Data2Services: enabling automated conversion of data to services

Vincent Emonet, Alexander Malic, Amrapali Zaveri, Andreea Grigoriu, and Michel Dumontier

Institute of Data Science, Maastricht University, The Netherlands
{firstname.lastname}@maastrichtuniversity.nl

Abstract. While data are becoming increasingly easy to find and access on the Web, significant effort and skill is still required to process the amount and diversity of data into convenient formats that are friendly to the user. Moreover, these efforts are often duplicated and are hard to reuse. Here, we describe Data2Services, a new framework to semi-automatically process heterogeneous data into target data formats, databases and services. Data2Services uses Docker to faithfully execute data transformation pipelines. These pipelines automatically convert target data into a semantic knowledge graph that can be further refined to conform to a particular data standard. The data can be loaded in a number of databases and are made accessible through native and auto-generated APIs. We describe the architecture and a prototype implementation for data in the life sciences.

Keywords: ETL · data transformation · data conversion · API

1 Introduction

There is a large and growing amount of valuable data available on the Web. These data contain relevant information to answer questions and make novel predictions. However, data come in a myriad of formats (e.g. CSV, XML, DB), which makes them difficult to integrate into a coherent knowledge graph for unspecified downstream use. Unsurprisingly, many tools have emerged to facilitate integration and analysis of diverse data. However, data transformation and integration often require substantial technical and domain expertise to do it correctly. Moreover, such transformations are hard to find and largely incompatible across tool chains. Users duplicate effort and are ultimately less productive in achieving their true objectives. Thus, easy, reliable, and reproducible transformation and publication of different kinds of data sources in target formats are needed to maximize the potential to find and reuse data in a manner that follows the spirit of the FAIR (Findable, Accessible, Interoperable, Reusable) principles [10].

An important use case in the life sciences is the collection and analysis of clinical and biomedical data to elucidate molecular mechanisms that underlie the

pathology and treatment of human disease. The National Center for Advancing Translational Sciences (NCATS) Biomedical Data Translator program[1] is a iterative effort to develop new software architectures and corresponding data and software ecosystems to generate and explore biomedical hypotheses. This project utilizes over 40 different datasets that are dispersed and represented in largely incompatible data formats and standards. This lack of interoperability makes it difficult to find answers to even the simplest questions (e.g. how many treatable diseases are there?), and even the more sophisticated questions that are important for the implementation of personalized medicine.

In this paper, we propose a software framework called Data2Services to (semi)automatically process heterogeneous data (e.g. CSV, TSV, XML) into a set of user-facing services (e.g. SPARQL endpoint, GraphQL endpoint, API). Data2Services aims to enhance the availability of structured data to domain experts by automatically providing a variety of services to the source data. This open-source tool is composed of different Docker containers for effortless usage by expert and non-expert users alike. We demonstrate the utility of our tool by applying it to a specific query relevant to the Translator program.

2 Data2Services Framework

The overall framework, illustrated in Figure 1, is based on 4 key steps:

1. Automatic generation of RDF data from input data
2. Loading of RDF data into an RDF store
3. Transformation of RDF data to RDF standard
4. Auto-configuration and deployment of data access services

Data2Services makes use of the Resource Description Framework (RDF) [3] as a formal, shared, accessible and broadly application knowledge representation language, in line with FAIR principle (Findable Accessible Interoperable Reusable). RDF offers a common data format for both data and their metadata, as well as data and the vocabularies used to describe them. RDF statements are largely in the form of "subject", "predicate", "object", "graph" quads, and can be serialized in a number of standard formats including CSV, TSV, JSON, XML, JSON-LD, Turtle, RDF/XML, N-Triples. RDF, in combination with the Web Ontology Language [1], thereby making them more expressive than simple file formats such as CSV, JSON, and XML.

2.1 Automated generation of RDF data

The generation of RDF data from input data is performed in a semi-automated manner. A semi-automated approach is currently needed, because the while RDF data can be automatically generated from a wide variety of data formats, these

[1] https://ncats.nih.gov/translator/about, which we shall refer to as "Translator program" in the rest of the paper.

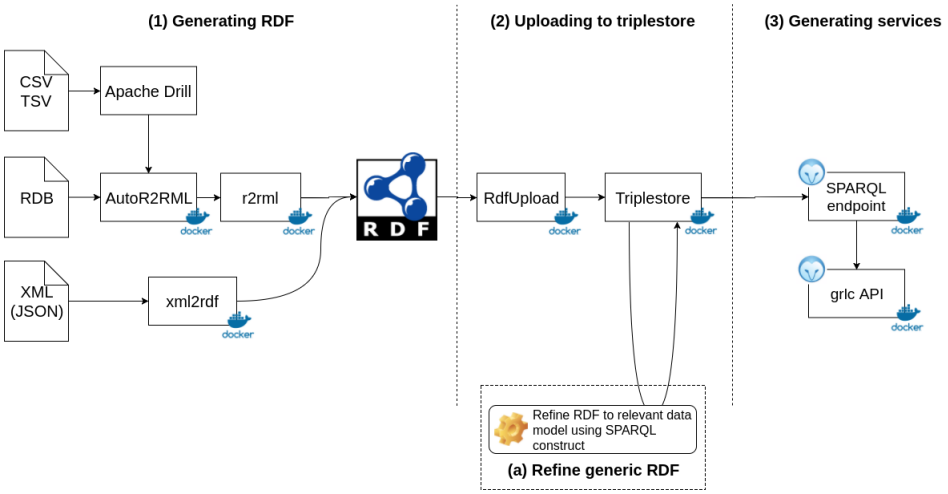

Fig. 1. Overview of Data2Services Framework

resulting data may generate incorrect relations and lack the intended semantics. To produce more accurate RDF data, we need a language to either add the RDF semantics prior to the transformation or after the transformation. We use of relational mapping languages such as R2RML[2], a W3C standard, while we turn to SPARQL [9] to transform an existing RDF dataset to some community standard. Input data are processed by an R2RML processor along with the R2RML mapping file to generate the output data.

Tabular files and relational database transforms: Tabular files (e.g.: CSV, TSV, PSV) are exposed as a relational database tables using Apache Drill[3]. Each file is represented as a table, and each column is considered as the attribute (properties in RDF) of the table. Attributes are named after the column headers.

R2RML mapping files are automatically generated from relational databases and Apache Drill accessible files through SQL queries issued by our AutoR2RML[4] tool. Table names are used to generate the subject type identifiers while attribute names are the basis for predicate identifiers. An R2RML processor[5] generates the resulting RDF using the mapping files in combination with the source data.

XML transforms: We developed an xml2rdf[6] tool to stream process an XML file to RDF, largely following the XML data structure. The output RDF captures

[2] https://www.w3.org/TR/r2rml/
[3] https://github.com/amalic/apache-drill
[4] https://github.com/amalic/AutoR2RML
[5] https://github.com/chrdebru/r2rml
[6] https://github.com/MaastrichtU-IDS/xml2rdf

name of the XML node, its XPath location, its value, any children and their attributes. We envision to broader version of this tool to process any kind of tree-like document format (JSON, YAML).

2.2 RDF Upload

The generated RDF data is then loaded into an RDF database through its REST API or SPARQL interface using RdfUpload[7]. RdfUpload is a project that automatically uploads an RDF file into a specified GraphDB SPARQL or HTTP Repository endpoint.

2.3 Transform RDF to target model

Finally, SPARQL insert are run to transform generic RDF representation of the XML data structure into the target data model. Those SPARQL queries are manually designed by a user aware of the input file structure, the target data model and the SPARQL query language.

2.4 Access the data through services

Once data is transformed into the target database, it can be accessed using a variety of services. RDF databases typically provide a SPARQL endpoint to query RDF data.

Here, we envision the development or inclusion of a variety of service interfaces on top of the RDF databases. This can include the automatic generation of REST APIs to manipulate entities and relations in the knowledge graph[8], the encapsulation of SPARQL queries as REST operations [9], the provision of standardized dataset metadata[10] and API metadata [11], the use of standardized hypermedia controls [11][12], the use of graph query languages (GraphQL, HyperGraphQL, openCypher, Gremlin, SPARQL) and user interfaces[13][14].

3 Data2Services Evaluation

To demonstrate the Data2Services framework we use following query, relevant to the Translator program: *Q1. Which drugs, or compounds, target gene products of a [gene]?*. To answer the query, we use two datasets from a pool of 40 datasets used by the Translator program. These datasets are: (i) HUGO Gene

[7] https://github.com/MaastrichtU-IDS/RdfUpload
[8] http://www.dfki.uni-kl.de/~mschroeder/demo/sparql-rest-api/
[9] https://github.com/CLARIAH/grlc
[10] https://www.w3.org/TR/hcls-dataset/
[11] https://www.w3.org/TR/ldp
[12] https://spring.io/understanding/HATEOAS
[13] http://yasgui.laurensrietveld.nl
[14] http://www.irisa.fr/LIS/ferre/sparklis/

Nomenclature Committee (HGNC)[15], a curated repository of HGNC-approved gene names, gene families and associated resources that provides detailed information about genes and gene products, available in TSV format, and (ii) DrugBank, a resource containing detailed information on drugs and the gene products they target[16], available in XML format. Listings 1.1 and 1.2 show excerpts of the datasets used. We chose these particular datasets because they contain relevant information to answer the query.

In the following, we outline the steps taken to execute the Data2Services pipeline on these two datasets using two different services to answer this query.

```
hgnc_id     symbol      name
HGNC:3535   F2          coagulation factor II, thrombin
HGNC:3537   F2R         coagulation factor II thrombin receptor
```

Listing 1.1. Excerpt of the HGNC TSV dataset.

```
<?xml version="1.0" encoding="UTF-8"?>
<drugbank xmlns="http://www.drugbank.ca" version="5.1">
  <drug type="biotech" created="2005-06-13" updated="2018-07-02">
    <drugbank-id primary="true">DB00001</drugbank-id>
    <name>Lepirudin</name>
    <target>
      <id>BE0000048</id>
      <name>Prothrombin</name>
      <external-identifier>
        <identifier>HGNC:3535</identifier>
```

Listing 1.2. Excerpt of the DrugBank XML dataset.

3.1 Automated generation of generic RDF

As the first step, we downloaded the HGNC dataset from ftp://ftp.ebi.ac.uk/pub/databases/genenames/hgnc_complete_set.txt.gz and the Drugbank dataset from https://www.drugbank.ca/releases/5-1-1/downloads/all-full-database[17]. To execute the Data2Services pipeline, the downloaded files need to be uncompressed and placed in different directories mapped into /data directory inside the Apache Drill Docker container. For convenience we have created two Shell scripts to build the Data2Service pipeline Docker containers, and start both Apache Drill and Ontotext GraphDB as services, as shown in Listing 1.3.

```
$ git clone --recursive https://github.com/MaastrichtU-IDS/data2services-
    pipeline.git
$ cd data2services-pipeline
$ git checkout tags/swat4ls
$ ./build.sh
$ ./startup.sh
```

Listing 1.3. Buiding the pipeline Docker images from GitHub

[15] https://www.genenames.org
[16] https://www.drugbank.ca/
[17] Needs an account to be created to download.

Before continuing, a repository needs to be created for GraphDB. It can be done by accessing GraphDB at http://localhost:7200 and go to: Setup → Repositories → Create new repository. Choose "test" as repository ID and check "Use context index".

To automatically run the data2services-pipeline that generates RDF out of the input dataset, the only requirement is to define a YAML configuration for each dataset as shown in Listing 1.4. The YAML allows the user to configure different parameters such as the path to the input file, Apache Drill and GraphDB parameters.

```
WORKING_DIRECTORY: "/data/hgnc/hgnc_complete_set.txt" #for HGNC
WORKING_DIRECTORY: "/data/drugbank/full_database.xml" #for DrugBank

JDBC_URL: "jdbc:drill:drillbit=drill:31010"
JDBC_CONTAINER: "drill"
GRAPHDB_URL: "http://graphdb:7200"
GRAPHDB_REPOSITORY: "test"
GRAPHDB_USERNAME: "import_user"
GRAPHDB_PASSWORD: "test"
```

Listing 1.4. YAML configuration file for HGNC or DrugBank

Then, Data2Services can be executed by providing the YAML configuration file to the run.sh script in the data2services-pipeline directory with the command `$ ./run.sh /path/to/config.yaml`.

HGNC processing: As a result of executing the YAML configuration file, a R2RML mapping file is produced for HGNC, in the directory where the input file is stored, as shown in Listing 1.5.

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
<#HgncMapping>
rr:logicalTable [ rr:sqlQuery """
select row_number() over (partition by filename) as autor2rml_rownum
 ,columns[0] as `HgncId`
 ,columns[1] as `ApprovedSymbol`
from dfs.root.`/data/hgnc/hgnc_complete_set.tsv`;"""];
rr:subjectMap [
  rr:termType rr:IRI;
  rr:template "http://data2services/data/hgnc/hgnc_complete_set.tsv/{
      autor2rml_rownum}";
];
rr:predicateObjectMap [
  rr:predicate <http://data2services/data/hgnc/hgnc_complete_set.tsv/HgncId>;
  rr:objectMap [ rr:column "HgncId" ];
];
```

Listing 1.5. Excerpt of the R2RML mapping file for HGNC TSV file

Then the R2RML implementation is executed to extract the data from the TSV input file to produce the generic RDF from the R2RML mapping files as shown in 1.6.

```
PREFIX d2s: <http://data2services/data/hgnc/hgnc_complete_set.txt/>
d2s:3535    d2s:symbol    "F2" ;
            d2s:name    "coagulation factor II, thrombin" .
```

Listing 1.6. Excerpt of the produced triples for HGNC TSV file

DrugBank processing: Executing the Data2Services pipeline on the Drugbank XML file produces a generic RDF model representing the DrugBank XML structure as shown in Listing 1.7.

```
d2sdata:3a7d47b8-c734 rdf:type d2smodel:drugbank/drug/drugbank-id .
d2sdata:0c1f3d83-5563 d2smodel:hasChild d2sdata:3a7d47b8-c734 .
d2sdata:3a7d47b8-c734 d2smodel:model/hasValue "DB00001" .
```

Listing 1.7. Excerpt of triples generated from Drugbank XML structure

3.2 RDF Upload

As the next step, RdfUpload is executed to load the RDF data to the "test" repository of the GraphDB service running on http://localhost:7200. RdfUpload has so far only been tested on GraphDB, but we envision to develop it as a generic tool that can be used on most of the popular RDF databases like Virtuoso.

3.3 Mapping to the BioLink model

As part of the standardization of knowledge graphs, the Translator project has created BioLink[18], is a high level datamodel of biological entities (genes, diseases, phenotypes, pathways, individuals, substances, etc) and their associations. We crafted two SPARQL INSERT queries [19] [20] to generate new RDF datasets that are compliant with the BioLink model as shown for HGNC in Listing 1.8.

```
PREFIX d2s:<http://data2services/data/hgnc/hgnc_complete_set.ts/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX bioentity: <http://bioentity.io/vocab/>
INSERT {
  ?hgncUri a bioentity:Gene .
  ?hgncUri rdfs:label ?geneName .
  ?hgncUri <http://purl.org/dc/terms/identifier> ?hgncid .
  ?hgncUri bioentity:id ?hgncUri .
  ?hgncUri bioentity:systematic_synonym ?symbol .
} WHERE {
 SELECT ?s ?hgncid ?geneName ?symbol ?hgncUri {
   ?s d2s:ApprovedName ?geneName .
   ?s d2s:HgncId ?hgncid .
   ?s d2s:ApprovedSymbol ?symbol .
   ?s ?p ?o .
   BIND ( iri(concat("http://identifiers.org/", lcase(?hgncid))) AS ?hgncUri)
 }}
```

Listing 1.8. SPARQL construct query to convert HGNC to BioLink

The transformed RDF data contains drug data from Drugbank, linked to gene data from HGNC in a manner that is compliant with the BioLink model. The next step is to use the available interfaces to answer the research question.

[18] https://biolink.github.io/biolink-model/

[19] https://github.com/vemonet/ncats-grlc-api/blob/master/insert_biolink_drugbank.rq

[20] https://github.com/vemonet/ncats-grlc-api/blob/master/insert_biolink_hgnc.rq

3.4 Services

Executing the Data2Services pipeline enables the BioLink data to be queried through two services: (i) SPARQL and (ii) an HTTP API.

SPARQL: The original question can be answered by executing a SPARQL query which retrieves all drugs linked to a given gene, as shown for the gene "coagulation factor II, thrombin" in Listing 1.9. The results show that 23 drugs are affecting this gene. Then any other question can be translated from natural language to a SPARQL query that will extract the requested informations from the graph.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX bioentity: <http://bioentity.io/vocab/>
SELECT distinct ?gene ?geneLabel ?geneProductLabel ?drug ?drugLabel
{   ?gene a bioentity:Gene .
    ?gene bioentity:id ?geneId .
    ?gene rdfs:label ?geneLabel .
    ?gene bioentity:has_gene_product ?geneProduct .
    ?geneProduct rdfs:label ?geneProductLabel .
    ?drug bioentity:affects ?geneProduct .
    ?drug a bioentity:Drug .
    ?drug rdfs:label ?drugLabel .
    FILTER regex(str(?geneLabel), "coagulation factor II, thrombin") .
}
```

Listing 1.9. SPARQL query to answer Which drugs, or compounds, target gene products of thrombin.

HTTP API: We used grlc[21] to expose the SPARQL query to answer the question as an HTTP web service. grlc is a lightweight server that takes SPARQL queries curated in GitHub repositories, and translates them to Linked Data Web APIs. Users are not required to know SPARQL to query their data, but instead can access a web API. We implemented a Swagger API with one call to retrieve URI and label of drugs that affect a gene defined by the user using its HGNC identifier e.g: HGNC:3535. This API is available at http://grlc.io/api/vemonet/ncats-grlc-api.

4 Related Work

A significant amount of research has been conducted in the domain of data conversion to a standard, semantically meaningful format. OpenRefine[22] offers a web user interface a to manipulate tabular data and generate RDF[23]. However, this user must be knowledgeable about RDF data modeling to generate sensible RDF data. Karma [6] allows customizable RDF conversion through a

[21] https://github.com/CLARIAH/grlc
[22] http://openrefine.org/
[23] https://github.com/fadmaa/grefine-rdf-extension/releases

web browser, but producing high quality mappings from ontologies with various structured sources (formats including databases, spreadsheets, delimited text files, XML, JSON, KML) requires expert ontology knowledge. Another example of user controlled RDF conversion is Sparqlify[24], which depends on the non-standardized Sparqlification Mapping Language(SML) and can be difficult for inexperienced users [5]. Other approaches also involve the transformation of XML data by using XSLT stylesheets [2] or templates [7]. SETLr [8] is another tool which can convert a variety of data types to RDF using JSLDT, together with Jinja Templates and Python Expressions, and is a great option for professionals familiar with those languages.

5 Conclusions, Limitations and Future Work

In this paper, we describe Data2Services, a software framework to automatically process heterogeneous data with multiple interfaces to access those data. This open-source framework makes use of Docker containers to properly configure software components and the execution of the workflow. We demonstrate the utility of Data2Services by transforming life science data and answering a question that has arisen in the Translator program.

This work represents a preliminary effort in which there are several limitations. While the automatic conversion of data does produce an RDF graph, it lacks a strong semantics that is obtained by mapping data to domain ontologies. Our strategy in this work was to show how a second transformation, expressed as a SPARQL INSERT query over the automatically converted data, could be mapped to a community data model (BioLink) for use by that community. We also acknowledge that it may be possible to edit the autogenerated R2ML file to produce BioLink data, but this would not be available for the XML conversion. Indeed, it would be desirable to have one declarative language for the transformation of a greater set of initial data format into RDF. RML [4] promises such a language, with existing processors for relational, XML, JSON data. However, preliminary work with the RML processor revealed that it does not scale currently with large data files because it loads the entire data into memory into so called logical structures, which enable joining data from so called logical sources[25]. Nonetheless, efforts to craft friendly user interfaces that hide the complexity of mapping languages could be useful to generate mappings from non-traditional users for all kinds of data files.

Our future work will explore the automated capture of metadata such as provenance of the data collection and processing in a manner that is compliant to community standards, the incorporation of data quality assessments on RDF data [12], and the evaluation of the framework in terms of performance with other use cases and data sources, both large and small.

[24] https://github.com/SmartDataAnalytics/Sparqlify
[25] http://rml.io/spec.html#logical-join

6 Acknowledgements

Support for the preparation of this project was provided by NCATS, through the Biomedical Data Translator program (NIH awards OT3TR002019 [Orange] and OT3TR002027 [Red]). Any opinions expressed in this document are those of the Translator community writ large and do not necessarily reflect the views of NCATS, individual Translator team members, or affiliated organizations and institutions.

References

1. Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference. Tech. rep., W3C, http://www.w3.org/TR/owl-ref/ (February 2004)
2. Breitling, F.: A standard transformation from xml to rdf via xslt. Astronomische Nachrichten: Astronomical Notes **330**(7), 755–760 (2009)
3. Brickley, D., Guha, R.: Rdf vocabulary description language 1.0: Rdf schema. Tech. rep., W3C Recommendation (2004)
4. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In: Proceedings of the 7th Workshop on Linked Data on the Web (Apr 2014), http://events.linkeddata.org/ldow2014/papers/ldow2014_paper_01.pdf
5. Ermilov, I., Auer, S., Stadler, C.: Csv2rdf: User-driven csv to rdf mass conversion framework. In: Proceedings of the ISEM. vol. 13, pp. 04–06 (2013)
6. Knoblock, C.A., Szekely, P., Ambite, J.L., Goel, A., Gupta, S., Lerman, K., Muslea, M., Taheriyan, M., Mallick, P.: Semi-automatically mapping structured sources into the semantic web. In: Extended Semantic Web Conference. pp. 375–390. Springer (2012)
7. Lange, C.: Krextor–an extensible xml rdf extraction framework
8. 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)
9. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation (January 2008), http://www.w3.org/TR/rdf-sparql-query/, http://www.w3.org/TR/rdf-sparql-query/
10. Wilkinson, M., et al: The FAIR Guiding Principles for scientific data management and stewardship. Scientific Data **3** (2016). https://doi.org/http://doi.org/10.1038/sdata.2016.18
11. Zaveri, A., Dastgheib, S., Wu, C., Whetzel, T., Verborgh, R., Avillach, P., Korodi, G., Terryn, R., Jagodnik, K., Assis, P., Dumontier, M.: smartapi: Towards a more intelligent network of web apis. In: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (eds.) The Semantic Web. pp. 154–169. Springer International Publishing, Cham (2017)
12. Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., Auer, S.: Quality assessment for Linked Data: A survey. Semantic Web Journal (2016)