Knowledge and Information Systems (2022) 64:2985–3015
https://doi.org/10.1007/s10115-022-01729-x

REGULAR PAPER

RDF-Gen: generating RDF triples from big data sources

Georgios M. Santipantakis[1] · Konstantinos I. Kotis[2] · Apostolos Glenis[1] · George A. Vouros[1] · Christos Doulkeridis[1] · Akrivi Vlachou[2]

Received: 22 May 2021 / Revised: 13 July 2022 / Accepted: 16 July 2022 /
Published online: 13 August 2022

Abstract
Transforming disparate and heterogeneous data sources that provide large volumes of data in high velocity into a common form allows integrated and enriched views on data and thus provides further opportunities to advance the effectiveness and accuracy of data analysis and prediction tasks. This paper presents the RDF-Gen approach for transforming data provided by archival and streaming data sources, provided in various formats, into RDF triples, according to a set of ontological specifications. RDF-Gen introduces a generic mechanism which supports the transformation of data efficiently (i.e., with high throughput and low latency), even in cases where the velocity of data presents high peaks, offering facilities for discovering associations between data from different sources, and supporting transformation of modular data sets. This paper presents a parallel implementation of RDF-Gen, also presenting data transformation workflows that allow variations incorporating RDF-Gen instances, adjusting to the needs of data sources, application areas and performance requirements. RDF-Gen is experimentally evaluated against state of the art, in both archival and streaming settings: Experimental results show RDF-Gen efficiency and highlight key contributions.

Keywords Data transformation · Data integration · RDF · Big data

1 Introduction

Providing integrated views of data from heterogeneous and disparate data sources using a common form is a challenging task with ubiquitous applications, which is further intensified with the advent of big data Dong and Srivastava [5]. Several application areas including the Internet of Things (IoT) Phuoc et al. [19], healthcare Venetis and Vassalos [26], mobility Vouros et al. [28], manufacturing Brecher et al. [1], Efthymiou et al. [6] Ocker et al. [17], finance, and social networks, produce data that require some form of data integration in order to construct meaningful representations in a uniform data model. The associated technical challenges are manifold, ranging from supporting different input formats and data modalities,

✉ Georgios M. Santipantakis
gsant@unipi.gr

1 University of Piraeus, Piraeus, Greece

2 University of the Aegean, Mytilene, Greece

to requirements related to large data volumes and stream processing Dell'Aglio et al. [3], Hirzel et al. [9].

An ontology expressed in RDFS or OWL often is used to provide a schema for data representation, also enabling integration of data from different sources, and semantic enrichment of data. Such an ontology can be populated with instances expressed in RDF, with respect to the semantics of the ontological specifications, assuming that data from disparate data sources respect own specifications. Therefore, tools that transform data from heterogeneous data sources to RDF, as a step towards populating an ontology with instances, aim at providing integrated and enriched views on data to advance the effectiveness and accuracy of data analysis and prediction tasks. Taking into consideration the constant increase of volume and data provision velocity of data sources, one may infer the importance of developing flexible and highly efficient methods for transforming data to RDF and—eventually—for enriching data semantically.

In this paper, we present the RDF-Gen approach for transforming data provided by heterogeneous archival and streaming data sources, with variety in data formats, into RDF triples, w.r.t. a set of ontological specifications. RDF-Gen introduces a generic mechanism which supports the transformation of archival and streaming data sources efficiently (i.e., with high throughput and low latency), even in cases where data are presented in high velocity, supporting a wide range of data formats, offering facilities for discovering associations between data from different sources, and supporting transformation of modular data sets. Towards this end, RDF-Gen exploits *triple templates*, which are patterns of triples specifying the RDF representation of data, and which are instantiated with data values from the sources at run time.

The contributions this work makes are as follows:

– It presents RDF-Gen, a novel, efficient and flexible approach for data transformation to RDF w.r.t. an ontology;
– It shows how RDF-Gen can satisfy the requirements for transforming data from archival and streaming data sources, which provide data in various modalities and formats;
– It provides variations of basic workflows incorporating instances of RDF-Gen towards satisfying requirements stemming from variations of data sources and from analysis tasks, thus supporting transformations from independent sources to joined, inter-dependent, and modular data sources;
– It supports to a certain extent the discovery of links between data from disparate sources, providing a generalization of the join mechanism among entities, with common attributes in different sources;
– It demonstrates the efficiency and scalability of RDF-Gen using various real-life data sets, in various settings, also in comparison with state of the art.

It is noteworthy that RDF-Gen has been extensively used to transform a variety of real-life, archival data sets, as well as data from surveillance streaming sources. RDF-Gen has been also used in time-critical big data analytics solutions in aviation and maritime domain Vouros et al. [28], while a user-friendly interface Santipantakis et al. [21] for RDF-Gen is also available.

This work extends the work described in Santipantakis et al. [22] to the following aspects:

– RDF-Gen has been extended to support data transformation to RDF from both archival and streaming sources, and to inherently support close-to-source data processing (e.g., for data cleaning and values' conversion).
– Custom data processing and manipulation functions are integrated in the transformation process.

– A wide range of data formats is supported, while RDF-Gen is made expandable to include new formats when needed.
– A parallel version of RDF-Gen is presented, enabling also data transformations' workflow variations incorporating multiple RDF-Gen instances.
– The computational efficiency of RDF-Gen in terms of high throughput and low data transformation latency is evaluated through extensive experimentation.

In a nutshell, this work makes RDF-Gen a complete solution for data transformation into RDF triples from various data sources and formats, by generalizing the join mechanism among data sources, providing a parallel implementation of the approach for extremely high demanding throughput, presenting a variety of data transformation workflows, and enabling the discovery of links between resources generated from different data sources.

The rest of this paper is structured as follows: Section 2 sets the requirements for data transformation to RDF, while Sect. 3 reviews the related works. Then, in Sect. 4, we present the RDF-Gen approach, and Sect. 5 describes how we address modular data sources. Section 6 presents the parallelized RDF-Gen and basic variations of data transformation workflows of RDF-Gen instances. Section 7 presents the results of a comparative evaluation with state of the art using real-world data sets, and finally, Sect. 8 concludes the paper.

2 Motivation and requirements

Several analysis tasks such as the analysis of moving objects behavior Vouros et al. [27] require the combination of streaming (surveillance, occurring events, etc.) with archival (space configurations, entities characteristics, etc.) data. In addition to that, data are often provided from disparate and heterogeneous sources in different formats and forms. For instance, integration of several sources providing the same data in different formats (e.g., integrating surveillance data from different providers, or weather conditions from different weather forecast services) may provide duplicate records or result in imperfections/noise in the final data. Also, integration of data sources (archival or streaming) may require data value conversions from one measurement or reference system to another. This indicates that functions preparing the data are necessary. Such functions can be also used to generate uniform resource identifiers (URIs) of entities via key values in the data. For example, the registration ID of an aircraft uniquely identifies an aircraft across any data source. In so doing, we can use a function that generates the URI of an aircraft from its registration ID. Additionally, if such a function is reused in more than one data sources, it enables the integration of the data from these sources (assuming that each source provides additional information about the aircraft). Usually, such functions are custom, depending on the data sources exploited and the entities that should be identified in the transformation process.

Thus, preparation and processing of data for transformation in a common, integrated form, is an important task that often takes a considerable amount of time, before any analysis is performed, introducing latency into any analysis pipeline. Addressing these needs, we present challenges that motivated our research and the implementation of RDF-Gen approach.

2.1 Archival data

Several data sets are provided in archival form: These have been collected over a period of time, and they are rarely updated once they have been archived. Archival data sets are often voluminous, and usually they do not share a common format. Most popular file formats used for archival data sets are tabular Comma (or Tab) Separated Values (CSV/TSV), XML, ESRI Shapefile ESRI [7], (Geo)JSON[1], and RDF/XML[2]. Several examples of voluminous data sets in CSV format are publicly available. For instance, CSV surveillance and ESRI shapefile data sets that have been processed by RDF-Gen can be found online[3].

Archival data sets can be modular: Information in these cases is distributed (usually under technical criteria) into more than one files. For example, a modular data set that have been transformed to RDF is the Sector Configuration data set in the aviation domain. A Sector Configuration defines how the Flight Information Regions (FIRs) are divided into 3D airspace volumes (sectors) over a period of time. The segregation of an airspace can change within the day, according to the number of flights expected to be crossing it in different periods. Such data sources usually employ more than one files (not necessarily in the same format), e.g., a file to describe the sectors' geometries, another file to specify the activation schedule of sectors, the flight plans crossing the sectors, locations of fixed waypoints used in the flight plans, etc.

Furthermore, archived data sets may have been produced by combining two or more data sources. For example, this is the case of combining historical surveillance data sources to eliminate "blind spot" areas each one of the sources separately may have. However, the combination of such data sources, in many cases, can generate duplicates. Unidentified duplicates often affect the performance and in some cases the soundness of results of data processing and analysis methods.

Finally, in some cases we may need to combine archived data sets and exploit only the necessary information. For example, joining archived surveillance data in aviation with an aircraft model database supports explaining the flying behavior of some aircraft. E.g., any training or search-and-rescue aircraft follows different moving patterns compared to a commercial aircraft. Furthermore, an aircraft model database can provide information such as wingspan, aircraft height and weight, personnel and passengers, etc., which can be irrelevant to the objectives of transformation, and may not even map to the specifications of the ontology to be populated. This example indicates that a data transformation method has to support the selection and filtering of data to be transformed from a source, providing also flexibility on how data from different data sources join.

2.2 Streaming data

The data transformation of streaming data sources is another challenging task, since performance in terms of processing time, throughput and latency is of great essence. Indeed, the data transformation task will inevitably add latency between records in the original stream and corresponding records in the stream of transformed data. It is crucial that this latency is the smallest possible, especially when the transformed data are exploited in online analytics tasks.

In addition to the above, challenging streams can be velocious or bursty, making real-time data transformation to RDF a cumbersome task. An example of such data set is a satellite surveillance data. This stream imposes strict latency requirements since hundreds of messages per second need to be processed. In addition to that, this stream is bursty, i.e., there are extremely high peaks on the message transmission rate when the system (periodically)

[1] GeoJSON Specification is available online at https://tools.ietf.org/html/rfc7946.

[2] RDF/XML Specification is available online at https://www.w3.org/TR/rdf-syntax-grammar/.

[3] https://zenodo.org/record/2576152.

transmits messages from linked satellites. We further discuss this surveillance data set in Sect. 7.

Similarly to the archival data sets, streams can provide data in various formats and protocols. Streams can be provided using plain TCP/IP sockets to the full-duplex communication channels of Websocket protocol. The data transformation method has to be compatible with most popular protocols. In addition, messages or records in streams can be provided in various formats, ranging from plain comma separated values text to XML and JSON objects. The data transformation method needs to support the most popular formats used in streams and be expandable to include new formats when needed.

2.3 Requirements

Our engagement with data sets such as those mentioned in the examples provided in the previous sections motivated the development of RDF-Gen, satisfying the following major requirements:

- The approach is required to be applicable on both archival and streaming sources, in a conjunctive way (i.e., not merely allowing to switch between these modes).
- The approach is required to demonstrate computational efficiency in terms of high throughput and low data generation latency. It is required to support parallel processing and transformations of data, to cope with extremely high-throughput data sources.
- The approach is required to support a wide range of data formats and it has to be expandable to include new formats when required.
- The approach has to provide facilities for close-to-source data processing (e.g., data cleaning and value conversion), which has to be expandable with custom functions.
- The approach is required to support the discovery of associations between resources of the same or different data sources, thus combining two or more data sources, using properties specified in a given ontology.
- The method is required to support modular data sources, i.e., it has to provide a generic join operator that enables data records originating from different sources (and possibly under different format), to be joined and transformed as a single record.

In addition to the above list, we expect that the approach is made easily integrated into RDF processing and analysis workflows via employing a user-intuitive configuration, using specifications that are close to the standards (RDF, SPARQL) used in these settings.

The proposed RDF-Gen data transformation approach is based on three assumptions, as follows:

The first assumption dictates that data are either structured, or there is a method that processes raw data in the source and produces structured records. This assumption allows us to focus on how the data can be efficiently transformed to RDF triples, without getting involved on issues regarding how the data can be extracted from raw data. Apparently, any sophisticated method can be employed prior to data transformation, such that data are made structured and finally transformed.

The second assumption requires that the data in the sources can be formally described by an ontology at the appropriate level of expressiveness, also with respect to the desired computational properties. In other words, we assume that the data can be transformed according to the specifications of a common vocabulary or ontology, expressed in RDFS up to OWL2. In some cases, the relations and concepts of an ontology can be discovered from the data. However, automatic ontology generation is an open research field and beyond the scope of this work.

Finally, we assume that any data conversions and custom functions needed during transformation are a priori known and they will not be modified at run time. For instance, temperature values in some data source must be converted from Fahrenheit to Celsius and the corresponding conversion function is implemented before data transformation is initialized.

We conjecture that a data transformation method is considered adequately generic and efficient, if it satisfies the requirements, under the assumptions specified in this section.

3 Related work

This section presents a number of different state-of-the-art approaches related to the field of data transformation/RDF generation, along with and their limitations. A criteria-based comparison of these approaches against RDF-Gen is also presented (Table 1).

R2RML[4] is a well-known mapping language for expressing mappings from relational databases to RDF graphs, which became W3C recommendation in 2012. For each logical table, a triples map is defined in R2RML, which completes the transformation of data to RDF. RML, Dimou et al. [4], is a mapping language based on R2RML. It follows an extensible RDF generation approach, while supporting the definition of graph templates (called mappings in the context of RML) for multiple heterogeneous sources. RML does not require storing files into memory, making it appropriate for processing datasets too big for the processor's memory. In relation to the ability to support close-to-the-sources data processing functionality, RML supports the integration of custom data processing functions using a script language, namely FunUL, Junior et al. [10], or FnO, Meester et al. [15]. However, the required functions should in-principle be re-defined in every mappings specification, increasing the time needed for the overall RDF generation process. On the positive side, this approach satisfies the requirements for supporting a wide range of data formats and provides facilities for close-to-the-sources data processing. On the other hand, experimental results by Santipantakis et al. [22] have shown that R2RML has performance limitations, while it is only applicable on archival sources.

D2RML, Chortaras and Stamou [2], is a transformation-oriented RDF mapping language. It draws significantly from R2RML and RML and follows the same strategy for defining mappings for RDF triples, composed by a mapping for the subject and several predicate-object mappings. D2RML is based on an abstract underlying data model, allows the orchestrated retrieval of data from diverse information sources, data transformation through relevant web services, data filtering and manipulation using simple operations, and mapping to RDF triples. It covers the requirements for close to the sources data processing and is able to specify more complex joins and filtering. It supports a variety of data formats, and features close to the sources processing functionality. However, it does not inherently support streaming data sources or parallel processing.

SPARQL-Generate, Lefrançois et al. [14], enables transformations and the generation of RDF from: (i) any RDF dataset and (ii) any set of data sources in arbitrary formats. SPARQL-Generate has been designed as an extension of SPARQL 1.1, so it can provably be implemented on top on any existing SPARQL engine. SPARQL-Generate supports easy validation of generated output, since the mappings are provided as an extension of SPARQL. SPARQL-Generate also supports a variety of data formats and allows reusability of the mappings across data sources that share common features. The latest version of SPARQL-Generate can process Websocket streams and HTTP GET requests. On the other hand, this

[4] https://www.w3.org/TR/r2rml/.

method does not support parallel processing, join operators, and the use of custom functions is not straightforward in the general case, since it requires the SPARQL engine to be re-compiled.

KR2RML, Slepicka et al. [25], is another interpretation of R2RML, integrated in the open-source tool, Karma Knoblock et al. [12], paired with a source-agnostic R2RML processor that supports data cleaning and transformation. It supports easy addition of new input/output formats, while supporting efficient cleaning, transformation, and generation of voluminous RDF data sets. This approach has been embedded in Apache Hadoop and Apache Storm for scalable batch and streaming processing, respectively. The data manipulation functions are specified via a built-in editor and enable data processing close to the sources. However, this approach does not support joins between data sources, and custom functions are stored within the mappings which may hinder the modification of mappings in some cases.

RMLProcessor, Dimou et al. [4], Junior et al. [11], is an RML-based approach supported by a proof-of-concept system that extends RML's vocabulary and engine by introducing constructs for describing functions, function calls and parameter bindings. Although functions execute simple string transformations, they can be generic and capable of more complex data transformations. Since the approach focuses on functions, these may provide functionality for any data format and can be reused in the mappings. To the best of our knowledge, this method inherently supports archival data sources in various data formats, and data processing close to the sources. However, it does not demonstrate computationally efficient data transformation, join operators and processing of streaming data.

The DataLift project, Scharffe et al. [23], aims at providing a framework and an integrated platform for publishing datasets on the Linked Data Cloud. DataLift can parse several types of data, e.g., CSV, RDF, XML and ESRI shapefiles. Although it can parse CSV files of captured streaming data, it cannot support streaming data sources directly. In addition to that, DataLift does not support the integration of custom data processing functions.

RocketRML, Simsek et al. [24], is an implementation of an RML mapper in NodeJS. It performs better than legacy RML, mostly because it does not depend on the data access libraries used in RML. Performance tests have shown that RocketRML has great potential to perform processing-intensive mapping tasks, but it also comes with some limitations: it currently does not support joins and input formats other than XML and JSON of archival sources.

GeoTriples, Kyzirakos et al. [13], is another approach based on RML, implementing two discrete phases: automatic generation of R2RML mappings and transformation to RDF. As it relies on RML, it inherits its performance limitations. In addition to that, when functions are needed to be incorporated in the mappings, user intervention is required. Under the scope of the specified requirements, GeoTriples supports various data formats and parallel processing (via the GeoTriples-Hadoop implementation). It does not, however, support joins and close to the sources data processing, on both archival and streaming sources.

Haesendonck et al. [8] recognize that most data conversion approaches ingest the data into memory, before any processing is done. This strategy hinders the parallelization of data conversion, and it is restricted to data sets that fit in memory. RMLStreamer, built on top of Apache Flink, aims to parallelize and distribute the ingestion tasks over multiple nodes, to scale with data volume. Also, it relies on Flink's memory management for processing large data sets. The ingestion phase is implemented as an input operator that consumes CSV, JSON and XML data formats. Only for the case of CSV format, a parallel input operator can be used to parallelize the process over multiple nodes. In case of JSON and XML formats, a sequential input generator is used. This method does not support other data formats, such as ESRI shapefile, and the requirement for a wide range data formats cannot be considered

Table 1 Comparison of existing approaches for RDF generation

	Archival and streaming sources	Computationally efficient	Wide range of data formats	Close-to-source data processing	Link discovery	Supports modular data sets (joins)
R2RML Dimou et al. [4]	✗	✗	✓	✓	✗	✗
D2RML Chortaras and Stamou [2]	✗	✗	✓	✓	✗	✓
SPARQL-Generate Lefrançois et al. [14]	✓	✗	✓	✗	✗	✗
KR2RML Slepicka et al. [25]	✓	✓	✗	✓	✗	✗
RMLProcessor Dimou et al. [4], Junior et al. [11]	✗	✗	✓	✓	✗	✗
Datalift Scharffe et al. [23]	✗	✗	✓	✗	✗	✗
RocketRML Simsek et al. [24]	✗	✗	✗	✓	✗	✗
GeoTriples Kyzirakos et al. [13]	✗	✓	✓	✗	✗	✗
RMLStreamer Haesendonck et al. [8]	✓	✓	✗	✗	✗	✗
RDF-Gen Santipantakis et al. [22]	✓	✓	✓	✓	✓	✓

fulfilled. Also, there is no support for custom functions and join operators. On the positive side, the method can process archival and streaming data sources.

Even though we acknowledge the existence of additional related works (e.g., OntoRefine tool of GraphDB[5], Sponger tool of Virtuoso[6]), however we omit them in this review, since (a) they are dependent on specific triple-stores, and (b) they cannot be evaluated in a standalone mode so as to be directly compared to the RDF-Gen.

Concluding this section, we summarize the comparison of related works in Table 1. The majority of methods presented already support a wide variety of data formats, varying from plain text CSV files, to geospatial vector data sets (such as ESRI shapefiles). Furthermore, only half of the related works allow custom-made functions for filtering and converting the data, close to the source. This is an important limitation, since values in data often need to be processed and converted to specific formats, especially when the RDF generation employs more than one data sources. Another important issue is that none of the related works supports link discovery, i.e., the detection of relations between resources in the same or different data sources. To the best of our knowledge, the majority of the presented works does not support

[5] http://graphdb.ontotext.com/documentation/free/loading-data-using-ontorefine.html.

[6] http://vos.openlinksw.com/owiki/wiki/VOS/VirtSponger.

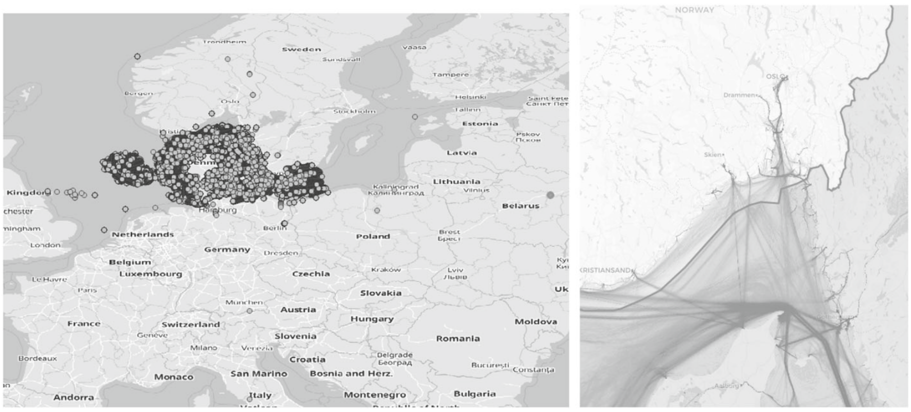


Fig. 1 The distribution of positions reported in the DMA (left) and NMA (right) surveillance data for one day

joins prior to the data transformation, which is another important limitation. In contrast, RDF-Gen has been designed to satisfy all the requirements specified in this section.

4 RDF-Gen

A data source in the general case cannot be processed and transformed to RDF triples with minimum effort. Often, a time-demanding preparation phase is necessary, for converting and processing values, filtering data, eliminating outliers and configuring the transformation process.

As a typical example, let us consider a surveillance data source of moving objects in a maritime region, such as the data provided by Danish Maritime Authority (DMA). In such data sets, we often need to consider the conversion of values of some of their attributes such as speed, positioning, or even timestamp, to certain units of speed, Coordination Reference System (CRS) and timezone, to enable interplay with other data sources. For instance, if the position of moving objects is reported in a different CRS than the one used for the geometries of regions of interest (e.g., protected areas), the computation of topological relations is hindered. In addition to that, outliers can be detected by their reported positions if the physical limitations of the surveillance sensors/setup are known. Such knowledge allows the definition of the area of coverage, and any moving object reported outside of that area can be safely marked as an outlier. Figure 1 illustrates the distribution of positions reported for one day, where some outliers can be easily identified (positions beyond the range of coverage and on land can be safely ignored).

Assuming that we want to extend the area covered by our data towards Norway, we need to adapt a surveillance source from the corresponding Norwegian authority (NMA) (Fig. 1). We need to apply the same functions on this data set to eliminate the outliers, but also when combining these two data sets, we have to apply one or more deduplication functions that will eliminate redundant data found in the common area covered by the two sources.

From this example, it follows that a set of value conversion and filtering functions that are domain or source specific need to be applied before any data transformation is considered. These functions often need to be slightly different (e.g., DMA has different area of coverage than MNA and the filtering functions need to be modified accordingly), which can become a

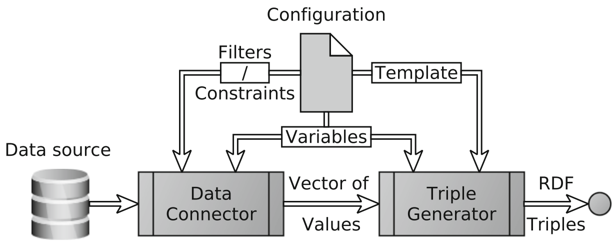


Fig. 2 The RDF-Gen architecture overview

time-demanding process, when more than two data sets are involved. In addition to that, this process will generate new, filtered data sets, that are stored locally, increasing the storage requirements of the approach. The RDF-Gen approach employs custom functions for such cases, available in an extensible library. These functions can be reused across similar data sources (as in the example above), to filter, convert values or eliminate duplicates (given a set of rules). This allows repetitive computations, such as unit conversion or elimination of outliers, to be done on the fly, while retrieving the data for the transformation process. The RDF-Gen approach achieves the data transformation to RDF triples by means of *triple templates*, which are similar to SPARQL graph patterns, and contains *template variables* that are associated with data from the source. Any data conversion, filtering or processing is applied via custom functions, on the template variables.

For instance, continuing the above example, a function that takes as arguments the template variables for the reported position (latitude and longitude) of the moving object and the geometry of the covered area (constant value) can be used to filter obvious outliers in the data. The same function can be applied on both surveillance data sets in this example, using different geometry arguments. Also, a function that takes as arguments the variables for Maritime Mobile Service Identity (MMSI) and vessel's name can be used in both data sets to generate the URI of a vessel. In fact, this function also guarantees that the same URI will be given for the same vessel once appeared in both data sets.

The syntax and semantics of RDF-Gen templates are discussed in the next paragraphs.

The variables used in the template are specified in the *vector of variables*. Each record extracted from a data source is represented as a vector of values, in correspondence to the vector of variables. Thus, each data record leads to a set of assignments: Each variable in the vector of variables is assigned the corresponding value in the vector of values. Values of variables instantiate the triple template, to generate the corresponding RDF triples for each record in the data source.

It must be noted that the triple template, the vector of variables, and the filters applied during the data transformation, as well as the specification of the data source type, are specific to the data source. Different data sources may require different triple templates, even though data are transformed under the same ontology. The specification of these settings comprises the RDF-Gen *Configuration*.

Figure 2 provides an overview of the RDF-Gen architecture, comprising two main modules: the *data connector* and the *triple generator*. Specifically, the data connector is responsible for fetching and filtering data from the source, based on user-defined and source-dependent constraints, converting the input data into a sequence of records. Then, the triple generator assigns the values to template variables and instantiates the triple template, to generate the corresponding RDF triples.

Both the data connector and the triple generator access the vector of variables. The data connector is responsible to connect to the data source and seamlessly provide (at the same transmission rate of the source) with data the triple generator. Its role is to decouple the triple generator from any format-specific constraints, as it provides the data in a unified form, regardless of their original format. For example, data sources providing data as comma separated values, binary (e.g., Automatic Identification System—AIS, ESRI shapefiles, etc.), XML or any other format, are processed by the data connector in such a way that the triple generator always receives a vector of values, regardless of the original format at the source. Currently, a wide range of data connectors is available for several popular formats, such as CSV, XML, JSON, ESRI shapefile, GRIB, and data access through web sockets and ODBC database connections. The data connector is also responsible to evaluate whether the number of values matches the number of variables for each record (any mismatch indicates an incorrect configuration), while the triple generator checks if the variables in the triple template appear in the vector of variables. The triple template can use any subset of the variables in the vector of variables.

The RDF triples that are generated by RDF-Gen approach are not randomly generated, but they have a meaning under some ontology. The ontology is an obvious requirement of all the data transformation to RDF triples approaches. Any specification dictating how the data should be converted to RDF triples, should be derived from (computationally) or be consistent to an ontology describing the data. RDF-Gen approach is not an exception; thus, the triple template is designed according to an ontology which can feature any level of expressiveness in the decidable profiles of OWL2 and RDFS.

The triple generator assigns the values received from data connector to the corresponding variables and calls any functions applied in the given template. The result will be a set of RDF triples (possibly empty, as in the case of a detected outlier), that represent the data provided at the source and the template applied. The RDF triples can then be used to populate the ontology considered for building the template.

4.1 Template semantics

The records generated from a data connector are consumed by a triple generator, which is responsible to convert the provided records into RDF triples w.r.t. a given ontology. A triple generator is configured by: (a) a vector of variables V, (b) an RDF graph template G, and (c) a set of functions. As already specified, variables in V correspond to the attributes of a record.

Formally, let **I**, **B**, and **L** the pairwise disjoint infinite sets of URIs, blank nodes and literals, respectively. A triple $(s, p, o) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ is called an RDF triple, where s is the subject, p is the predicate and o is the object of the triple. **V** is the infinite set of variables that is disjoint from the above sets.

A variable $?x \in \mathbf{V}$ is said to be bound to a value q in $\mathbf{I} \cup \mathbf{L}$ if, and only if, $\text{bound}(?x) = q$. Let **F** be an infinite set of function names disjoint with **I**, **B**, **L**, and **V**. We distinguish the following categories of functions **F** that can be used in triple templates:

1. The *resources' functions* $f \in \mathbf{F}$ s.t. $s = f(A)$ (or $o = f(A)$), $s \in \mathbf{I}$ (resp. $o \in \mathbf{I} \cup \mathbf{L}$), $A \subseteq \mathbf{V}$ which can be used as subject (resp. objects) in triples,
2. The *triples' functions* $f \in \mathbf{F}$ s.t. $\{(s, p, o)\} = f(A),\ A \subseteq \mathbf{V}$ which can be used for the construction of a (possibly empty) set of triples from the bounded variables of A.

Definition 4.1 A *triple template* is a triple $\tau \in (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V} \cup \mathbf{F}) \times (\mathbf{I} \cup \mathbf{F}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V} \cup \mathbf{F})$: The subject, predicate or object in the triple can be a variable or a function.

Table 2 Sample data of DMA and vessel characteristics data sets

Surveillance data

Timestamp	Type of mobile	MMSI	Latitude	Longitude	...	Navigational status	SOG	COG	IMO
01/09/18 12:00 AM	Class A	219001682	57.158983	8.65872	...	Engaged in fishing	8.2	46.6	

Vessel characteristics

MMSI	IMO	Name	Home Port	Country	Type	Breadth	Length	...
219001682		HM196 FRU FJORD	Viborg, DK terrestrial	Denmark	Fishing	5m	20m	...

Definition 4.2 A *graph template* is defined as a set of:

1. $\tau \in (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V} \cup \mathbf{F}) \times (\mathbf{I} \cup \mathbf{F}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V} \cup \mathbf{F})$
2. $T = f(A)$, where $f \in \mathbf{F}$ and $A \subseteq \mathbf{V}$, s.t. T is a set of triples constructed from a triples' function f given the bounded variables in A.

A sample of DMA data set and the corresponding record in Vessels Characteristics data set are reported in Table 2. Figure 3 illustrates the graph to be generated by these data, where ovals represent resources, parallelograms represent literals and rounded rectangles represent concepts of the ontology. White colored elements in the figure have been transformed from Vessel Characteristics data set, while gray colored elements have been transformed from the DMA surveillance data set. The triple templates that achieve this transformation in this example are the following:

DMA surveillance Triple Template:

```
getVessel(?mmsi) :hasTrajectory getTrajectory(?mmsi).
getTrajectory(?mmsi) :hasPart makeSemanticNode(?mmsi,?time,?lon,?lat) .
makeSemanticNode(?mmsi,?time,?lon,?lat) :hasGeometry makeGeometry(?lat,?lon) ;
   :hasTemporalFeature makeTimeInstantByDateTime(?time) ;
   :hasSpeedOverGround asString(?sog) ; :hasCourseOverGround asString(?cog) ;
   :hasStatus asString(?navStat) .
makeGeometry(?lat,?lon) a :Geometry ; :hasWKT makePointLatLon(?lat,?lon) ;
   :hasLatitude ?lat ; :hasLongitude ?lon .
makeTimeInstantByDateTime(?time) a :TimeInterval ; :hasTimeStart asDateTime(?time) ;
   :hasTimeEnd asDateTime(?time).
```

Vessel Characteristics Triple Template:

```
getVessel(?mmsi) a getVesselType(?type) ;
   :hasCode asString(?mmsi) ; :hasLength asString(?len) ;
   :hasBreadth asString(?width) ; :hasName asString(?name) ;
   :registeredAt getCountry(?country).
```

In this example, we observe that functions have been used for either generating a URI from a value in the data (such as the function getVessel(?mmsi) which returns the URI for the vessel assigned to the MMSI code), or for formatting the values as literals (such as the function asString(?name) which ensures that the literal will be formatted as a string). The function getVesselType(?type) employs a lookup table built from the given ontology and returns the vessel type (URI of the class in the ontology) that corresponds to the bound value to ?type variable. The lookup table is built before the data transformation process is initialized. We observe that the use of the same getVessel() function in both templates, simplifies the integration of the two data sets, since it guarantees that exactly the same URI will be built for the same ?mmsi values; thus, no additional effort on building owl:sameAs statements is needed.

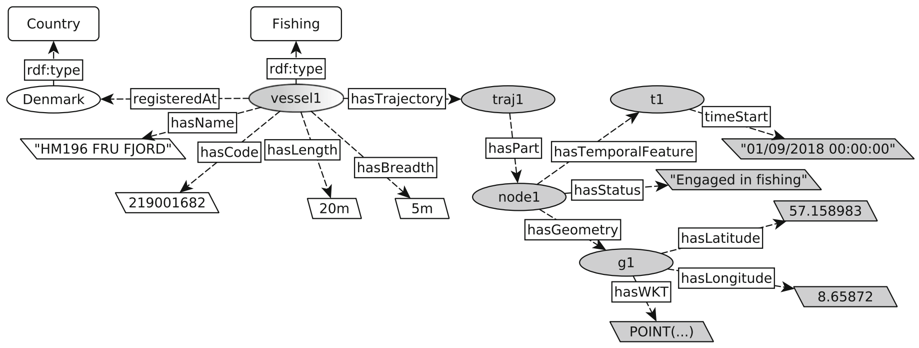


Fig. 3 Example of transformed DMA data into RDF triples. Rounded rectangles represent concepts, ovals represent resources and parallelograms denote literals

4.2 Data connector

The RDF-Gen configuration specifies how the data connector connects to a set of input data sources, and how the data are provided to the triple generator. Distinguishing the data access from the data conversion process allows RDF-Gen to be applied to any new data source (streaming or archival, providing data in any known format). Currently, RDF-Gen provides data connectors for most common data formats such as XML, CSV, JSON, ESRI shapefiles, GRIB binary files, databases, web sockets and SPARQL endpoints. New data connectors can be built either extending the provided connectors, or implementing the data connector interface, which practically consists of three simple methods: connect(), hasNext() and next(). From a more technical point of view, the data connector implements an iterator over the records of data provided in the source. Following this record-by-record access model, data connectors treat both streaming and archival data sources in a uniform way. Essentially any data source is considered to be a “stream” of records that needs to be processed with minimal latency. Since operations are performed on individual records, the memory footprint of the triples generation process is very low, rendering the RDF-Gen approach to be highly efficient and scalable. Finally, the triple generator instantiates the variables in the triple template with their bound values and generates the RDF triples from the corresponding data.

4.3 Configuration and template syntax

The configuration setup is provided in an RDF/XML file[7], where (a) the path to the data source, (b) the type of data connector to be used, (c) the path to the triple template file, and (d) the path to the output file, as well as (e) any attributes that need to be configured, are specified.

Often, we may be interested only to some part of the data provided by some data source. For example, given a weather reporting data source that provides a stream of JSON objects, we may need only a subset of the attributes (e.g., temperature and wind components, but not precipitation) of each object. We can select the attributes that will be considered in the data transformation process using the term `inputVariables`. Additionally, the term `xPathEntity` allows the specification of an attribute that uniquely identifies each entity

[7] The predefined terms for the configuration file are in the namespace http://www.datacron-project.eu/RDFGen_conf#.

```
<xml>
  <document>
    <aircraft model="A320">
      <manifacturer>AIRBUS</manifacturer>
    <OptimumSpeed>233.56 m/s</OptimumSpeed>
    <reference>https://doc8643.com/aircraft/A320</reference>
  </aircraft>
  <aircraft id="B701">
    <manifacturer>BOEING</manifacturer>
    <OptimumSpeed>249 m/s</OptimumSpeed>
    <reference>https://doc8643.com/aircraft/B701</reference>
  </aircraft>
  </document>
</xml>
```

Fig. 4 An aircraft models XML data source example

```
...
  <dcf:xPathEntity>/home/example/aircrafts.xml/xml/document/aircraft/ </dcf:xPathEntity>
  <dcf:inputVariables>@model,/manifacturer,
      /OptimumSpeed, /reference
      </dcf:inputVariables>
  <dcf:TemplateVariables>?model,?manifacturer, ?OptSpeed,?ref
  </dcf:TemplateVariables>
```

Fig. 5 Fragment of the configuration file for the aircraft models XML example

```
makeURI(?model) a :AircraftModel ; :manifacturedBy ?manifacturer ;
  :speed ?OptSpeed ; rdfs:seeAlso ?ref.
```

RDF triples:

```
:A320 a :AircraftModel ; :manifacturedBy "AIRBUS" ;
  :speed "233.56 m/s" ;
  rdfs:seeAlso "https://doc8643.com/aircraft/A320".
:B701 a :AircraftModel ; :manifacturedBy "BOEING" ;
  :speed "249 m/s" ;
  rdfs:seeAlso "https://doc8643.com/aircraft/B701".
```

Fig. 6 Example of a triple template for the aircraft models XML data set and the corresponding RDF triples generated from the example data

referenced in the source. The configuration setting specifies a mapping between attributes in `inputVariables` and the variables specified by the term `TemplateVariables`. Thus, for XML, JSON and ESRI shapefile data sources, the data connector will iterate over the values of the attribute specified in `xPathEntity` and assign the values of attributes specified in `inputVariables` to the corresponding variables in `TemplateVariables`.

The term `xPathEntity` applies on XML, JSON and ESRI shapefile data sources. For XML and JSON sources, it specifies the path identifying each entity. Similarly, for shapefile sources, it specifies the attribute providing the geometry (e.g. `the_geom`). For tabular data sources (e.g., CSV, TSV, etc.), the `xPathEntity` term has no use since data are provided as records. In this case, the term `inputVariables` contains the index of each column starting from 0 for the first column, 1 for second column, etc. For example, a configuration `4,7,8` for `inputVariables` will construct a record per line in the tabular data using only the fifth, eighth and ninth columns of the source.

Example 4.1 Given the XML file of Fig. 4 at `/home/examples/aircrafts.xml`, the mapping in the configuration file will be as in Fig. 5. Specifically, the iterator will iterate in `/xml/document/aircraft` entries, and it will construct the records using the attributes `@model`, `/manifacturer`, `/OptimumSpeed`, `/reference`. The `TemplateVariables` specify that the first place in the record corresponds to the `?model` variable in the template, the second place corresponds to `?manifacturer`, the third place corresponds to `?OptSpeed`, and the fourth place corresponds to `?ref` variable in the template of Fig. 6. Please notice that definition of prefixes is not required in the template, but the prefixes must be provided in a separate file when the RDF-Gen output populates the ontology. In this example, we use the prefixes: "rdfs:<http://www.w3.org/2000/01/rdf-schema#>" and ": <http://example.org/mod#>".

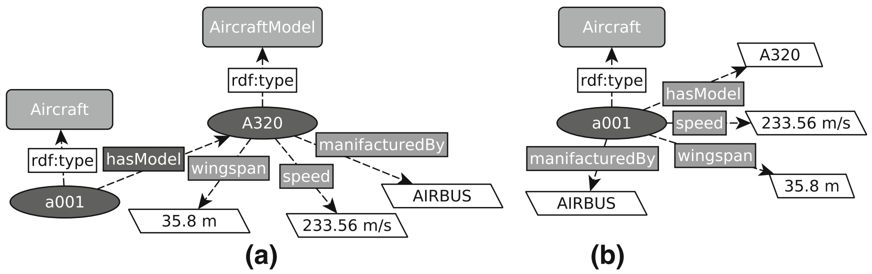
5 RDF-Gen on modular data sources

An early version of RDF-Gen Santipantakis et al. [22] provided a mapping only between XML files. Here, we present a generalization of the mapping configuration which allows joins between modular data sets of any format. This operation can be applied between two or more data sources (or even between data sets in a single data source). As an overview, the join operator in the presented approach concatenates columns (or attributes) of different sources, as if they were in the same tabular form. Thus, data are processed as if these columns are of the same record, although they originate from different sources.

We recall that the generated triples have to be consistent with the ontology according to which the triple template has been constructed. Thus, the triple template and the ontology are mutually dependent, i.e., any modification in the one affects the other. For example, renaming a class of the ontology requires the corresponding renaming in the triple templates that generate instances for that class. Similarly, modifying a triple template requires also modifications on the ontology, so that the generated triples will be consistent with the ontology. Since the triple template is also affected by the data available in the data source, often the data confine the options available on how they can be used w.r.t the ontological specifications. A useful tool that provides more flexibility on engineering the triple template and the related ontology is the join operator. To motivate the need of the join operator, we briefly discuss the data transformation options available on modular data sources.

A naive data transformation approach would transform every data source available, independently. This approach implies that the ontology considered is purely driven by the existing data sources, which confines the options of data engineers to conceptualize the domain w.r.t. the data sets at hand, while it may provide limitations to add new data sources in the process: Often, in the presence of such changes, data engineers cannot modify the ontology to fully comply with the data sets at hand.

As a simple example, let us assume the transformation of two data sets: an aircraft database and an aircraft models archive under an ontology describing aircraft. A naive approach would populate the ontology with aircraft instances and with model instances independently, requiring an ontology property to associate each aircraft to a model. This indicates that following such a naive approach, data may require changes on the ontology (which is not always desirable or possible) in cases where some classes or properties may not exist. In addition to the above, the given ontology may not be a perfectly detailed conceptualization of aircraft models and may focus on aircraft, therefore some model properties (e.g., wing span, ceiling altitude, top speed, fuel capacity, etc.) are directly associated with aircraft instead of the model. In this case, the naive approach fails, since the aircraft models archive cannot be used to populate the ontology independently from the aircraft source. The naive approach will



[h]

Fig. 7 Example of data transformation to RDF triples from (**A**) independent data processing, (**B**) joined data processing

generate a new resource for each aircraft model, and then aircraft entities must be associated with aircraft models, which is different from the intended specifications in this case. Figure 7 illustrates the triples generated by the naive RDF transformation of data sources for aircraft and aircraft models (case (A) in Fig. 7), and the triples generated from an approach that treats data sources jointly (case (B) in Fig. 7). Hence, depending on the ontology specifications, the independence of RDF transformations may generate longer property paths (e.g., in case (A) in Fig. 7, compared to case (B)), or it may enforce ontology specifications that are considered unnecessary (e.g., specification of resources for aircraft models in case (B) of Fig. 7).

5.1 Syntax and semantics of mappings

Extending RDF-Gen to support more data sources, we provide a generalization of mappings to enable joins of data sources of different formats. It is also possible that a single data source comprises a set of individual files, where join operations also are necessary.

The mappings established between data sources (or files in a single data source) enable the computation of the join w.r.t. the specified attributes, in such a way that any combination of sources and formats (e.g., CSV, JSON, XML, Shapefile, etc.) can be combined and processed by the data connector as a single source.

From a more technical point of view, this implementation applies a "hash-join" algorithm, on attributes that uniquely identify the records in each data set. Specifically, a hash table from the first data set is built by hashing the key attribute of the data set. Next, the algorithm scans the second data set and joins the records with matching attributes. The method keeps track of the attributes that have been joined and appends the result set with records that have not been joined (records of the first data set with empty values for attributes of the second data set, and vice versa). The join operation is performed at the data connector, which provides the joined records to the triple generator, as if they were originating from a single source.

The current version allows mappings to be established only between archived data sources. However, this limitation can be overridden for cases of streaming sources, where sliding time windows of predefined length can be considered. The mappings extend the configuration syntax already presented, with the attributes *mapping* and *same*, where *mapping* defines a mapping between variables specified by *same*.

Figure 8 illustrates a mapping between a CSV describing aircraft ownership and identification, and a JSON data set describing aircraft models and their characteristics. The mapping configures an equi-join operation on the attributes `?mdl` from the first data set and `?id` from the second data set. The process projects 4 attributes from the first data set and 5 attributes

```
...
<dcf:DataSource rdf:about="http://datacron-project.eu/mappings#source1">
<dcf:connector>csv</dcf:connector>
<dcf:InputPaths>0,1,2,3</dcf:InputPaths>
<dcf:OutputVariables>?ICAO,?name,?owner,?mdl</dcf:OutputVariables>
<dcf:delimiter>;</dcf:delimiter>
<dcf:file>/example/aircraftDatabase.csv</dcf:file>
</dcf:DataSource>
<dcf:DataSource rdf:about="http://datacron-project.eu/mappings#source2">
<dcf:connector>json</dcf:connector>
<dcf:InputPaths>0,1,2,3,4</dcf:InputPaths>
<dcf:OutputVariables>?id,?wingSpan,?fuelCapacity,?ceilingAltitude,?seeAlsoURL
   </dcf:OutputVariables>
<dcf:file>/sample/AircraftModels.json</dcf:file>
</dcf:DataSource>
...
<dcf:Mapping rdf:about="http://datacron-project.eu/mappings#mapping1">
<dcf:same>?mdl,?id</dcf:same>
</dcf:Mapping>
...
```

Fig. 8 Fragment of a mapping file for combining CSV with JSON

from the second. The joined records contain 8 attributes (`?mdl` and `?id` values are merged into one attribute), which are assigned with values when `?mdl` and `?id` match, otherwise have empty values. For example, the records about aircraft with unidentified model (i.e., `?mdl` does not match with any of the `?id` values) will have empty values for the attributes wingSpan, fuelCapacity, ceilingAltitude and seeAlsoURL.

Formally, given a set of data sources $D = \{d_1, d_2, \ldots, d_n\}$, we assume a mapping function $R = \mu_f(d_i, e)$, which for each entry e in a data source d_i with values of attributes $(e.a_1, \ldots, e.a_j, \ldots, e.a_k)$ constructs a record R. Reusing this functionality, we can apply equi-join operations in the set of data sources in D. In this case, the mapping configuration specifies the attributes of sources to be used for the computation of full joins s.t. $R = \mu_{fi}(d_i, e_i) ⟗ \mu_{fj}(d_j, e_j)$, where e_i, e_j have common attributes that can be used as unique identifiers of entries in d_i, d_j, respectively.

5.2 Parallel RDF-Gen

This section describes the parallel version of RDF-Gen using the Apache Flink[8] platform.

Given a configuration for data transformation to RDF, the system first computes any joins specified for the data sources. This process is done only once, and the computation of joins is considered as a preprocessing task on the data set preparation.

Each data connector—for each data source type supported—has been implemented as a function that connects to the data source according to the configuration and provides the data in a stream. The triple generator has been implemented as an "asynchronous function" which is based on Flink's Asynch I/O API and allows asynchronous data access. This option enables each worker in the platform to process independently from the others the retrieved records.

In case the triple generator transforms the data in a naive way, using a map function, this implies synchronous interaction, i.e., the map function waits until all the functions in the triple template return values. In such a case, the majority of the time required by functions

[8] https://flink.apache.org/.

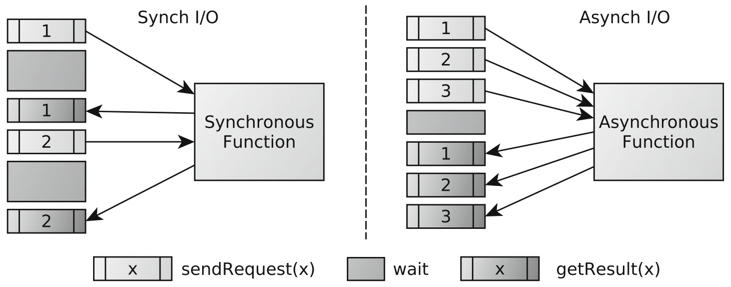


Fig. 9 Illustration of synchronous versus asynchronous functions

is idle time, waiting for responses. Asynchronous interaction, on the other hand, means that a single parallel triple generator instance can handle many records concurrently and receive the responses from functions in an asynchronous way. Therefore, the waiting time can be overlaid with sending other requests and receiving responses. This can improve in most cases the streaming throughput as illustrated in Fig. 9. The triple generator has to process each vector of values received using the triple template and evaluating any custom functions in it. Therefore, the triple generator implements this asynchronous interaction for consecutive records of the data source, given that a template may use any number of functions whose complexity can be arbitrary to the input size. The data connector is not implemented as asynchronous function, since the transformation of a record into the vector of values has low complexity and very low processing time.

Finally, it must be mentioned that the records of the input data source are partitioned randomly according to a uniform distribution, creating equal load per partition and worker. Since join operations are applied at the preprocessing phase, each record needs to be provided to exactly one worker.

6 Flexibility of RDF-Gen

Data transformation to RDF triples usually involves more than one data sources. Often, the method (or methods) employed for transforming the data cannot affect how the transformed data are to be combined. The method transforms each data source to a set of RDF triples (i.e., an RDF graph) in an independent way, and the data engineer has to connect the constructed RDF graphs when the transformation is completed. In this section, we highlight the flexibility provided by RDF-Gen on building workflows for transforming data to RDF triples from various sources in conjunctive and intertwined ways. Workflows illustrated in Fig. 10 as well as the Link Discovery abilities supported provide a wide range of options for transforming multiple data sources to RDF triples.

6.1 Variations of transformation workflows

The workflow illustrated in Fig. 10a implements the naive transformation on any number of data sources into RDF triples. This achieves high throughput, given that sufficient storage and computational resources are provided. Please note that since all triple generator instances access the same custom functions, this ensures that RDF triples generated will be interlinked, as needed. For example, the transformation of an airport database (e.g., reporting location,

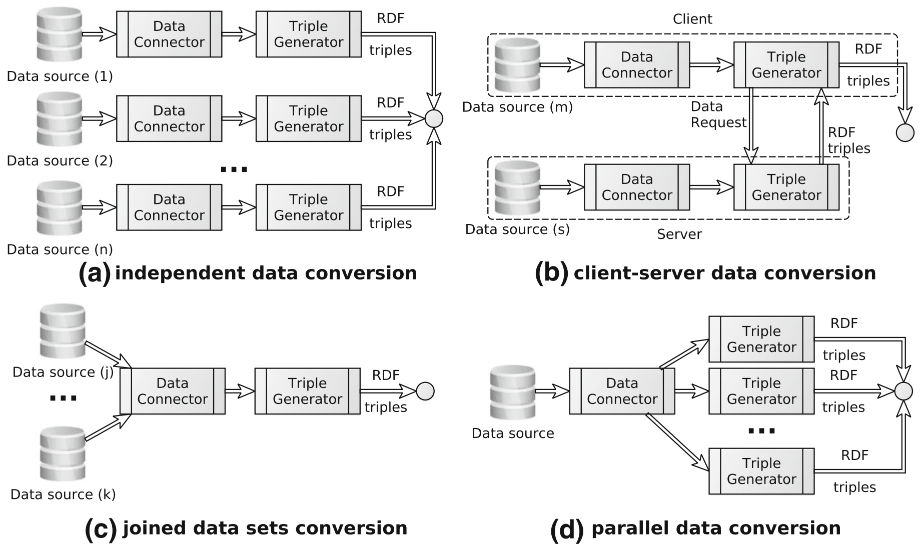


Fig. 10 Transformations' workflow variations

name, facilities, etc.) where each airport is identified by its International Civil Aviation Organization (ICAO) code, can employ a function which constructs the airport's URI from the given ICAO code. Assuming another data transformation process on a data source providing flight plans (specifying departure and destination airport, aircraft identification, estimated take off time, etc.), we can guarantee that reusing the same function to construct the URIs of departure and destination airports using their ICAO codes will result to an integrated RDF graph, when the output of these (independently) transformed data sources populates the ontology.

The workflow (b) exploits the functionality of RDF-Gen to be initialized as a server, as discussed in Santipantakis et al. [22]. Specifically, an RDF-Gen instance (server) can access a data source and generate (on demand) only those RDF triples that will be requested by remote RDF-Gen instances (clients). In this workflow, two or more RDF-Gen instances can establish a "client–server" connection through REST API, to construct triples on the servers side, on-demand from the clients. Each RDF-Gen instance is initialized by its own triple template. A client RDF-Gen instance requests data from a server, using assignments to the variables specified in the configuration of the latter. The server RDF-Gen instance uses these assignments to request zero or more records from its associated data connector. Then, it transforms the retrieved records according to its triple template and responds to the client with the generated triples.

The client–server communication is implemented by means of custom functions used in the triple template of the client, where the server responses are processed. Thus, any number of servers can be queried from a single client according to the client's triple template. This workflow sacrifices the parallelism of variation (a) in Fig. 10 in favor of space: In the worst case scenario, the entire data set of the server side will be transformed to triples, after a sequence of requests from the client. The enrichment of surveillance data with weather conditions is a typical case where this workflow is employed. Indeed, transforming all the available weather data to RDF triples, independently from the surveillance data, wastes a considerable amount of resources and storage space. In this case, the weather conditions at some position (and for a given time instance) are only needed if there is at least one record from surveillance data for that position and that time instance. Thus, the surveillance data source provides “filtering criteria” for selecting the weather conditions data to be transformed. In this case, the RDF-Gen instance that transforms the surveillance data acts as a client and requests from the RDF-Gen server the weather conditions at certain positions and time instances. The triples retrieved by the client can be further modified by the custom functions applied in its triple template. This indicates that this workflow can be also applied in cases where the ontology specifications are abstract and generic (in contrast to the workflow variation (a) in Fig. 10). For example, the triple template at the client may abstract weather conditions (e.g., using terms such as “cloudy”, “sunny”, “light wind”, etc.). In this case, the ontology does not need to describe the weather condition attributes at a fine level of detail.

The workflow variation (c) in Fig. 10 relies on mapping configurations to join two or more data sets and convert the data into triples as it would be from a single source, using one triple template. Mappings can be established between any number of data sources. However, increasing the number (or size) of data sources to be joined increases the work load (as well as storage and computational resources needed) at the data connector. For this reason, this workflow is not recommended to be used alone in the general case, but in combination with variations (a) and (b) shown in Fig. 10. Specifically, since the join operator is applied on the data connector, any of the data connectors of these workflows can join two or more data sources. This requires thorough data source analysis, to identify which data sources can be directly transformed into RDF triples w.r.t. a given ontology, and which data sources can be transformed on-demand (using the workflow variation (b) in Fig. 10) or jointly (using the workflow variation (c) in Fig. 10). As already discussed, the workflow variation (c) provides more flexibility on how the data are transformed. Recalling the example of Fig. 7, independent data transformation “highlights” equally all the data sources, transforming data about aircrafts and models. On the other hand, data source joins allow the data engineer to shape the data closer to the needs, e.g., focusing on aircraft and opting out the conceptualization of aircraft models. This variation, however, cannot be applied on streaming data sources.

Finally, the workflow variation (d) in Fig. 10 enables parallel processing of a data source to further reduce processing time. The triple generator is instantiated in more than one workers, and the data connector transmits the records to the workers. This variation is appropriate for Big Data scenarios or bursty streams, as we will discuss in the evaluation section. Given that each data source is processed record by record, it is sufficient to distribute the records to workers that will process the data independently and in parallel. Specifically, the centralized implementation illustrated in Fig. 2 is modified by replacing the data connector with a data access function which distributes the records to multiple workers. Each worker is a triple generator instance, which is initialized with the given configuration and triple template. This RDF-Gen variation can be combined with any other variation. However, when variation (d) is combined with variation (b), the parallel data conversion is applicable on the client side if only the server has enough resources to handle multiple requests. On the other hand, applying parallel data conversion on the server side of variation (b) may not improve performance, since the overall throughput is confined by the client’s RDF triples generation rate.

Table 3 summarizes the basic workflow variations presented in this section.

6.2 Link discovery via RDF-Gen

Link discovery (LD) is the process of discovering relations (links) between entities that originate from two different data sources. Formally, given two data sources, named *target*

Table 3 Summary of the basic workflow variations

Variation type	Features	Suggested use
Independent data conversion	Data are transformed independently, output can be archived in separate files,	On settings where data sources are independent and self-contained
Client–server data conversion	Strong interaction between RDF-Gen instances, enriching triples on demand	On settings where the output of one RDF-Gen instance is heavily dependent on another source (data from one source need to be transformed only if some other data exist in another source)
Joined data conversion	Join the data of one or more sources, to enable processing as if they where columns of a single table in the same source	On settings where the data source consists of multiple files (no need to transform each file separately)
Parallel data conversion	Parallel processing to further improve scalability	On settings where single thread scalability is not enough, Big Data/extreme scale settings

(denoted by **T**) and *source* (denoted by **S**), that provide information about entities, and a set of relations **R** of interest, the objective of Link Discovery is to compute pairs of entities $\langle a, b\rangle$ that satisfy r, where $a \in \mathbf{S}$, $b \in \mathbf{T}$ and $r \in \mathbf{R}$. It should be noted that LD tasks can be performed over different data sets, e.g., relational data sets, text or even spatial data. For instance, in the case of spatial data sets, the objective of Link Discovery is to discover pairs of spatial objects that satisfy a given spatial relation. Existing works in this area have primarily focused on the discovery of topological relations (within, overlaps, touches, etc.) between spatial objects by Nentwig et al. [16].

A naive method to compute this task is to check every relation for each pair of entities, resulting in computational complexity of $|\mathbf{S}| \cdot |\mathbf{T}| \cdot |\mathbf{R}|$. However, this can be improved by organizing the data in *blocks*, so that only pairs of entities in the same block need to be compared, often without sacrificing soundness or completeness. The mechanism that organizes the entities is known a "blocking mechanism". The LD task can be seen as a filter-and-refine process, where entities first are organized into blocks (groups) that are likely to satisfy some relations (filtered), and then, pairs of entities in each block are evaluated (refine) to discover those pairs that actually satisfy each relation.

RDF-Gen supports Link Discovery tasks during data transformation to RDF in two basic ways. First, by exploiting the client–server communication between RDF-Gen instances indicated as RDF-Gen variation (b) in Fig. 10. In more detail, for each resource in the data source processed by the client RDF-Gen instance, a request is sent to the server RDF-Gen instance, which returns back the information related to the given resource. Thus, each resource in the data source processed by the client RDF-Gen instance can be associated with the information provided by the server RDF-Gen instance. This approach has performance limitations, since data are not organized in either client or server RDF-Gen instances, and network communication cost is added on each client–server interaction. However, it can be used to discover links between any pair of data sources, including synchronized streams of data.

A second way to support Link Discovery tasks in RDF-Gen is by exploiting the triples functions in the triple template. Recall that based on its definition, a triples function returns a (possibly empty) set of triples for a given set of arguments. In this approach, RDF-Gen first organizes the data that are necessary in an in-memory data structure for efficient access. The exact data structure used is subject to the task at hand. For simple tasks over relational data, a plain key-value map can be used. For spatial Link Discovery tasks, a spatial index (such as an Rtree or a Grid) can be used to organize the spatial data. Then, any triples function in the template can exploit the prepared data structures, in order to discover links in the data. In case an empty set of triples is returned, this indicates that there is not any association between the data being processed and the in-memory data.

The following real-world examples explain more clearly how LD is realized in the context of RDF-Gen.

Example 6.1 In the maritime domain, each vessel is associated to a country flag, which indicates the country where the vessel has been registered. The country code in some data sources is provided in an ISO2 standard (e.g., “GR” for Greece) while other sources use ISO3 (e.g., “GRC” for Greece), which can be unrelated to the URI of the country as specified in the ontology. In this case, RDF-Gen builds a key-value dictionary between the ISO2/ISO3 codes and the URI of the corresponding country, and employs a function in the triple template which looks up the dictionary and associates each vessel to the URI of the country where it has been registered.

Example 6.2 Another example from the domain of spatial Link Discovery concerns regions of interest and vessel positions, where the need is to discover topological relations between them. In a preprocessing phase, RDF-Gen organizes in an in-memory spatial data structure the regions of interest. Then, for each vessel position that is received, the triples function in the template will take as arguments the identifier of the vessel and its position. The function will rely on the prepared data structure to identify if any topological relation can be discovered between the position of the vessel and the regions of interest.

7 Evaluation

In this section, we present an experimental evaluation of RDF-Gen. Aiming to a brief but thorough evaluation of the approach, the following paragraphs are organized into four sections. The first section evaluates the centralized implementation of RDF-Gen against state of the art related work on archival data sets, SPARQL-Generate Lefrançois et al. [14]) and GeoTriples Kyzirakos et al. [13]. The second section compares RDF-Gen with the state of the art on streaming data, SPARQL-Generate Lefrançois et al. [14]. Finally, the third section presents a comparison between various configurations of the parallel implementation of RDF-Gen, to evaluate the gain of performance achieved through parallelization. The data sets used for each experiment are briefly described in each section.

RDF-Gen has been extensively used in practice over a wide variety of data sets (e.g., various formats, input data sizes, input rates) in the EU-funded research project datAcron[9]. Some of the data sets are open to the public and can be accessed online[10]. Also, the config-

[9] http://www.datacron-project.eu.

[10] https://zenodo.org/record/1167595.

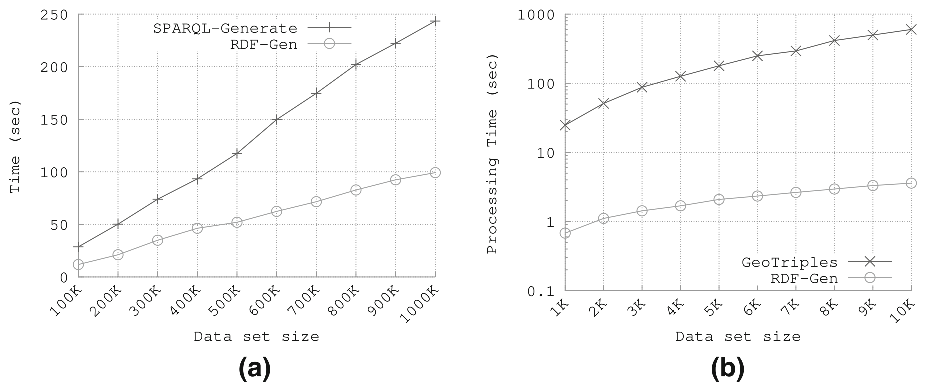


Fig. 11 Comparison of RDF-Gen with (**a**) SPARQL-Generate and (**b**) GeoTriples, *y*-axis reports time in seconds (logarithmic scale in **b**), for each data set size (number of records) reported in *x*-axis

uration files[11] and the generated RDF triples (in Turtle format) are available online[12]. The templates and generated RDF triples comply to the publicly available[13] datAcron ontology Santipantakis et al. [20].

7.1 Transformation of archival data

The performance of RDF-Gen on archival data is compared to SPARQL-Generate Lefrançois et al. [14] and GeoTriples Kyzirakos et al. [13]. Since the latter aims to be more efficient on domains related to spatial data, we use two different data sets, (a) a plain CSV file for the comparison between RDF-Gen and SPARQL-Generate, and (b) an ESRI shapefile for comparison between RDF-Gen and GeoTriples. This decision enables the evaluation of RDF-Gen in domains where state of the art approaches excel (e.g. GeoTriples is optimized for converting spatial data sources to GeoSPARQL Perry and Herring [18] (accessed August 10, 2019 triples), to justify that RDF-Gen is an efficient and generic solution.

7.1.1 Transformation of CSV files

Figure 11a shows how the performance of RDF-Gen compares to the performance of SPARQL-Generate on CSV files. For this experiment, we have generated 10 synthetic CSV files. Each file contains exactly 10 columns, and they vary in size from 100 K to 1 M records using a step of 100 K records. Although both approaches scale linearly to the size of the data, we observe that RDF-Gen is more efficient for all the cases. Also, the difference between the evaluated approaches seems to increase, linearly to the size of the data set. This indicates that RDF-Gen can be more prominent for data transformation on Big Data settings, compared to SPARQL-Generate.

[11] https://github.com/datAcron-project/RDF-Gen/tree/master/configurations/zenodo_dataset_configurations.

[12] https://zenodo.org/record/2576584.

[13] http://ai-group.ds.unipi.gr/datacron_ontology/.

7.1.2 Transformation of ESRI shapefiles

For comparison of RDF-Gen with GeoTriples, we use a set of ESRI shapefiles constructed from samples of maritime surveillance data sets regarding vessels in the Brest area. We evaluate the performance and scalability of the approaches using 10 such files, varying the number of geometries from 1 to 10 K, using a step of 1 K. The RDF-Gen configuration in this experiment enables the serialization of geometries to GeoSPARQL, so as to achieve identical RDF graphs with GeoTriples towards a fair comparison. The geometries in the shapefile contain 26 attributes, which are also transformed to the corresponding RDF resources or literals in the output.

Figure 11b shows the performance results of RDF-Gen and GeoTriples for the transformation of ESRI Shapefiles. The *y*-axis of the chart reports the data transformation time for GeoTriples and RDF-Gen in logarithmic scale. We recall that GeoTriples operates in two phases: First it constructs the mappings that will be used for the transformation of data to triples, and then, it applies the mappings on the data to generate the output. We observed that the time spent by GeoTriples to construct the mappings is not affected by the size of data set. This is expected since the mapping mainly depends on the number of attributes, which is the same for all files in this experiment. The average time spent by GeoTriples for constructing the mappings was approximately 1.5 s, which is already more than the total time spent by RDF-Gen to transform the first three data sets in this experiment. In addition to that, GeoTriples relies on RML for converting the data, and its performance seems to increase exponentially to the size of input. The results in this experiment show that RDF-Gen scales linearly to the size of input and it is up to two orders of magnitude faster than GeoTriples. Indeed, we observe that for the 1 K data set RDF-Gen takes less than a second, while GeoTriples takes more than 10 s. For 3 K data set RDF-Gen remains close to one second, while time processing for GeoTriples has already changed the order of magnitude to approximately 100. Finally, 10 K data set is processed in less than 10 s by RDF-Gen (justifying that the time processing is linear to the size of input), while GeoTriples has increased to approximately 1000 s.

7.2 Transformation of streaming data

The evaluation of RDF-Gen and SPARQL-Generate on streaming data sources was made by a data source streaming simulator to ensure that both approaches will be tested on the same data and transmission rate. The simulator can transmit messages varying both in size (i.e., number of attributes that will have to be transformed) and in rate (i.e., number of messages per second). Although RDF-Gen supports plain TCP socket streams, the simulator transmits data through a Websocket, since SPARQL-Generate can connect only to Websocket streams.

Various streams of data have been transformed by RDF-Gen. Among them, the bursty streams seem to be the most interesting cases. Specifically, these streams transmit a large number of messages in a very short time. A typical example of a bursty stream is the maritime surveillance stream on regions far away from the shores received from satellite links, transmitting thousands of positions of vessels in just a few minutes. Figure 12 provides an example of a bursty stream. Although the average number of messages in this example is approximately 45 messages per second, the number of messages in bursts can reach up to hundreds of messages per second. From our experience so far, the maximum number of records of all time in that stream was not higher than 700 messages per second for the constrained area of the Mediterranean Sea. This observation leads to the experimental setup for the evaluation of performance on streaming data sources. Specifically, we evaluate both RDF-Gen

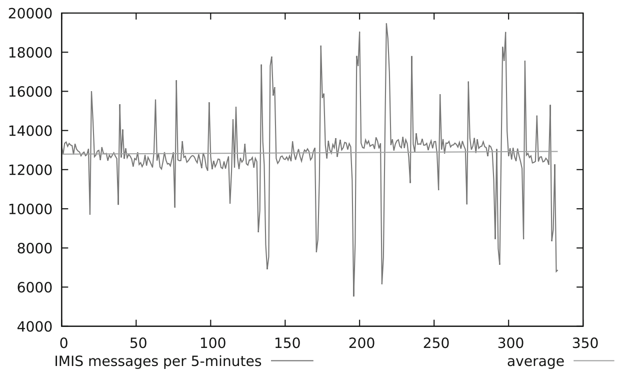


Fig. 12 Example of a bursty stream. Maritime surveillance (IMIS) number of records, aggregated per 5 min

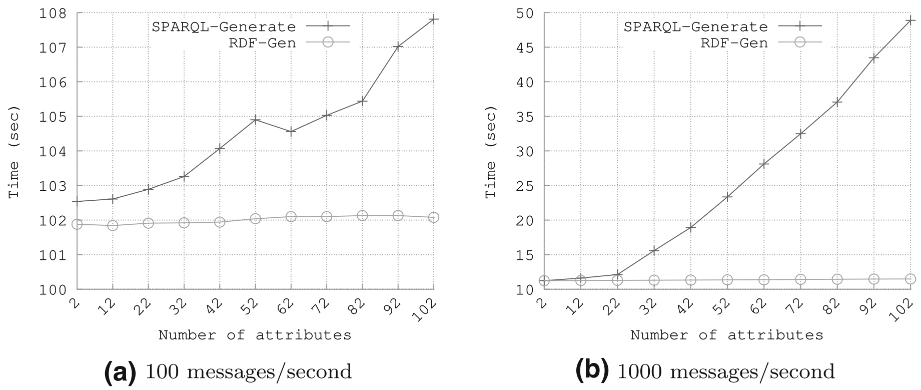


Fig. 13 Total processing time for 10^4 messages at different message rates for SPARQL-Generate and RDF-Gen. Ideal performance is the first value on *y*-axis for each case

and SPARQL-Generate on two transmission rates, one for 100 messages per second, and the other for 1000 messages per second. For each of these rates, the simulator transmits 10 types of JSON messages, varying in number of attributes from 2 to 102 (by a step of 10 attributes). Each experiment transmits 10^5 messages through the predefined Websocket.

Figures 13a and b illustrate the experimental results for rates of 100 messages per second and 1000 messages per second, respectively. We observe that SPARQL-Generate shows an exponential increase of total processing time to the number of attributes in the messages, while RDF-Gen seems to be unaffected. The main reason for this is that RDF-Gen allocates the memory to be used by the vector of variables at the initialization phase once and transforms each record using the same vector. The ideal transformation system would need exactly 100 (resp. 10) seconds to transform the 10,000 messages at a rate of 100 (resp. 1000) messages per second. Since the transformation systems cannot process instantly the data (e.g., parsers and custom functions require processing time), a difference of processing time from the ideal values is observed, which appears on the flow of data as a delay to the output stream. It is important therefore that the delay on the output, as computed from the difference between actual and ideal processing time values, is as close as possible to zero. The experimental results show that RDF-Gen has always lower delay compared to SPARQL-Generate. In addition to that, the delay from RDF-Gen is almost constant to the number of attributes in the

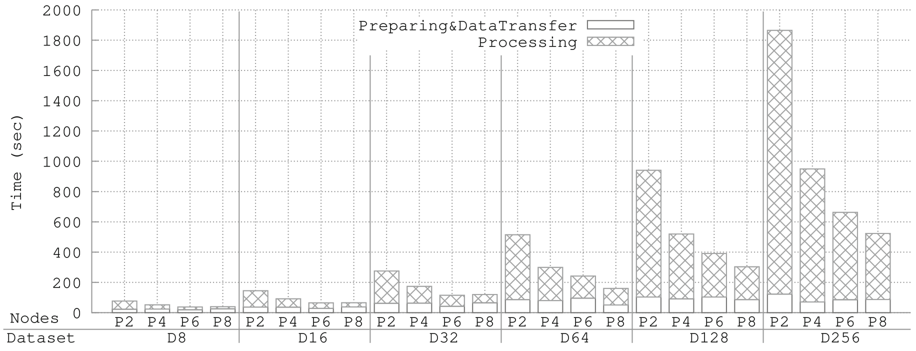


Fig. 14 Scalability of parallel RDF-Gen for configurations employing 2, 4, 6, 8 workers, on data set sizes from D8 to D256

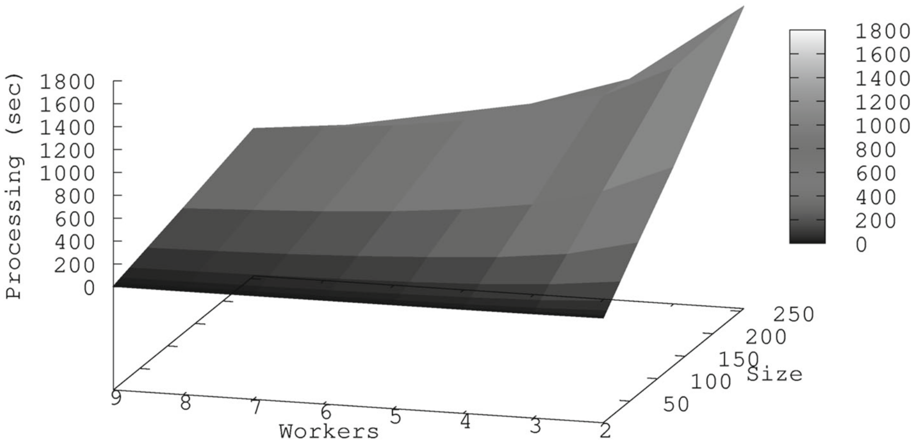


Fig. 15 3D surface illustrating the scalability of parallel RDF-Gen with respect to number of workers and data set size

messages, while the delay from SPARQL-Generate increases up to approximately 37 s for a rate of 1000 messages per second. This indicates that RDF-Gen can be used efficiently on bursty streams, for two main reasons: (a) the delay for as high as 1000 messages per second is considerably low, and (b) the delay is constant to the number of attributes, i.e., the size of message transmitted. On the other hand, SPARQL-Generate is expected to considerably increase the delay at the peaks of the bursty stream. This may result on a non-constant delayed view of the real conditions reported in the stream, which can make harder the real time exploitation of the transformed data.

7.3 Evaluation of parallel RDF-Gen implementation

This set of experiments evaluate various configurations of the parallel version of RDF-Gen, using real-life surveillance data sets that vary in size. The data sets used in this evaluation are samples from aviation radar logs, varying in number of records. Specifically, we extracted 138,781 records of actual surveillance data between a random pair of airports. We construct the another 8 data sets by replicating the extracted records, such that the i-th data set contains the double number of records of $(i-1)$-th. For example, the last data set contains $2^8 \times 138,781 = 35,527,936$ records. We denote each data set by the multiplication factor applied, i.e., the second data set will be "D4", the third will be "D8", until the eighth data set denoted as "D256". The parallel version of RDF-Gen has been evaluated on the datAcron cluster which consists of 10 computing nodes (workers). Each computing node has 128 GB of RAM, 256 GB of SSD storage and 2*XEON e5-2603v4 6-core 1.7 GHz with 15MB cache. We evaluate the parallel version of RDF-Gen using 2–8 workers for each experiment, denoted in the experimental results as P2–P8.

Figure 14 illustrates the total processing time for each data set from "D8" to "D256" and for configurations employing 2 to 8 workers. The total execution time is separated to (a) the time spent for cluster initialization and data transfer, and (b) the actual processing time on workers for data transformation. For example, the bar for value "D8,P6" reports the preparing, data transfer and processing time of the configuration employing 6 workers on a data sample with 8× 138,781 = 1,110,248 records. We observe that the preparation and data transfer time is not considerably increased to either the input data set size or the number of workers. This is explained by the fact that no interaction between workers is needed, as they transform the data independently. We also find that until data set D32 (i.e., approximately 4.4 million records), the benefit from employing more than 2 workers seems to be small, i.e., the input data set is too small for the parallel processing to make any difference on the processing time. However, it is clear from the case of D256 that increasing the number of workers can considerably boost the performance. Finally, we observe that the total processing time of "P4" (as well as "P8") configuration for any data set "D(i)" is almost the same as the processing of "P2" (resp. "P4") for the previous data set "D(i-1)". This fact shows that distribution of records is done evenly to the workers; thus, each worker on "D(i-1)P2" (resp. "D(i-1)P4") configuration has the same work load as "D(i)P4" (resp. "D(i)P8"). This is a strong evidence, that doubling the number of workers will divide the input data set by two, justifying that parallel RDF-Gen can be efficiently exploited in Big Data settings.

Finally, Fig. 15 summarizes the experimental results, illustrating the total processing time for all the data sets and number of workers evaluated. This figure also verifies that increasing the number of workers for small data sets may not improve performance, while processing of large data sets can certainly benefit from increasing the number of workers.

8 Conclusions and future work

In this paper, we presented RDF-Gen, an efficient and flexible approach for data transformation to RDF from diverse and heterogeneous data sources, both streaming and archival. RDF-Gen covers a multitude of objectives for generation of RDF data in comparison with existing approaches and tools, thus offering a richer repertoire of functionality. In addition, RDF-Gen provides a data-parallel variant that can be used for maximizing the performance of RDF data generation, in case of high-rate streaming data sources or very large archival data sets. In turn, this enables the manifestation of different workflows for RDF data generation, ranging from independent data conversion to joint conversion of data sources. Last, but not least, a comparative evaluation against state-of-the-art approaches demonstrates the advantages of RDF-Gen in a wide variety of setups.

Regarding the future work, we intend to expand the capabilities of RDF-Gen toward a full-scale solution to data integration, performed seamlessly to data transformation. Moreover, an interesting research direction is to infer the mappings between data sources and a given ontology in an automatic or semi-automatic way, thus minimizing the human intervention on configuring RDF-Gen, and offering a solution for "plugging" new sources into the process. Finally, the task of semi-automatic ontology construction from the underlying data sources would expedite the data integration task, eliminating the need for manual ontology design and facilitating ontology updates based on the requirements of new data sources.

Acknowledgements This work was supported by EU projects datAcron (Grant Agreement No 687591), VesselAI (Grant Agreement No 957237), and by the Hellenic Foundation for Research and Innovation (H.F.R.I.) under the "First Call for H.F.R.I. Research Projects to support Faculty members and Researchers and the procurement of high-cost research equipment grant" (Project Number: HFRI-FM17-81).

References

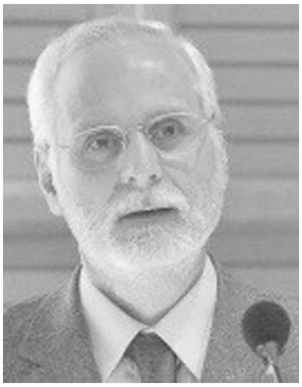
1. Brecher C, Özdemir D, Feng J, Herfs W, Fayzullin K, Hamadou M, Müller A (2010) Integration of software tools with heterogeneous data structures in production plant lifecycles. IFAC Proc Vol 43(4):48–53
2. Chortaras A, Stamou G (2018) D2RML: integrating heterogeneous data and web services into custom RDF graphs. In: Workshop on linked data, LDOW@WWW 2018
3. Dell'Aglio D, Valle ED, van Harmelen F, Bernstein A (2017) Stream reasoning: a survey and outlook: a summary of ten years of research and a vision for the next decade. Data Sci. J. 1:59–83
4. Dimou A, Sande MV, Colpaert P, Verborgh R, Mannens E, de Walle RV (2014) RML: a generic language for integrated RDF mappings of heterogeneous data. In: Proceedings of the 7th workshop on linked data on the web
5. Dong XL, Srivastava D (2015) Big data integration. Synthesis lectures on data management. Morgan & Claypool Publishers. https://doi.org/10.2200/S00578ED1V01Y201404DTM040
6. Efthymiou K, Sipsas K, Mourtzis D, Chryssolouris G (2013) On an integrated knowledge based framework for manufacturing systems early design phase. Procedia CIRP 9:121–126
7. ESRI (1998) Esri shapefile technical description. Technical report. Tech. rep., Environmental Systems Research Institute, Inc., 380 New York Street, Redlands, CA 92373–8100 USA, http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf
8. Haesendonck G, Maroy W, Heyvaert P, Verborgh R, Dimou A (2019) Parallel RDF generation from heterogeneous big data. In: Proceedings of the international workshop on semantic big data, SBD '19. pp 1:1–1:6
9. Hirzel M, Baudart G, Bonifati A, Valle ED, Sakr S, Vlachou A (2018) Stream processing languages in the big data era. SIGMOD Rec 47(2):29–40
10. Junior AC, Debruyne C, Brennan R, O'Sullivan D (2016a) FunUL: a method to incorporate functions into uplift mapping languages. In: Proceedings of the 18th international conference on information integration and web-based applications and services. pp 267–275
11. Junior AC, Debruyne C, O'Sullivan D (2016b) Incorporating functions in mappings to facilitate the uplift of CSV files into RDF. In: The semantic web - ESWC 2016 satellite events. pp 55–59
12. Knoblock CA, Szekely PA, Ambite JL, Goel A, Gupta S, Lerman K, Muslea M, Taheriyan M, Mallick P (2012) Semi-automatically mapping structured sources into the semantic web. In: The semantic web: research and applications. pp 375–390
13. Kyzirakos K, Vlachopoulos I, Savva D, Manegold S, Koubarakis M (2018) GeoTriples: transforming geospatial data into RDF graphs using R2RML and RML mappings. J Web Semant 52:16–53
14. Lefrançois M, Zimmermann A, Bakerally N (2017) A SPARQL extension for generating RDF from heterogeneous formats. In: The semantic web. pp 35–50
15. Meester BD, Maroy W, Dimou A, Verborgh R, Mannens E (2017) Declarative data transformations for linked data generation: the case of DBpedia. In: Proceedings of the 14th ESWC. pp 33–48
16. Nentwig M, Hartung M, Ngomo AN, Rahm E (2017) A survey of current link discovery frameworks. Semant Web 8(3):419–436. https://doi.org/10.3233/SW-150210
17. Ocker F, Vogel-Heuser B, Seitz M, Paredis CJ (2020) A knowledge based system for managing heterogeneous sources of engineering information. IFAC-PapersOnLine 53(2):10511–10517
18. Perry M, Herring J (2012) Open geospatial consortium. GeoSPARQL - A geographic query language for RDF data, OpenGIS implementation standard. Accessed 10 Aug 2019
19. Phuoc DL, Quoc HNM, Ngo QH, Nhat TT, Hauswirth M (2016) The graph of things: a step towards the live knowledge graph of connected things. J Web Semant 37–38:25–35

20. Santipantakis GM, Vouros GA, Glenis A, Doulkeridis C, Vlachou A (2017) The datAcron ontology for semantic trajectories. In The semantic web: ESWC 2017 satellite events. pp 26–30
21. Santipantakis GM, Glenis A, Kalaitzian N, Vlachou A, Doulkeridis C, Vouros GA (2018a) FAIMUSS: flexible data transformation to rdf from multiple streaming sources. EDBT 2018
22. Santipantakis GM, Kotis KI, Vouros GA, Doulkeridis C (2018b) RDF-Gen: generating RDF from streaming and archival data. In: WIMS, ACM. pp 28:1–28:10
23. Scharffe F, Atemezing G, Troncy R, Gandon F, Villata S, Bucher B, Hamdi F, Bihanic L, Képéklian G, Cotton F, Euzenat J, Fan Z, Vandenbussche PY, Vatant B (2012) Enabling linked data publication with the Datalift platform. In: Semantic cities @AAAI 2012, AAAI workshops, vol WS-12-13
24. Simsek U, Kärle E, Fensel D (2019) RocketRML - a NodeJS implementation of a use-case specific RML mapper. CoRR arXiv:1903.04969
25. Slepicka J, Yin C, Szekely P, Knoblock C (2015) KR2RML: an alternative interpretation of R2RML for heterogeneous sources. In: Proceedings of the 6th international workshop on consuming linked data (COLD 2015)
26. Venetis T, Vassalos V (2015) Data integration in the human brain project. In: Ambite JL, Ashish N (eds) Data integration in the life sciences. Springer, New York, pp 28–36
27. Vouros G, Santipantakis G, Doulkeridis C, Vlachou A, Andrienko G, Andrienko N, Fuchs G, Martinez MG, Cordero JMG (2019) The datAcron ontology for the specification of semantic trajectories: specification of semantic trajectories for data transformations supporting visual analytics. J Data Semant 8:235–262
28. Vouros GA, Vlachou A, Santipantakis GM, Doulkeridis C, Pelekis N, Georgiou HV, Theodoridis Y, Patroumpas K, Alevizos E, Artikis A, Claramunt C, Ray C, Scarlatti D, Fuchs G, Andrienko GL, Andrienko NV, Mock M, Camossi E, Jousselme A, Garcia JMC (2018) Big data analytics for time critical mobility forecasting: recent progress and research challenges. In: Proceedings of the 21th international conference on extending database technology, EDBT 2018, Vienna, Austria, March 26–29, 2018. pp 612–623

Georgios M. Santipantakis (BSc, MSc, PhD) holds a BSc in Electrical Engineering from TEI of Crete, an MSc in Information Management and a PhD in Knowledge Representation and Reasoning from the University of Aegean, Greece. Currently, he is a Post-Doctoral researcher in the Department of Digital Systems in the University of Piraeus. His research interests include Distributed Reasoning in Description Logics, Representation and Reasoning with Big Data, and Ontology Evolution.

Konstantinos I. Kotis (https://orcid.org/0000-0001-7838-9691) is currently a tenure track assistant professor at the University of the Aegean, Dept. of Cultural Informatics and Communication, i-Lab, and a research associate at the University of Piraeus, Dept. of Digital Systems, AI Lab. His research interests include Knowledge/Ontology Engineering, Semantic Web technologies, Semantic Data Management, the Internet/Web of Things, and conversational AI (Chatbots). He has published more than 70 papers in peer-reviewed international journals and conferences (Google Scholar h-index 18, citations>1500) and served as reviewer and PC member in several journals and conference events. He has also contributed to several national and European projects from different roles/positions. For more, please visit: http://i-lab.aegean.gr/kotis.

Apostolos Glenis is a PhD candidate in the Department of Digital Systems of the University of Piraeus. He holds a BS and an MS in Computer Science from the University of Piraeus and the University of Crete, respectively. His research interests include Data Streams, Machine Learning, Time Series Classification, Big Data Systems and Recommender Systems.



George A. Vouros is a Professor in the Department of Digital Systems in the University of Piraeus. His published work and scientific interests are in the areas of Expert Systems (has developed 4 successful systems in critical and complex domains during 1990-1997), Ontologies & Semantic Integration of data, Agents and MultiAgent Systems Reinforcement and Imitation Learning in multiagent and complex settings. He served/serves as program chair, chair and member of organizing committees of national and international conferences and as member of steering committees/boards of international conferences/workshops. He has given tutorials and keynote speeches in conferences and he has organized several workshops in well-known conferences. He served as guest editor in special issues in well-reputed journals. He is/was senior researcher in numerous EU-funded and National research projects.

Christos Doulkeridis received the BSc degree in electrical engineering and computer science from the National Technical University of Athens and the MSc and PhD degrees in Information Systems from the Department of Informatics of Athens University of Economics and Business. He is currently an associate professor in the Department of Digital Systems of the University of Piraeus. His research interests include parallel and distributed query processing, large-scale data management, distributed knowledge discovery, and spatio-temporal data management.

Akrivi Vlachou is an associate professor in the Department of Information and Communication Systems Engineering (ICSD) of the Aegean University, Greece. She received her PhD in 2008 entitled “Efficient Query Processing over Highly Distributed Data” from the Athens University of Economics and Business. She has published and been involved in the program committees of major database conferences, including SIGMOD, VLDB, and ICDE. Her research interests lie in query processing and data management in large-scale distributed systems.