run this query either on just the single nanopublication that has been added, or on the entire graph. Whole-graph queries will need to exclude query matches that would cause the agent to be invoked over and over. This can take some consideration for complex cases, but excluding similar knowledge to the expected output or nodes


Search
WELCOME, JAMIE
Character Encounters by Book - Les Miserables
Chart Information
Character Encounters by Book - Les Miserables
An exploration as a streamgraph of which characters show up in different parts of Les Miserables for each book in the novel.
book: 3.8
Count of Records: 31
PersonA: Jean Valjean
book

(a) Visualizing appearances of characters in *Les Miserables* over time. Data curated into the knowledge graph can easily be re-analyzed for multiple purposes.



(b) Creating a visualization using Data Voyager in Whyis. By supplying an initial query, users can then compose many Vega Lite visualizations using an embedded Data Voyager [36] instance.

Fig. 5. Visualizing the *Les Miserables* knowledge graph using Vega Lite and Data Voyager.

that have already had the agent run on them will often suffice. The Python function "head" is invoked on each query match, and generates new RDF for to be added to the knowledge graph as a new nanopublication. The agent superclass will assign some basic provenance and publication information related to the given inference activity, but developers can expand on this by overriding the *explain()* function.

These inference agents are run using the Celery distributed task queueing system. As knowledge is added to the graph on a per-nanopublication basis, an update function in the task queue is called to invoke the agent body query, and any matches are added to the queue. As each agent processes the relevant instances, they create new nanopublications, which are also published. The update function is in turn called on these, and the process continues until there are no more matches.

Prebuilt inference agent types include some NLP support, including entity detection using noun phrase extraction, basic entity resolution against other knowledge graph nodes, and Inverse Document Frequency computation for resolved nodes. It also includes agents for knowledge curation (processing SETL scripts and SDD files), an email notifier, a nanopublication versioning archive, and an ontology import closure loader. Because of the forward-chaining nature of the inference agent system, Whyis also provides support for custom deductive rules using the autonomic.Deductor class. Developers can write rules by providing a *construct* clause as the head and a *where* clause as the body. Further, Whyis also provides customized Deductor instances that are collected up into OWL and RDFS partial profiles for RDFS, OWL 2 EL, RL, and QL. OWL 2 property chain support is still in-progress.

An example inference agent is provided in the documentation.[12] The example shows a very simple example of how an external Python library, BeautifulSoup, can be used to extract the text from a node that has HTML associated with it. The agent mostly consists of a SPARQL query that looks for entities that have have *sioc:content*. The Python function then extracts that content as HTML from an RDFlib Resource in the "i" variable, parses it using BeautifulSoup, and adds a new schema:text statement to the entity as a Resource in a new graph (as "o"). While very simple, this agent is key to other downstream agents that can extract named entites from the text. Similar agents can be written to extract text from PDFs, Word Documents, or other custom data. If they use the same *schema:text* predicate, existing downstream agents will see those statements the same way as the HTML2Text agent, streamlining knowledge processing pipelines. Many other examples are available in the Whyis source code.[13]

3 Related Work

We see the following frameworks as providing some, but not all, aspects of knowledge graphs as we define them:

Stardog According to their current marketing literature, Stardog[14] includes OWL reasoning, mapping of data silos into RDF, and custom rules.

Ontowiki provides a user interface on top of a RDF database that tracks history, allows users to browse and edit knowledge, and supports user interface extensions [2].

Callimachus calls itself a "Semantic Content Manager" and lets developers provide custom templates by object type using RDFa, and supports versioning of the knowledge graph [3].

Virtuoso Openlink Data Spaces is a linked data publishing tool for Openlink Virtuoso. It provides a set of pre-defined data import tools and a fixed set of views on the linked data it creates.[15]

Vitro is part of the Vivo project [5]. It is "a general-purpose web-based ontology and instance editor with customizable public browsing."[16] It supports the creation of new ontology classes and instances that are driven by the ontology, to view, browse, and search those instances, but does not allow users to create custom interfaces.

iTelos (in preprint [9]) is a knowledge graph development methodology for which no software framework has been provided. It provides a method for separating schema development of knowledge graphs (Entity Type Graphs, or ETGs) from entity development (Entity Graphs, or EGs). Some tools are have been developed for creation of ETGs, and nothing seems to be offered for EGs.

[12] https://whyis.readthedocs.io/en/latest/inference.html.

[13] https://github.com/tetherless-world/whyis/tree/main/whyis/autonomic.

[14] A case study: https://www.stardog.com/blog/nasas-knowledge-graph/.

[15] Documentation: https://ods.openlinksw.com/wiki/ODS.

[16] Available: https://github.com/vivo-project/Vitro.

Semantic MediaWiki [14] is an extension to MediaWiki that allows editors to annotate wiki articles with RDF triples relative to the entity represented by the current article. Semantic MediaWiki was one of the first attempts at providing semantics to the Wikipedia project. It is limited to what could be expressed in the wiki page format extension, does not provide custom user interfaces, although it does offer the ability to push annotations to a SPARQL database, some of which offer reasoning.

Wikibase [37] is the technology platform for Wikidata. It provides a user interface for editing a knowledge graph as well as APIs for large scale knowledge import. Wikibase supports the introduction of provenance through reification using RDF* [12], but it does not allow for extensible inference nor does it allow for type-oriented custom user interfaces.

None of these tools support general-purpose inference beyond Datalog-like rules (Stardog), and only Stardog provides a general purpose method for importing data. Stardog does provide truth maintenance within the scope of its reasoning methods, and Ontowiki and Callimachus provide version histories, but do not provide reasoning. None of these tools allow for arbitrary knowledge inference extensions, NLP or otherwise, and only Callimachus and Ontowiki provide extensible user interfaces.

4 Case Studies and Evaluation

We present three case studies from our projects with Whyis for the use of it as a knowledge graph development framework. While these are all published examples, Whyis has been used many times for knowledge graph development on a smaller scale for prototyping, knowledge explorations, or for proofs of concept. We also note that because the underlying database is Fuseki, the performance of Fuseki through Whyis is comparable to using Fuseki directly when using the SPARQL endpoint.

4.1 MaterialsMine: A Materials Science Knowledge Graph

The MaterialsMine knowledge graph was initially introduced as NanoMine, a knowledge graph for curating experimental results from nanocomposite materials [22]. MaterialsMine is a collaboration between RPI, Duke University, California Institute of Technology, Northwestern University, and University of Vermont. Whyis is being used by materials scientists at all of these institutions to publish curated experimental and simulation data as an integrated materials science knowledge graph. Part of the Materials Genome Initiative (MGI) [35], MaterialsMine has expanded into providing data uploads for any materials science data using the Dataset Catalog (DCAT) [18], visualization of materials science knowledge [7], and curation of metamaterial computational experiment results into the knowledge graph.

MaterialsMine uses SETLr [20] to convert the detailed XML files originally supported by Nanomine into RDF nanopublications. Tabular data, mostly generated by the metamaterials community, is described using SDDs [27], which are annotated to DCAT dataset files using *dcterms:conformsTo*. The SDDAgent is then keyed to notice these links and process those files into the knowledge graph. Currently, MaterialsMine supports the processing of static properties of metamaterials through SDDs.

Users of MaterialsMine can explore the merged results using a gallery of visualizations (see Fig. 6), which are produced using a combination of SPARQL queries against the knowledge graph and a Vega Lite grammar of graphics specification [31]. This approach was discussed in Deagen *et al.* [7], and produced over 150 visualizations of the knowledge graph.

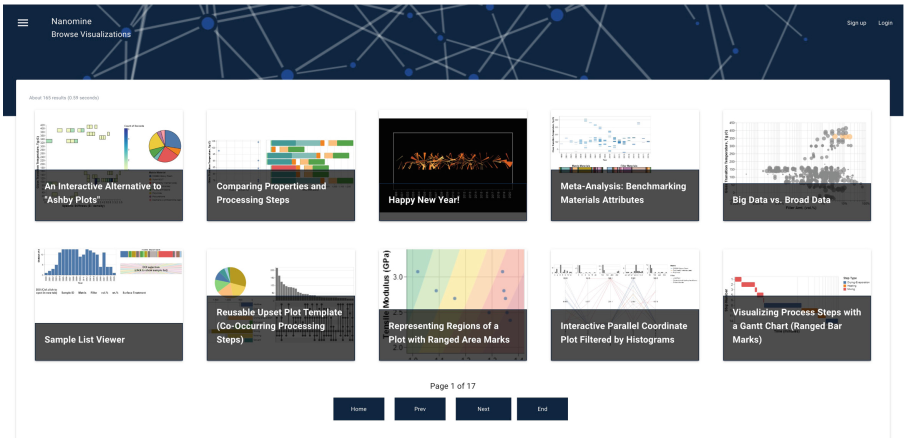

Nanomine
Browse Visualizations
Sign up
Login
About 165 results (0.59 seconds)
An Interactive Alternative to "Ashby Plots"
Comparing Properties and Processing Steps
Happy New Year!
Meta-Analysis: Benchmarking Materials Attributes
Big Data vs. Broad Data
Sample List Viewer
Reusable Upset Plot Template (Co-Occurring Processing Steps)
Representing Regions of a Plot with Ranged Area Marks
Interactive Parallel Coordinate Plot Filtered by Histograms
Visualizing Process Steps with a Gantt Chart (Ranged Bar Marks)
Page 1 of 17
Home
Prev
Next
End

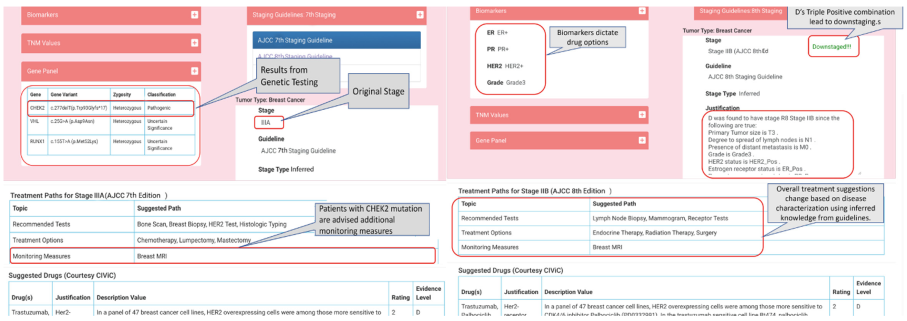
Fig. 6. A current view of the MaterialsMine visualization gallery, available at https://materialsmine.org/wi/gallery.

4.2 Dynamic Spectrum Access Policy Framework

Normally, radio spectrum allocation is performed manually by humans over the course of months. Dynamic Spectrum Access (DSA) promises to accellerate that allocation to allow for allocation of spectrum use on-demand within hours or minutes of request. The DSA Policy Framework project [30] encoded a number of radio spectrum policies as OWL Ontology fragments for fast, automated validation of radio frequency allocation requests. The Whyis knowledge graph was used to manage the domain ontology that described types of equipment used, geographic regions managed, and the backing ontologies, as well as the actual policies expressed as OWL constraints. The DSA Policy Framework was deployed as a docker application with Whyis embedded in the deliverable to manage policies and supporting terminology.

4.3 Semantic Breast Cancer Restaging

The American Joint Committee on Cancer (AJCC) periodically updates its criteria for severity of different cancers into stages. In 2018, the AJCC release its first breast cancer staging guidelines, the 8th edition [13], based on biological and molecular markers (biomarkers), which dramatically improved the accuracy of breast cancer staging, but also complicated its assessment. Seneviratne *et al.* [32] were able to use OWL ontology-based rules for cancer staging of the 7th and 8th editions and were able to automatically recompute cancer stages for a number of example breast cancer cases. The project used the Whyis reasoning framework to classify cases into relevant cancer stages in both guidelines, and was also able to explain through recorded provenance traces how it had performed that classification. The Whyis view system was also used to produce user interfaces for the project, as shown in Fig. 7 (previously published in [32]), including the ability to show inference agent explanations to users.


Biomarkers
TNM Values
Gene Panel
Staging Guidelines: 7th Staging
AJCC 7th Staging Guideline
AJCC 8th Staging Guideline
Results from Genetic Testing
Original Stage
Patients with CHEK2 mutation are advised additional monitoring measures
Biomarkers dictate drug options
Staging Guidelines:8th Staging
D's Triple Positive combination lead to downstaging.s
Downstaged!!!
Overall treatment suggestions change based on disease characterization using inferred knowledge from guidelines.

Gene	Gene Variant	Zygosity	Classification
CHEK2	c.277delT(p.Trp93Glyfs*17)	Heterozygous	Pathogenic
VHL	c.25G>A (p.Asp9Asn)	Heterozygous	Uncertain Significance
RUNX1	c.155T>A (p.Met52Lys)	Heterozygous	Uncertain Significance

Tumor Type: Breast Cancer
Stage
IIIA
Guideline
AJCC 7th Staging Guideline
Stage Type Inferred

Treatment Paths for Stage IIIA(AJCC 7th Edition)

Topic	Suggested Path
Recommended Tests	Bone Scan, Breast Biopsy, HER2 Test, Histologic Typing
Treatment Options	Chemotherapy, Lumpectomy, Mastectomy
Monitoring Measures	Breast MRI

Suggested Drugs (Courtesy CIViC)

Drug(s)	Justification	Description Value	Rating	Evidence Level
Trastuzumab,	Her2-	In a panel of 47 breast cancer cell lines, HER2 overexpressing cells were among those more sensitive to	2	D

ER ER+
PR PR+
HER2 HER2+
Grade Grade3

Tumor Type: Breast Cancer
Stage
Stage IIB (AJCC 8th Ed
Guideline
AJCC 8th Staging Guideline
Stage Type Inferred
Justification
D was found to have stage R8 Stage IIB since the following are true:
Primary Tumor size is T3 .
Degree to spread of lymph nodes is N1 .
Presence of distant metastasis is M0 .
Grade is Grade3 .
HER2 status is HER2_Pos .
Estrogen receptor status is ER_Pos .

Treatment Paths for Stage IIB (AJCC 8th Edition)

Topic	Suggested Path
Recommended Tests	Lymph Node Biopsy, Mammogram, Receptor Tests
Treatment Options	Endocrine Therapy, Radiation Therapy, Surgery
Monitoring Measures	Breast MRI

Suggested Drugs (Courtesy CIViC)

Drug(s)	Justification	Description Value	Rating	Evidence Level
Trastuzumab,	Her2-	In a panel of 47 breast cancer cell lines, HER2 overexpressing cells were among those more sensitive to	2	D

Fig. 7. (a) AJCC 7th Edition Staging Characterization. (b) AJCC 8th Edition Staging Characterization (previously published in [32]).

5 Discussion

Managing knowledge graphs at nano-scale within the Whyis semantic ecosystem has made it simple to realize core knowledge graph requirements for knowledge curation, integration and exploration. Knowledge curation from diverse sources can occur with detailed provenance of where everything comes from and what knowledge is contingent on other knowledge, using existing provenance standards. User interfaces can query knowledge provenance to provide deep explanations of what they show. Inference agents become more scalable when much of the analysis can be performed on knowledge fragments, rather than the entire knowledge graph. Whyis, through SETLr, supports the transformation of databases into rigorously modeled knowledge, regardless of the source format.

The inherent use of provenance standards opens up the ability to capture contingent knowledge that may or may not be consistent. Questions of "Who or what made this claim?", "What other knowledge have they added?", "Do we trust them?" can be asked by knowledge graph developers so that users can decide what knowledge they want to trust. We also have the option to provide services that look for evidence for a statement as well as evidence for the negation of a statement. For knowledge inference, the use of update-based general inference provides an inherent forward-chaining paradigm – when an agent adds new knowledge, the other agents are checked to see if they can provide new conclusions from that output. Further, the use of a distributed task queuing system (Celery) for implementation means that, while the inference engine may not be as fast as an in-database reasoner, it is potentially more scalable, because additional compute nodes can be added to serve as Celery workers. The central database is only responsible for recording changes and answering queries, which means that it can serve whatever knowledge has been computed so far, rather than waiting for a complete answer to be returned.

We see Whyis as a potential unifying testbed and integration point for algorithms and methods used in working with knowledge graphs. For example, NLP algorithms that purport to generate knowledge graphs and deductive or inference algorithms that expand knowledge graphs, graph learning algorithms that can infer new links between entities. This work has the potential to encourage algorithm developers to work on common knowledge representations, which will make comparisons easier across algorithms. A unified approach could make it easier to put new algorithms into practice by providing a context in which those algorithms can operate, including a common representation for existing entities in the knowledge graph and pre-loaded knowledge, and for algorithms to build off of each others' output. The challenge here is to provide a guide for NLP and other algorithm developers to map their output into a useful knowledge graph representation.

5.1 Limitations

There are a number of limitations to consider when evaluating Whyis for implementation. The inference engine is not as fast as in-database reasoners because it needs to query the database remotely, parse, and then serialize new knowledge back into the database. Currently, inference agents must be written in Python or have a Python wrapper around an external script or service. Also, the process of revising knowledge in largest imports can be slowed if there are many pre-existing revisions to retire. The commitment to using nanopublications as a transactional atom has also complicated our adoption of standards like the Linked Data Platform (LDP) [16]. LDP itself makes assumptions about the best way to manage knowledge graph fragments that are incompatible with arbitrarily-scoped nanopublications. Whyis does support LDP POST, DELETE, and PUT for nanopublication edits, but not for entities.

5.2 Future Work

Work and research on Whyis is active and ongoing; we have a number of improvements and research projects planned. While the immediate future of Whyis is secured by a number of research grants, we will be looking to plan out the long term sustainability of Whyis. Additionally, with the release of Whyis 2.0, we plan to expand usage by others, we will be providing tutorials at academic conferences and online video tutorials on using Whyis to create knowledge graphs. While most KGApp development has been driven by projects the authors were driving, we feel that Whyis is ready for use by a broader audience of knowledge graph developers.

We plan to perform a benchmark of the inference agent framework to compare both the capabilities and performance against other inference systems. We also plan to include support for the RDF Mapping Language (RML), a more familiar method of knowledge curation for the Semantic Web community than SETLr and SDDs. We are also going to research new methods for display in the knowledge explorer user interface, including custom renderings and layout using the Cytoscape.js [8], and to support the original use case developed in [21]. We are working on a suite of integrated inference agents. Inductive agents will utilize statistics and machine learning algorithms to automatically infer new knowledge and relationships in the knowledge graph. For example, we plan to investigate graph learning algorithms like [17] that can learn from the structure of published knowledge graphs to find new relations. Finally, with the increased uptake of ActivityPub-based standards in the "Fediverse" [15], we hope to explore the role of knowledge graphs and conversational agents in social networks by augmenting our commentary system to conform to the ActivityPub standard. This will allow Whyis knowledge graphs to serve as Fediverse nodes, and for Whyis computational agents to interact with humans through a medium they are already familiar with.

6 Conclusions

We introduce Whyis as the first provenance-aware open source framework for knowledge graph development that fulfills key user stories in knowledge graph curation, interaction, and inference within a unified ecosystem. We discussed the importance of nanopublications in the architecture of Whyis, and why it is valuable to develop nano-scale knowledge graphs that are built on nanopublications. The architecture of Whyis was designed to support use cases in three key areas of knowledge graph development: curation, inference, and interaction. As a result we were able to show how Whyis has been used in materials science, radio spectrum policy, and breast cancer restaging to produce valuable knowledge resources to those communities. Finally, we discussed the potential for Whyis to be a testbed of algorithms for curation of, interaction with, and inference from knowledge graphs, including algorithms that automatically build knowledge graphs using NLP, probabilistic network analysis, and machine learning methods for graph learning and statistical inference.

References

1. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: DBpedia: a nucleus for a web of open data. In: Aberer, K., et al. (eds.) ASWC/ISWC -2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-76298-0_52
2. Auer, S., Dietzold, S., Riechert, T.: OntoWiki – a tool for social, semantic collaboration. In: Cruz, I., et al. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 736–749. Springer, Heidelberg (2006). https://doi.org/10.1007/11926078_53
3. Battle, S., Wood, D., Leigh, J., Ruth, L.: The Callimachus project: RDFA as a web template language. In: Proceedings of the Third International Conference on Consuming Linked Data, vol. 905, pp. 1–14. CEUR-WS.org (2012)
4. Bizer, C., et al.: DBpedia - a crystallization point for the Web of Data. Web Semant.: Sci. Serv. Agents World Wide Web **7**(3), 154–165 (2009). https://doi.org/10.1016/j.websem.2009.07.002
5. Corson-Rikert, J., Cramer, E.J.: Vivo: enabling national networking of scientists. In: International Association for Social Science Information Services and Technology (2010)
6. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 concepts and abstract syntax. W3C Recommendation (2014)
7. Deagen, M.E., et al.: Fair and interactive data graphics from a scientific knowledge graph. Sci. Data **9** (2022)
8. Franz, M., Lopes, C.T., Huck, G., Dong, Y., Sumer, O., Bader, G.D.: Cytoscape.js: a graph theory library for visualisation and analysis. Bioinformatics **32**(2), 309–311 (2016)
9. Giunchiglia, F., Bocca, S., Fumagalli, M., Bagchi, M., Zamboni, A.: iTelos- building reusable knowledge graphs. CoRR abs/2105.09418 (2021). https://arxiv.org/abs/2105.09418
10. Groth, P., Gibson, A., Velterop, J.: The anatomy of a nanopublication. Inf. Serv. Use **30**(1), 51–56 (2010). https://doi.org/10.3233/ISU-2010-0613
11. Harris, S., Seaborne, A.: SPARQL 1.1 query language. W3C recommendation, W3C (2013). http://www.w3.org/TR/2013/REC-sparql11-query-20130321/
12. Hartig, O.: Foundations of RDF* and SPARQL*: (an alternative approach to statement-level metadata in RDF). In: AMW 2017 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, 7–9 June 2017, vol. 1912. Juan Reutter, Divesh Srivastava (2017)
13. Kalli, S., Semine, A., Cohen, S., Naber, S.P., Makim, S.S., Bahl, M.: American joint committee on cancer's staging system for breast cancer, eighth edition: what the radiologist needs to know. RadioGraphics **38**(7), 1921–1933 (2018). https://doi.org/10.1148/rg.2018180056, pMID: 30265613
14. Krötzsch, M., Vrandečić, D., Völkel, M.: Semantic MediaWiki. In: Cruz, I., et al. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 935–942. Springer, Heidelberg (2006). https://doi.org/10.1007/11926078_68
15. La Cava, L., Greco, S., Tagarelli, A.: Understanding the growth of the Fediverse through the lens of Mastodon. Appl. Netw. Sci. **6**(1), 1–35 (2021)
16. Le Hors, A.J., Speicher, S.: The linked data platform (LDP). In: Proceedings of the 22nd International Conference on World Wide Web, pp. 1–2 (2013)
17. Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: AAAI, vol. 15, pp. 2181–2187 (2015)
18. Maali, F., Erickson, J., Archer, P.: Data catalog vocabulary (DCAT). W3C recommendation. World Wide Web Consortium, pp. 29–126 (2014)

19. Margara, A., Urbani, J., Van Harmelen, F., Bal, H.: Streaming the web: reasoning over dynamic data. Web Semant.: Sci. Serv. Agents World Wide Web **25**, 24–44 (2014)
20. McCusker, J.P., Chastain, K., Rashid, S., Norris, S., McGuinness, D.L.: SETLr: the semantic extract, transform, and load-r. PeerJ Preprints **6**, e26476v1 (2018)
21. McCusker, J.P., Dumontier, M., Yan, R., He, S., Dordick, J.S., McGuinness, D.L.: Finding melanoma drugs through a probabilistic knowledge graph. PeerJ Comput. Sci. **3**, e106 (2017). https://doi.org/10.7717/peerj-cs.106
22. McCusker, J.P., Keshan, N., Rashid, S., Deagen, M., Brinson, C., McGuinness, D.L.: NanoMine: a knowledge graph for nanocomposite materials science. In: Pan, J.Z., et al. (eds.) ISWC 2020. LNCS, vol. 12507, pp. 144–159. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-62466-8_10
23. McGuinness, D.L., Bennett, K.: Integrating semantics and numerics: case study on enhancing genomic and disease data using linked data technologies. In: Proceedings of SmartData, pp. 18–20 (2015)
24. Mons, B., Velterop, J.: Nano-publication in the e-science era. In: Workshop on Semantic Web Applications in Scientific Discourse (SWASD 2009), pp. 14–15 (2009)
25. Ogbuji, C.: SPARQL 1.1 graph store HTTP protocol. W3C recommendation, W3C (2013). http://www.w3.org/TR/2013/REC-sparql11-http-rdf-update-20130321/
26. Paskin, N.: Digital object identifier (doi®) system. Encyclopedia Libr. Inf. Sci. **3**, 1586–1592 (2010)
27. Rashid, S.M., et al.: The semantic data dictionary - an approach for describing and annotating data. Data Intell. **2**, 443–486 (2020)
28. Redaschi, N.: UniProt in RDF: tackling data integration and distributed annotation with the semantic web. Nat. Precedings 1 (2009)
29. Rietveld, L., Hoekstra, R.: The YASGUI family of SPARQL clients 1. Semant. Web **8**(3), 373–383 (2017)
30. Santos, H., et al.: A semantic framework for enabling radio spectrum policy management and evaluation. In: Pan, J.Z., et al. (eds.) ISWC 2020. LNCS, vol. 12507, pp. 482–498. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-62466-8_30
31. Satyanarayan, A., Moritz, D., Wongsuphasawat, K., Heer, J.: Vega-lite: a grammar of interactive graphics. IEEE Trans. Vis. Comput. Graph. **23**(1), 341–350 (2016)
32. Seneviratne, O., et al.: Knowledge integration for disease characterization: a breast cancer example. In: Vrandečić, D., et al. (eds.) ISWC 2018. LNCS, vol. 11137, pp. 223–238. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00668-6_14
33. Singhal, A.: Introducing the knowledge graph: things, not strings. Official Google Blog (2012). https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html. Accessed 11 Apr 2016
34. Smith, B., et al.: The OBO foundry: coordinated evolution of ontologies to support biomedical data integration. Nat. Biotechnol. **25**(11), 1251 (2007)
35. Ward, C.H.: Materials genome initiative for global competitiveness (2012). https://www.mgi.gov/sites/default/files/documents/materials_genome_initiative-final.pdf
36. Wongsuphasawat, K., Moritz, D., Anand, A., Mackinlay, J., Howe, B., Heer, J.: Voyager: Exploratory analysis via faceted browsing of visualization recommendations. IEEE Trans. Vis. Comput. Graph. **22**(1), 649–658 (2015)
37. Zhou, L., et al.: The enslaved dataset: a real-world complex ontology alignment benchmark using wikibase. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 3197–3204 (2020)