RDF-Gen: Generating RDF from Streaming and Archival Data

Georgios M. Santipantakis
Department of Digital Systems
University of Piraeus, Greece
gsant@unipi.gr

George A. Vouros
Department of Digital Systems
University of Piraeus, Greece
georgev@unipi.gr

Konstantinos I. Kotis
Department of Digital Systems
University of Piraeus, Greece
kotis@aegean.gr

Christos Doulkeridis
Department of Digital Systems
University of Piraeus, Greece
cdoulk@unipi.gr

ABSTRACT
Recent state-of-the-art approaches and technologies for generating RDF graphs from non-RDF data, use languages designed for specifying transformations or mappings to data of various kinds of format. This paper presents a new approach for the generation of ontology-annotated RDF graphs, linking data from multiple heterogeneous streaming and archival data sources, with high throughput and low latency. To support this, and in contrast to existing approaches, we propose embedding in the RDF generation process a close-to-sources data processing and linkage stage, supporting the fast template-driven generation of triples in a subsequent stage. This approach, called RDF-Gen, has been implemented as a SPARQL-based RDF generation approach. RDF-Gen is evaluated against the latest related work of RML and SPARQL-Generate, using real-world datasets.

CCS CONCEPTS
• Information systems → Resource Description Framework (RDF); • Computing methodologies → Knowledge representation and reasoning;

KEYWORDS
RDF generation, RDF knowledge graph, data-to-RDF mapping

RDF generation tool, and (c) the computation of data dependencies and links with any of the existing sources. One additional issue concerns hindering the reusability of data processing functions (e.g. for converting data to common formats, or constructing URIs for entities) used by different RDF generators: Using different functions for converting data to common formats, or constructing URIs for RDF generation (or RDFization) process. The variety and heterogeneity of data sources and existing data models, together with their streaming or archival nature, set the challenge of building computational efficient and comprehensive solutions to transforming and linking data from all sources, that can be flexibly tailored to the idiosyncrasies of individual data sources, supporting reusability and extensibility.

Indeed, a large volume of streaming data is daily being generated in a variety of domains, varying from data that are crucial to situation awareness (e.g. surveillance data), to social networks: Such data need to be associated with archival data provided by different stakeholders, most commonly in different formats and in different models, so as to generate integrated and coherent views on data that analysis tasks require. This is the case for instance in the air traffic management domain (ATM) where surveillance data from flights provided by radar tracks (IFS) and by ground-based receivers (ADS-B) should be associated with airspace configurations and contextual information provided in CSV (e.g. sector configuration) and XML (e.g. flight plans provided by Eurocontrol Demand Data Repository (DDR) and Eurocontrol Network Manager). As an example of the size of data to be processed, just for one day, IFS provides approximately $10^9$ records of data in the European airspace.

The use of different technologies for the generation of RDF from necessary data sources in a domain, results to approaches that require maintaining and extending different workflows regarding the data processing and management tasks, setting-up and tuning RDF generators’ parameters, and maintaining implementation/customization solutions. This may imply hindering (a) fine control on the generated RDF data; as dependencies and links between different sources may be hidden in different mapping specifications and coding details; (b) the extensibility and reusability of the solutions; as any new data source type may require the incorporation of a new RDF generation tool, and (c) the computation of data dependencies and links with any of the existing sources. One additional issue concerns hindering the reusability of data processing functions (e.g. for converting data to common formats, or constructing URIs for entities) used by different RDF generators: Using different functions in different tools, and relying on black-box solutions, entails additional, considerable cost for maintaining solutions, and verifying RDF data generated.

Therefore, an RDF generation approach from multiple data sources ideally should imply a unique workflow. This should be familiar to the engineers working with RDF models and SPARQL, easily
We aim at providing coherent and integrated views of data, while they may not support linking data from different sources. This is in particular the case with positional data provided by any moving object, either in air, sea or terrain. Processing data close to the data sources in the RDF generation domain means to move the execution of processing functionality provided by RDF-Gen in early steps of data management, and as close to the data sources as possible.

The work presented in this paper has been motivated by the need to gather data from large number of moving entities in large geographical areas, in combination with contextual data from various sources, and other data concerning the moving entities themselves. We aim at providing coherent and integrated views of data, contributing to increasing the accuracy of predictive analysis tasks. In the context of datAcron project [13], in particular, we aim to transform large volumes of data arriving to datAcron system through streaming and archival sources, into meaningful RDF triples, according to the datAcron ontology [10]. The process, aiming to support either online or batch analysis tasks, needs to be highly efficient, scalable, and satisfying strict latency requirements, significantly reducing storage requirements.

The datAcron project addresses core challenges related to the European Big Data Vision towards increasing abilities to acquire, integrate, process, analyse and visualize data-in-motion and data-at-rest in integrated manners. It addresses requirements from the air-traffic management and maritime domains by developing advanced tools for detecting and visualizing threats, abnormal activity, increasing the safety and efficiency of operations related to vessels and airplanes, and further reducing the impact of these operations on the environment.

Recent state-of-the-art approaches and technologies for generating RDF graphs from non-RDF data use languages that have been specifically designed for transforming or mapping data of various kinds of format [3, 7, 8, 11]. However, these approaches do not satisfy the computational efficiency and scalability requirements required in many domains exploiting surveillance data, or have inherent limitations in extensibility and reusability of solutions, while they may not support linking data from different sources seamlessly to RDF generation.

Driven by the datAcron domains’ requirements and the limitations of existing RDF generation approaches, this paper presents a new approach towards meeting the goal of generating RDF knowledge graphs, integrating data from multiple heterogeneous streaming and archival data sources with high throughput and low latency. To support this, and in contrast to existing approaches, we propose a generic framework embedding a close-to-sources data processing and linkage stage in the RDF generation process, supporting the fast template-driven generation of triples in a subsequent stage. This approach, called RDF-Gen, has been implemented as a SPARQL-based RDF generation approach and currently supports the generation of RDF data from numerous data sources. RDF-Gen is evaluated against the latest related work of RML and SPARQL-Generate, using real world datasets.

Closely to the objectives of recent state-of-the-art approaches [3, 11], and based on the datAcron aviation and maritime requirements, as presented in this section, RDF-Gen satisfies the following individual objectives (subsequently denoted by Ox):

O1 Inherently supports the RDF generation of both streaming and archival datasets.
O2 Provides facilities for close-to-source data processing tasks, e.g. for data cleansing, data manipulation and conversion, and generation of URLs.
O3 Supports close-to-source link discovery functionality.
O4 Demonstrates computational efficiency in terms of high throughput and low data-generation latency.
O5 Demonstrates the scalability which is necessary for the transformation of big data.
O6 Demonstrates extensibility, in the sense that (i) it can integrate custom data processing and manipulation functions, and (ii) it can be instantiated to new data formats.
O7 Supports reusability of solutions across data sources of the same domain.

Thus, this paper presents in a comprehensive way the RDF-Gen approach, and compares it against latest related RDF generation tools (RML and SPARQL-Generate), using real world datasets. A demonstration of the data transformation functionality has also been presented in [9].

The rest of the paper is organized as follows: Section 2 presents related work and Section 3 presents the RDF-Gen framework in detail and with specific examples. Section 4 evaluates RDF-Gen instantiations against state-of-the-art approaches. Finally, Section 5 provides a short discussion on results and future plans.

2 RELATED WORK

In this section, we conduct a meticulous review of existing methods and tools for data transformation to RDF. Our findings are summarized in Table 1, which also provides an evaluation of the reviewed approaches in comparison with the objectives outlined earlier.

RML [3] is a mapping language based on R2RML, the W3C standard for mapping relational databases into RDF. It follows an extensible RDF generation approach, while supporting the definition of graph templates (called mappings in RML’s context) for multiple heterogeneous sources. The language is extensible, as the whole solution relies on the extension of the R2RML mapping language. RML does not require storing files into memory, making it appropriate for processing datasets too big for the processor’s memory (e.g. datasets from streaming sources). However, mapping times reported for streaming data transformations are in trade-off with memory usage. 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 [4] or FunO [2]. However, the required functions should in-principle be re-used in every mapping, increasing the time needed for the overall RDF generation process. Furthermore, the validation of the
generated output is not a straightforward task since it requires familiarity with RML and FunUL or FnO. On the positive side, similar to RDF-Gen objectives listed previously, the approach supports O2, O3, O5, O6, and O7.

Table 1: Comparison of existing approaches for RDF generation.

|                | O1 | O2 | O3 | O4 | O5 | O6 | O7 |
|----------------|----|----|----|----|----|----|----|
| RML[3]         | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| SPARQL-Generate[8] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| KR2RML[12]     | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| RMLProcessor[5] | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| DataLift[11]   | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |
| RDF-Gen        | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  | ✓  |

SPARQL-Generate [8], has been recently introduced, supporting the generation of RDF from: (i) any RDF dataset, and (ii) any set of documents in arbitrary formats. SPARQL-Generate has been designed as an extension of SPARQL 1.1, so it can provably (i) be implemented on top any existing SPARQL engine, and (ii) leverage the SPARQL extension mechanism to deal with an open set of formats. Furthermore, it can be easily learned by knowledge engineers that are familiar with SPARQL 1.1, incorporated seamlessly to their workflows. Authors report that their first SPARQL-Generate open source implementation performs better than the reference implementation of RML for big transformations. As in other related works that use SPARQL-like mappings in the RDF generation process, SPARQL-generate supports easy validation of generated output. On the other hand, this approach does not support streaming data sources (objective O1) and does not integrate close-to-source processing and link discovery functionality (objectives O2 and O3). On the positive side, the approach supports (or there is adequate published information to claim that it supports): O6 (ii), and O7.

KR2RML [12] is another interpretation of R2RML, integrated in the open source tool Karma [6], paired with a source-agnostic R2RML processor that supports data cleaning and transformation. The approach supports easy to add new input and output formats without modifying the language or the processor, while supporting efficient cleaning, transformation, and generation of voluminous RDF datasets. This alternative interpretation of R2RML and its processor have been embedded in Apache Hadoop and Apache Storm to generate billions of triples and billions of JSON documents in both a batch and streaming fashion and can be extended to consume any hierarchical format. It supports the creation of data manipulation functions via an editor. However, once those functions (written in Python) are stored, capturing also structured information as a string, it requires parsing these specifications while computing mappings, imposing additional computational overhead to the overall approach. On the positive side, this approach supports O1, O2, O4, O5, and O6.

RMLProcessor [3, 5] 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 complex data transformations. Since the approach focuses on functions, these may provide functionality for any data format and can be reused in the mapping. On the positive side, similar to RDF-Gen objectives, the approach (to the best of our knowledge) supports: O2, O6, and O7.

The DataLift [11] project aims at providing a framework and an integrated platform for publishing datasets on the web of Linked Data. DataLift does not support the integration of custom processing functions. 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. On the positive side, the approach supports (or there is adequate published information to claim that it supports): O6, O7.

Finally, other existing approaches, such as GeoTriples [7], reuse existing RDF mapping engines such as R2RML and RML, thus they inherit their limitations and advantages. We also acknowledge the most recent RML-engine implementations of CARML, which is still in early beta state, according to their github page (https://github.com/carml/carml).

3 RDF-GEN

Motivated by the syntactic and semantic heterogeneity of datasets encountered in the use cases of the datAcron project, we have reached the RDF generation objectives listed above by means of the RDF-Gen framework.

As depicted in Figure 1, the implemented RDF-Gen framework comprises three main generic components: a) the Data Connectors, b) the Triple Generator, and c) the Link Discovery component.

As already mentioned, this approach supports embedding a close-to-sources data processing in the RDF generation process, implemented by the Data Connectors, aiming to support the fast template-driven generation of triples in a subsequent stage, implementing by the Triple Generator. The link discovery component supports link discovery functionality that can be provided during the RDF generation process, but not as close-to-the-sources as it can be done by the Data Connectors.

3.1 Data Connector

Given a configuration setting, Data Connectors connect to a set of input data sources, consuming, processing and providing data to the Triple Generator. Data Connectors provide data in a uniform size vector of values, i.e. a record. In doing so, Data Connectors can
support any data source type (streaming or archival, providing data in any kind of format).

Therefore, the Data Connector can be seen as an iterator over the data source, iterating over the data entries provided by the source, fetching the data needed to construct the records, according to the configuration provided. For example, a record from a CSV file may be constructed by a complete line, or by specific attribute values per line. For an XML file, a record can be an XML element with all of its children and attributes, while for a shapefile it can be a feature (i.e., a geometry) with all of its attributes. In addition, the Data Connector can also connect to SPARQL endpoints for an RDF-to-RDF (e.g., under different schema) conversion. The configuration involves a time interval, setting the time period to repeatedly pose the query, if this is needed. The size vector in this case is the selected variables in the SPARQL query. A variable can be marked as optional, in which case the corresponding position in the vector will be empty and handled accordingly in the template.

The configuration setting of a Data Connector specifies a mapping between attributes in the data source and record constituents. Thus, the Data Connector retrieves attribute values from the source and constructs records using these values in the appropriate record constituents. Such a mapping specification may include a filtering mechanism to exclude entries in the data source w.r.t. their position in the source (e.g., exclude the first n entries) or w.r.t. values on specific attributes. Transformation of values is supported by means of specific functions. Indicative examples are included in Table 2, demonstrating value transformations. For instance, such a filtering functionality can be used for basic data cleansing and/or for filtering out entries whose values of attributes appear to be erroneous or outliers. Further processing options can be supported such as conversion of values (e.g., between unit systems or coordination reference systems) or data extraction (e.g., extracting the Minimum Bounding Rectangle, or the Well-Known-Text representation of a geometry).

The Data Connector is configured to the corresponding data source. Specifically, the configuration specifies the connector type to be used (CSV, XML, ShapeFile, SPARQL endpoint, etc), and connector dependent options, such as delimiter character for CSV files, base XPath for XML files, records to be excluded (e.g., the first line for some CSV files usually contains labels), service address for remote SPARQL endpoint connectors, etc. The connector configuration specifies also the attributes whose values need to be considered for RDF generation. This decision obviously affects the size of variables vector used by Triple Generator, as depicted in Figure 2 and discussed in Section 3.2. Specifically, the values of the selected attributes will form a vector of values, and each value in the vector will be assigned to the corresponding variable in the vector of variables. Missing values in the data source do not affect the configuration, since in this case the empty value will be assigned to the corresponding variable (handled accordingly in the template).

For example, in the case of XML files, the configuration file first specifies the level of XML element by its XPath with filename prefix; e.g., the path

\[ \text{AIXM/ArrivalLeg.BASELINE/ADRMessage/hasMember/ArrivalLeg} \]

specifies that the Data Connector will iterate through all the elements in XPath /ADRMessage/hasMember/ArrivalLeg of file “ArrivalLeg.BASELINE” in folder “AIXM”. The configuration file also provides the XPath specifications to the attributes that will be used in the mapping, separated by commas. For instance, the following line

\[ /gml:identifier,/aixm:timeSlice/aixm:ArrivalLegTimeSlice/aixm:endPoint/aixm:TerminalSegmentPoint/aixm:pointChoice_airportReferencePoint/@xlink:href \]

specifies that only the values of gml:identifier and the xlink:href should be retrieved from the XML file, for each element at

\[ /ADRMessage/hasMember/ArrivalLeg \]

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 \( R \), iff the attribute values satisfy the filter \( f \).

Reusing the filter functionality in the Data Connector, we can apply equi-join operations in the set of data sources in \( D \). In this case, the mapping function generates a record \( R \), s.t.

\[ R = \mu_f(d_i, e) \mid \mu_f(d_i, e), \]

where \( e_i \in d_i, e_j \in d_j \), have common attributes. For example, if we know that a specific attribute value is shared in entries in the data source and it can be used as unique identifier, then we can join the entries and produce one record for these “joined” entries. This may also happen for entries in different sources, resulting to crossing/linking data from these sources in one record. Data format dependent optimizations can be applied (e.g., caching or preprocessing), but these are technical details and beyond the scope of this paper. Following the XML example, we use XPath also for the case of equi-join operations. For example, the statement:

\[ \text{AIXM/ArrivalLeg.BASELINE/ADRMessage/hasMember/ArrivalLeg/aixm:timeSlice/aixm:ArrivalLegTimeSlice/aixm:endPoint/aixm:TerminalSegmentPoint/aixm:pointChoice_airportReferencePoint/@xlink:href=AIXM/} \]

will equi-join elements of \( /ADRMessage/hasMember/ArrivalLeg \) in file ArrivalLeg.BASELINE, with elements of \( /ADRMessage/hasMember/AirportHeliport/ \) in file AirportHeliport.BASELINE, at the level of \( aixm:pointChoice_airportReferencePoint \) (i.e. as an inner element), when values of xlink:href and gml:id are equal.

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 RDF generation process is very low.

### 3.2 Triple Generator

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 vector of variables \( V \), a RDF Graph template \( G \) and a set of functions, which are made available to all instances of the Triple Generator. As already specified, variables in \( V \) correspond to the attributes that form a record: The i-th variable in \( V \) will be assigned
the i-th (possibly empty) value in any record provided by the Data Connector.

Formally, let \(I, B, \text{and } L\) the pairwise disjoint infinite sets of URIs, blank nodes and literals, respectively. A triple \((s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)\) is called an RDF triple, where \(s\) is the subject, \(p\) is the predicate and \(o\) is the object of the triple. An RDF graph \(G\) is a set of RDF triples. \(V\) is the infinite set of variables that is disjoint from the above sets and \(F\) an infinite set of function names disjoint with \(I, B, L, \text{and } V\).

A variable \(?x \in V\) is said to be bounded to a value \(q\) in \(I \cup L\) if, and only if, \(\text{bound}(?x) = q\). We distinguish the following categories of functions in \(F\):

1. The functions \(f \in F\) s.t. \(s = f(A)\) (or \(o = f(A)\)), \(s \in I\) (resp. \(o \in I \cup L\)), \(A \subseteq V\) which can be used as subject (resp. objects) in triples,

2. The functions \(f \in F\) s.t. \((s, p, o) = f(A)\), \(A \subseteq V\) which can be used for the construction of a (possibly empty) set of triples from the bounded variables of \(A\).

A triple template is a triple \(\tau \in (I \cup L \cup V \cup F) \times (I \cup F) \times (I \cup L \cup V \cup F)\). Thus, any of the triple subject, predicate or object can be a variable or a function.

A graph template is defined recursively as a set of:

1. A triple \(\tau \in (I \cup L \cup V \cup F) \times (I \cup L \cup V \cup F) \times (I \cup L \cup V \cup F)\)
2. Functions \(T = f(A)\), where \(f \in F\) and \(A \subseteq V\), s.t. \(T\) a set of triples constructed from \(f\) given the bounded variables in \(A\).

Given that the Triple Generator generates triples simply by bounding variables to their corresponding values using the graph template for the construction of triples, and/or by evaluating the pre-compiled functions with bounded values as arguments, we can expect computational efficient and scalable RDF generation. Although functions may implement any computation needed, they are in general of very low complexity.

An example of data conversion into triples, is provided in Figure 2, where the data source provides surveillance data from the aviation domain describing the spatiotemporal position of aircraft. For the sake of presentation, the input to the Triple Generator is just one line from a CSV file, provided by an appropriate CSV Data Connector: the Data Connector provides a record that corresponds to a specific time instant (\(\text{bound}(?ts) = "15:21:30 UTC"\)) and a specific moving object (\(\text{bound}(?d) = "001"\)).

A set of eight variables is provided in order to bind input values to variable names. For instance, the first variable (?1d) will be bound to the first value (CSV column) of the input record i.e. ”001”, corresponding to the aircraft’s ID. The graph template constructs trajectory nodes for aircraft, given the following: ID, 3-D position in the form of (?lon), (?lat) and (?alt) corresponding to longitude, latitude and altitude respectively, time point (?ts), status (?status), speed (?speed) and heading (?heading).

The Triple Generator binds each variable to the corresponding value in the record, and replaces the variables in the graph template with the bounded values. In case these are arguments of functions, it evaluates the functions and appends their result to the output. The graph template is specified w.r.t. the datAcron ontology\(^1\).

Another example from the same domain is provided in Table 2, illustrating the triples generated from a given template and a set of bounded variables, for a dataset of airports.

### 3.3 Link Discovery

When transforming data from different and – in the general case – heterogeneous sources to RDF, the generated data should be linked to provided integrated views of data. The link discovery component performs such a link discovery task, by identifying entities in different sources, discovering domain-specific links among entities, and describing entities gathering data from different sources. This component aims to discover links between the data that is generated by RDF-Gen.

In doing so, the proposed method is to instantiate the appropriate generator component as a service in a client-server mode: This is particularly useful in cases where dependencies among data in different sources exist and data should be linked. For example, we need to link weather conditions of a spatiotemporal position \(p\), only if there is a moving object at \(p\). In this example, the Triple Generator instance which is responsible for positional data acts as a client and the Triple Generator instance which is responsible for weather conditions acts as a server. The server instance is listening on a port, accepting HTTP requests from functions in the graph template of the client, and responds with triples, which are used as results of these functions in the client. This capability is an additional feature of the RDF-Gen framework, which enables distributed processing during RDF generation and link discovery between data from different sources.

There are cases where close-to-the-source link discovery is not applicable: Consider a data source with large volume of data that cannot be stored in memory and have to be asynchronously accessed for the generation of links. For example, an attribute that needs to be added in the flight plan and which is submitted asynchronously to the time the flight plan is issued. Thus, close-to-the-source link discovery is desired (and supported by our method), but it is not always applicable.

### 3.4 Distributed Processing and Scalability

As already discussed, RDF-Gen has been designed for generating RDF from archival and streaming data sources. To achieve scalability for voluminous archival sources but also for high input rate streams, RDF-Gen supports distributed processing by its design. In particular, RDF-Gen adopts a ”record-by-record” approach for data processing, which guarantees that any input record is processed individually and independently of other records. As such, distributed processing of input records is naturally supported, and can be implemented by partitioning input records to the available computing nodes. Moreover, this distribution is also supported in the link discovery component of RDF-Gen. This feature is demonstrated by the performance of our approach, as presented in the evaluation section of this paper.

### 4 EVALUATION

This section presents the results of experiments performed for the evaluation of RDF-Gen against the state-of-the-art approaches,
Figure 2: Example of RDF-Gen illustrating input, output, and configuration files

Table 2: Examples of triples generated from templates and bounded variables.

| Template                  | Bound Variables | Triples and Explanation                                                                 |
|---------------------------|-----------------|-----------------------------------------------------------------------------------------|
| makeURI(?icao,?iata) a :Airport; | ?icao="LAGV", ?iata="ATH" | :Place_Athina_EleftheriosVenizelos_Airport a :Airport; Explanation: makeURI("ELGR", "") will create a URI for the airport given its ICAO code (or IATA if it hadn’t been empty). If both codes are empty the function creates a random URI. |
| :hasPlaceName asString(?name); | ?name="ATHINA/ ELEFTHERIOS VENIZELOS" | :hasPlaceName "ATHINA/ ELEFTHERIOS VENIZELOS" ; Explanation: Lookup("ELGR") will request the name of the airport. |
| makeCodes(?icao,?iata) | ?icao="LAGV", ?iata="ATH" | :hasIATAcode "ATH" ; :hasICAOcode "LAGV" ; Explanation: makeCodes("ELGR", ") creates zero (if ICAO and IATA are empty) or more triples for the given codes |
| :hasGeometry getGeom(?long,?lat); | ?long=23.9444, ?lat=37.9366 | :hasGeometry :geom_52 ; Explanation: getGeom(23.94, 37.93) retrieves from external service (e.g. remote Triple Generator) the URI of the geometry at the specified latitude and longitude values. |
| :hasWKT getMBRGeom(?long,?lat). | ?long=23.9444, ?lat=37.9366 | :hasWKT "POINT (23.9444 37.9366)" . Explanation: getMBRGeom(23.94, 37.93) computes the Minimum Bounding Rectangle of the geometry retrieved for the specified latitude and longitude values. |

namely, RML and SPARQL-Generate. All experiments were performed on a desktop computer with 8 cores (Intel(R) Core(TM) i7-7700 CPU @ 3.6GHz) and 16 GB of RAM, running 64-bit Linux OS with Linux Kernel 4.8.0-53-generic and Java 8. The code and online testbed of the evaluated implementation is available at https://github.com/datAcron-project/RDF-Gen/ for reproducibility purposes.

We performed experiments using three different datasets, for typical or large volumes of data varying between 100 and 100,000 entries. The paper reports on the achieved micro average throughput per dataset (i.e. the number of records processed per second, as...
the ratio of TotalNumberOfRecords/TotalProcessingTime) and the total processing time for each dataset. The datasets are from the aviation domain of the datAcron project and from the following datasets used in SPARQL-Generate experiments [8]:

1. An artificial dataset of Persons, generated by GenerateData.com, used in SPARQL-Generate [8], mapping 8 properties
2. A real-life archival dataset of aircrafts [3], mapping 9 properties
3. Aircraft surveillance streaming data, mapping 5 properties

Since RML and SPARQL-Generate available implementations [4] do not support streaming data sources, we evaluated those methods using offline dumps of streaming data sources to CSV files.

Figures 3-5 present the achieved throughput by each RDF-Gen instance, for each of the datasets, varying their size. We observe that RDF-Gen has considerably higher throughput compared to the others. As it can be observed, for datasets of less than 5,000 records, RDF-Gen has not achieved its maximum throughput. This shows that RDF-Gen can support streaming data sources with strict latency requirements: Overall, the average time per triple generated is approximately 0.04 seconds, given that the frequency of position reporting per aircraft/vessel is at least 2 seconds. The presented system is limited to support a certain number of updates. This limitation however can be easily confronted by the proposed approach by introducing parallelization i.e. splitting the input data set per aircraft and processing the data by different instances of the RDF-Gen system.

For example, we observe that for 100,000 entries in the "Persons" dataset, RML achieves a throughput of 3.65 entries per second, SPARQL-Generate achieves a throughput of 6,477.52 entries per second, and RDF-Gen is capable of processing 27,034.33 entries per second. It must be pointed out that this is an estimation, given that, to the best of our knowledge, neither RML nor SPARQL-Generate publicly available implementations support streams of data.

Figures 6 and 7 illustrate the scalability of the evaluated RDF generators. Specifically, Figure 6 shows the processing time for surveillance data sources of varying size, in logarithmic scale, since RML seems to have exponential time behavior.

We repeat the experiment for the same dataset and only for SPARQL-Generate and RDF-Gen, using larger samples of the dataset, ranging from $10^5$ to $10^6$ records.

The results depicted in Figure 7 support our argument that RDF-Gen outperforms SPARQL-Generate. This is due to the design of our approach for efficient “record-by-record” access, inherently supporting distribution of processing. Taking advantage of this, the performance and scalability of the RDF-Gen approach can be further improved.

In addition to the evaluation of RDF generators performance we further discuss here usability issues. We do this using a transformation/mapping example for the “Persons” dataset since this dataset
Figure 6: Processing time of evaluated approaches for surveillance datasets of different size.

Figure 7: Processing time of SPARQL-Generate and RDF-Gen for large surveillance datasets.

has been previously used in other published work [8] for the comparison of RML and SPARQL-Generate. We have placed the full syntax of this example online at: https://github.com/datAcron-project/RDF-Gen/tree/master/MappingExample.

The following is a typical example of RML syntax to specify a mapping of “Persons” data w.r.t. FOAF and Schema.org vocabularies. The user needs to be familiar with both the Turtle syntax and the RML namespace (rr prefix):

```turtle
@prefix rr: <http://www.w3.org/2001/rml#>
@prefix rml: <http://semweb.mmlab.be/ns/rml#>
@prefix ql: <http://semweb.mmlab.be/ns/ql#>
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>
@prefix ex: <http://www.example.org/>
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix schema: <http://schema.org/>
@base <http://example.org/>

<#BenchmarkMapping>

rr:subjectMap {
  rr:template "http://example.org/person/(PersonId)(Name)";
  rr:class foaf:Person;
  rr:variable RML:template .
  rr:predicateObjectMap {
    rr:predicate foaf:name;
    rr:objectMap [ rml:reference "Name" ];
    rr:variable RML:template .
  }
  rr:predicateObjectMap {
    rr:predicate foaf:mbox;
    rr:objectMap [ rr:template "mailto:(Email)" ];
    rr:variable RML:template .
  }
  rr:predicateObjectMap {
    rr:predicate foaf:phone;
    rr:objectMap [ rr:template "tel:(Phone)" ];
    rr:variable RML:template .
  }
  rr:predicateObjectMap {
    rr:predicate schema:birthDate;
    rr:objectMap [ rml:reference "Birthdate" ; rr:datatype xsd:dateTime ];
    rr:variable RML:template .
  }
  rr:predicateObjectMap {
    rr:predicate schema:birthDate;
    rr:objectMap [ rml:reference "Height" ; rr:datatype xsd:decimal ];
    rr:variable RML:template .
  }
  rr:predicateObjectMap {
    rr:predicate schema:birthDate;
    rr:objectMap [ rml:reference "Weight" ; rr:datatype xsd:decimal ];
    rr:variable RML:template .
  }
}
```

The corresponding SPARQL-Generate transformation/mapping specification is provided in the following lines:

```sparql
PREFIX sgiter: <http://w3id.org/sparql-generate/iter/>
PREFIX sgfn: <http://w3id.org/sparql-generate/fn/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX schema: <http://schema.org/>
BASE <http://example.org/>

GENERATE {
  ?personIRI a foaf:Person;
  foaf:name ?name;
  foaf:mbox ?email;
  foaf:phone ?phone;
  schema:birthDate ?birthdate;
  schema:height ?height;
  schema:weight ?weight .
} 
SOURCE <personsDataSource> AS ?persons
ITERATOR sgiter:CSV(?persons) AS ?person
WHERE {
  BIND( sgfn:CSV(?person, "PersonId") AS ?personId )
  BIND( URI(CONCAT("http://example.com/person/",?personId )) AS ?personIRI )
  BIND( sgfn:CSV(?person, "Name") AS ?name )
  BIND( URI( CONCAT("tel:", sgfn:CSV(?person, "Phone") ) ) AS ?phone )
  BIND( URI( CONCAT("mailto:", sgfn:CSV(?person, "Email") ) ) AS ?email )
  BIND( xsd:dateTime( sgfn:CSV(?person, "BirthDate") ) AS ?birthdate )
  BIND( xsd:decimal( sgfn:CSV(?person, "Height") ) AS ?height )
  BIND( xsd:decimal( sgfn:CSV(?person, "Weight") ) AS ?weight )
}
```
From this example, it can be observed that SPARQL-Generate is very similar to SPARQL 1.1, although an extension of it. Furthermore, it seems to be not straightforward on how a custom user-defined function can be added in the mapping. For example, given a map of values to events in the ontology, such as ("100" mapped to HeadingChange, and "001" mapped to TakeOff), mapping any combination/aggregation of these values (e.g. the value "101") to both HeadingChange and TakeOff events, may not be trivial.

The corresponding RDF-Gen transformation/mapping specification (graph template) is provided in the following lines:

```rml
//template.rdf
turtle
asURI(?PersonId) a foaf:Person ;
  schema:birthDate asDateTime(?Birthdate) ;
  schema:height ?Height ;
  schema:weight ?Weight ;
  foaf:mbox asQuotedURI (?Email) ;
  foaf:name asString (?Name) ;
  foaf:phone asQuotedURI (?Phone) .
```

It can be observed that the RDF-Gen mapping specification is considerably more compact and simple compared to the correspondent RML and SPARQL-Generate mappings. As we can observe in this example, custom functions can be used both in subject and object placeholders in the triple templates. A set of commonly used functions is available, e.g. `asDateTime(?Birthdate)` that converts a date entry to a valid `xsd:dateTime` literal, which can be easily extended according to the application/user needs. The prefix specification is not required in the mapping specification. Unless specified in a custom function, URIs will be constructed under the base namespace.

The RDF-Gen allows the automatic validation of the generated triples using open source API Jena. The validation process uses the turtle (TTL) parser implemented in Jena, and validates the syntax of the generated triples. For large volumes of data, this option may be only activated on a user-defined data sample. The specification of the generated triples. For large volumes of data, this option may be only activated on a user-defined data sample.

In the following section, in order to show the scalability of the proposed method compared to recent approaches. For this purpose we have used real world data sets using the minimum possible user-defined functions towards a fair comparison between approaches, without employing any technical optimizations such as parallel processing or caching. The comparison between mappings used in each approach, as shown in this section, also verifies that RDF-Gen provides the most compact and less-verbose mapping, allowing users to inspect, modify and verify the mapping easily, even for big volume of data.

## 5 CONCLUSIONS

This paper presents a new approach towards generating RDF knowledge graphs from multiple heterogeneous streaming and archival data, in a uniform, efficient and scalable way. Separating the Data Connector from the Triple Generator, the RDF-Gen approach outperforms the state of the art tools RML and SPARQL-Generate, in terms of throughput, scalability and usability. This is achieved by implementing data access and close-to-the-sources data processing facilities in the Data Connectors, providing data in a record-by-record approach to the Triple Generators, which use graph templates as a generic way to map data to RDF.

In addition to the reported evaluation, compared to RML, RDF-Gen needs no further knowledge of a specific vocabulary, and it can be used by anyone who can write simple SPARQL queries. Furthermore, compared to SPARQL-Generate, it requires no underlying SPARQL engine, and it inherently supports distribution of processing and the exploitation of streaming data sources.

On the other hand, as it also holds for the other RDF generators, RDF-Gen instantiations require human effort for the specification of graph templates and configuration files. Although currently there is no any approach to fully overcome this limitation, introducing a fully automated construction of the mappings/templates, there are approaches (e.g. KARMA [6]) that provide a set of suggestions of data-to-vocabularies mappings. RDF-Gen enhancements aim to incorporate a new component for suggesting variables as well as bindings to the available data.

### ACKNOWLEDGMENTS

This work is supported by project datAcron, which has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 687591.

### REFERENCES

1. ConverterToRdf 2017. https://www.w3.org/wiki/ConverterToRdf (2017). Accessed: 2017-12.
2. Ben De Meester, Wouter Maroy, Anastasia Dimou, Ruben Verborgh, and Erik Mannens. 2017. Declarative Data Transformations for Linked Data Generation: The Case of DBpedia. In Proceedings of the 14th ESWC (Lecture Notes in Computer Science), Eva Blomqvist, Diana Maynard, Aldo Gangemi, Rik Van de Walle. 2014. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In Proceedings of the 7th Workshop on Linked Data on the Web (CEUR Workshop Proceedings). Christian Bizer, Tom Heath, Sören Auer, and Tim Berners-Lee (Eds.), Vol. 10250. Springer, 33–48. https://doi.org/10.1007/978-3-319-58451-5_3
3. Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. 2014. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In Proceedings of the 7th Workshop on Linked Data on the Web (CEUR Workshop Proceedings). Christian Bizer, Tom Heath, Sören Auer, and Tim Berners-Lee (Eds.), Vol. 10250. Springer, 33–48. https://doi.org/10.1007/978-3-319-58451-5_3
4. Ademar Crotti Junior, Christophe Debruyne, and Declan O’Sullivan. 2016. 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 (iiWAS ’16). ACM, New York, NY, USA, 267–275. https://doi.org/10.1145/3011141.3011152
5. Ademar Crotti Junior, Christophe Debruyne, and Declan O’Sullivan. 2016. Incorporating Functions in Mappings to Facilitate the Uplift of CSV Files into RDF. In The Semantic Web - ESWC 2016 Satellite Events, Heraklion, Crete, Greece, May 29 - June 2, 2016, Revised Selected Papers. 55–59. https://doi.org/10.1007/978-3-319-47602-5_12
6. Craig A. Knoblock, Pedro Szekely, José Luis Ámbite, Amael Goel, Shubham Gupta, Kristina Lerman, Maria Muslea, Mohsen Taherzadeh, and Parag Mallick. 2012. Semi-automatically Mapping Structured Sources into the Semantic Web. In The Semantic Web Research and Applications, Elena Simperl, Philipp Cimiano, Axel Polleres, Oscar Corcho, and Valentina Presutti (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 375–390.
7. Kostas Kyzirakos, Ioannis Vlahopoulos, Dimitrios Savva, Stefan Manegold, and Manolis Koubarakis. 2014. GeoTriples: A Tool for Publishing Geospatial Data As RDF Graphs Using R2RML Mappings. In Proceedings of the 2014 International Conference on Posters & Demonstrations Track - Volume 1272 (ISWC-PD’14). 267–275. https://doi.org/10.1145/2678455.2678552
8. Maxime Lefrançois, Antoine Zimmermann, and Noorani Bakerally. 2017. A SPARQL Extension for Generating RDF from Heterogeneous Formats. In The Semantic Web, Eva Blomqvist, Diana Maynard, Aldo Gangemi, Rik Veheokstra, Pascal Hitzler, and Olaf Hartig (Eds.). Springer International Publishing, Cham, 35–56.

---

available at https://jena.apache.org
