UNIVERSITEIT
GENT

Faculty of Sciences

Stream join processing in RDF mapping engines

by

Sitt Min Oo
Student number: 01503244

Supervisors: Prof. Dr. Ruben Verborgh and Dr. Anastasia Dimou
Counsellor: Gerald Haesendonck

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science

Academic Year 2020–2021

Preface

"The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In all cases of other use, the copyright terms have to be respected, in particular with regard to the obligation to state explicitly the source when quoting results from this master dissertation."

Acknowledgement

Sitt Min Oo, December 2020

Stream join processing in RDF mapping engines

by

Sitt Min Oo

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Computer Science

Academic year 2020–2021

Supervisor: Prof. Dr. Ir. Bjorn De Sutter
Counsellor: Dr. Tim Besard

Faculty of Sciences
University Ghent

Abstract

Here comes abstract.

Keywords

RDF, RMLStreamer, RML, Adaptive windows, Stream joins.

Contents

Chapter 1

Introduction

A large volume of data is generated daily on the Web in a variety of domains. These data are often structured according to an organization's specific needs or formats: Leading to a difficulty in integrating the data across the different applications. These generated data might have to be associated with archival data, also of heterogeneous formats, to provide a coherent view required by analysis tasks. Heterogeneous Web data formats, such as CSV or HTML, are not explicitly defined to enable linking entities in one document to other related entities in external documents.

Based on W3C standard, semantic data formats such as RDF triples [1], are a solution to this particular problem by enriching the data with knowledge and association across different domains, through the use of common ontologies. RDF triples also form the basic building blocks of knowledge graphs. Knowledge graphs are extensively used in social networks like Facebook[2], IoT devices[3] and especially with Google's search engine[4], it enables machines to understand the data and perform complex automated processing using the knowledge graphs.

Considering the aforementioned scenarios, there is a need to transform these non-RDF data to RDF compliant formats on the fly while new data are being generated. Furthermore, we would also like to apply stream operators on the input tuples before transforming, to enhance the enrichment of the data.

There exists state-of-the-art techniques to solve the task of consolidating heterogeneous data and transforming them to an RDF compliant format. In this thesis, we will focus on one such format called TURTLE [5]. These RDF transformation engines can be categorized into two major categories based on the type of input which they consume; bounded and unbounded data input. Since we are focussing on the generation of RDF data in a streaming environment, the class of

RDF transformation engines on unbounded data will be of interest to our study.

Some engines support traditional stream operators like joins and aggregations. However, they do not consider the characteristics of the streaming sources such as velocity and time-correlations between the different input streams. This leads to a decline in the quality of the generated RDF triples. Moreover, due to the nature of the infinite, continuous and real-time changing data of the streaming environment, these operators have to be applied in the context of windows over a subset of the incoming data. Clearly, with these restrictions and characteristics of the streaming sources, we need an adaptive approach to applying these operators in windows for the data transformation engines.

Chapter 2

Semantic Web Technologies

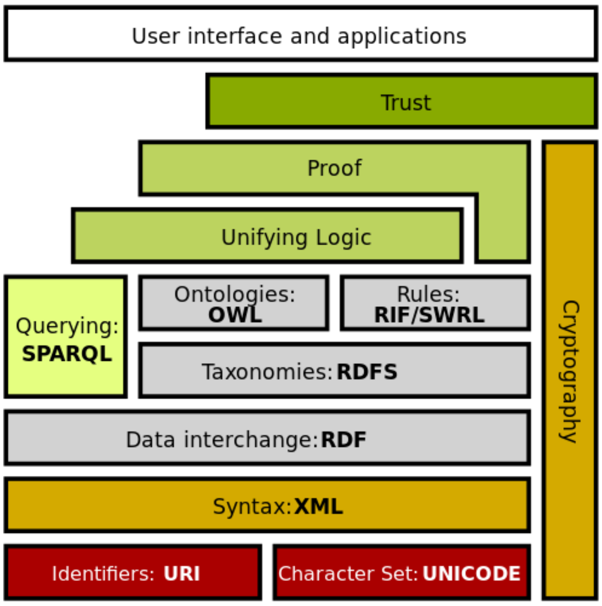

Figure 2.1: An overview of semantic web stack with core technologies[6].

Semantic web extends the existing *web of documents* with the ability to *link* different documents and *embed knowledge* in the document to transform it into a *web of data*. It is perceived by Tim Beners-Lee in 2001[7] to be integrated into the existing architecture of World Wide Web. Embedding *knowledge* into documents enables machines to interpret the *meaning* of the document and interoperate with each other on a more complex level.

Semantic web stack enables one to start building a *web of data* using the core technologies shown in Figure 2.1. However, this is provided that the data already exists in RDF compliant format. Existing data on the web in non-RDF format must, therefore, be transformed into RDF data enable transition to a *web of data*. This work focuses on transforming non-RDF to RDF data, thus the RDF format will be elaborated in detail.

2.1 RDF

Resource Definition Framework [8] is a framework for representing data on the Web. It portrays the data as a directed graph with the resources as nodes in the graph and the edges as the relationship between the different resources. Figure 2.2 shows an example of an RDF triple statement describing the information "John has an apple". The triple statement consists of the subject *John*, the predicate *has* and the object *apple*.

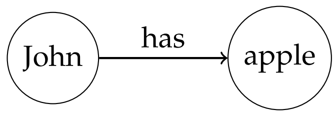

Figure 2.2: An RDF triple representing the information "John has an apple".

By composing these simple triple statements into a set of RDF triples, it yields us an RDF graph. In Figure 2.3, 4 triple statements are composed together to form a simple RDF graph describing *John* and *Mary* having the same *apple*. It might not be evident from the simple figures, about the advantages of RDF graphs. Data representation in a graph model allows machines to follow the *links* between the resources, and discover more unknown data in the linked knowledge graph. Link following is possible due to the nodes in the triples being classified as one of the 3 different term types.

2.1.1 Term types

Resources are classified into 3 different term types; IRI (Internationalized Resource Identifier), literals and blank nodes. IRI is a string identifier unique in the global scope to represent a resource. It is usually in the form of a web address, however, it can also be in other forms so long as it conforms to the syntax defined in RFC 3987 [1]. IRI can represent a relationship, a concept or an object. Therefore, it could be used in the *subject*, the *predicate* and the *object* components of an RDF triple.

Literals term type is used to represent a value such as strings, numbers, boolean and dates. To ensure that the machines know the type of the data being read, we could explicitly specify

[1]RFC 3987: https://www.ietf.org/rfc/rfc3987.txt

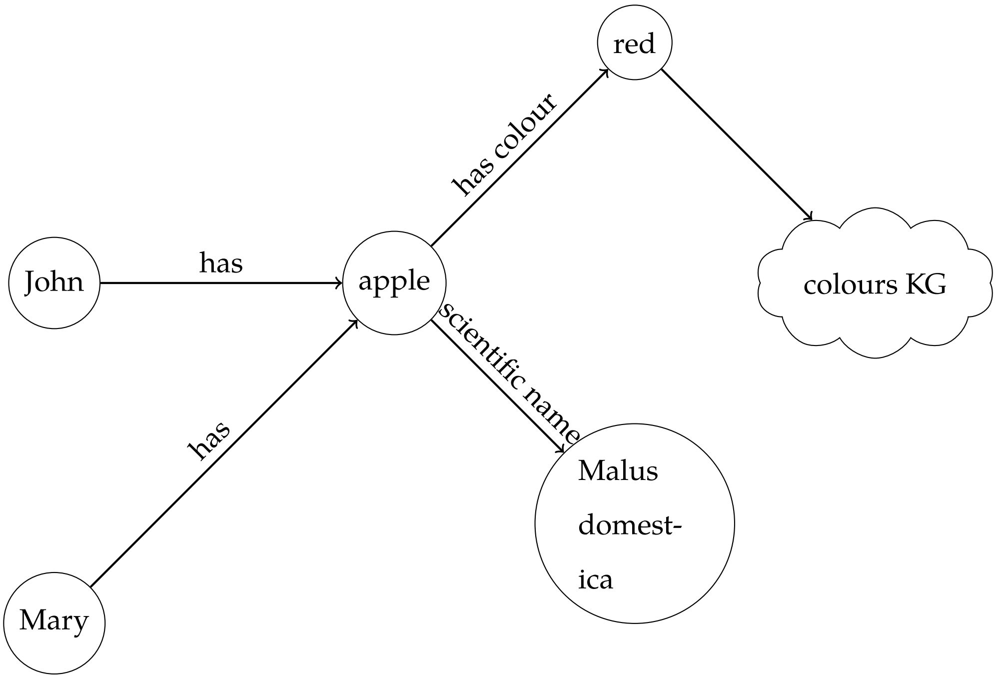

Figure 2.3: A simple RDF graph where the same "apple" is shared by both John and Mary.

the type of the data with a datatype IRI. Moreover, the language in which the data is written, could also be explicitly stated with a language tag.

Lastly, we have the term type blank nodes. Blank nodes identify resources in the local scope (i.e. to a local file or an RDF store). Since it is used to identify resources in nodes, blank nodes are only applicable as the *subject* and the *object* components of an RDF triple.

2.1.2 TURTLE syntax

The aforementioned term types are defined for abstract RDF syntax. For a concrete syntax to write RDF triples, we would focus on the TURTLE (Terse RDF Triple Language) [5] syntax in this work. A simple triple statement is a sequence of *subject, predicate, object* terms, ending in a '.'. To reduce the repetition of writing the same subject and predicate combination with different objects, TURTLE allows the use of ',' to separate the different objects. Additionally, one could also use ';' to separate the different predicates and objects sharing the same subject.

Listing 2.1: Usage of ';' where triples share the same subject.

```
<http://example.org/apple>  <http://example.org/hasColor> "red";
                            <http://example.org/scientificName> "Malus domestica".
```

Listing 2.2: Usage of ',' where triples differs only in the objects.

```
<http://example.org/apple> <http://example.org/scientificName> "Malus pumila",
                                                                "Malus domestica".
```

IRIs are written between the angle brackets like <http://example.org/#John>. Since blank nodes are locally scoped version of IRIs, the same syntax to write IRIs is also used. TURTLE syntax allows us to define *prefixes* at the head of the TURTLE file. Users could then use prefixes, to write RDF triples in a more compact form. For example, <http://example.org/#John> could be shortened to <#John> using the relative @*base* path.

Listing 2.3: Prefixes in TURTLE syntax.

```
    @base <http://example.org/> . # default base IRI
    @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
    @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
    @prefix foaf: <http://xmlns.com/foaf/0.1/> .
    @prefix rel: <http://www.perceive.net/schemas/relationship/> .
    ...
```

Literals are written between the double quote '""'. It has a default datatype of a string. One could also *cast* the literal value to a specific datatype by appending '^^ [IRI of the datatype]'. For example, "12" is cast to an integer in "12"^^xsd:integer. Language of the literal value could also be specified using @ similarly to datatype casting.

Chapter 3

Data Stream Processing

Chapter 4

RDF data generation from non-RDF data

Several state-of-the art implementations exist to generate RDF data from non-RDF data. These engines could be categorized into two groups; *query-based* and *rule-based* engines. These can be further categorized into two more subgroups based on the data that they consume; *bounded* and *unbounded* data processing engines. As mentioned in Chapter 1, this work will focus on engines working with unbounded data in a streaming environment, thus engines consuming bounded data will not be elaborated. Before we dive into the engines, we need to elaborate more on the *languages* these engines use to transform non-RDF to RDF data.

4.1 SPARQL Query Language

Query languages already exist in established relational database systems such as MySQL or PostgreSQL. It allows users to manipulate and retrieve data from relational databases using concise statements. Due to its widespread usage in the industry for querying databases, it is important that a similar query language is used for RDF datasets to ease the transition for the users.

SPARQL[9] achieves this goal by emulating as many SQL syntaxes as possible to allow a seamless transition to RDF datasets usage by existing data engineers. Similar to the relational databases, RDF datasets can be considered as a table consisting of three columns — the subject column, the predicate column, and the object column. Unlike relational databases, RDF datasets, in a table representation, allows the object column to be of heterogeneous datatype. Recall from Chapter 2.1.2, one could explicitly specify the datatype of the object term which allows the heterogeneity in the object column.

Also, different from SQL, SPARQL allows matching based on *basic graph patterns* composed of a set of *triple patterns*. Triple patterns are similar to the triple statements as clarified in Chapter 2.1.2 but extended with declared variables (i.e. *?variable_name*). The declared variables are then bound to the concrete value in the corresponding *triple statements*, matching the given *triple pattern*, from the RDF dataset. The result of a SPARQL query is returned as an RDF sub-graph of the queried RDF dataset.

Now a question definitely gets raised in our mind, how does this relate to transforming an unbounded dataset to RDF format in a streaming environment? There exists state-of-the-art engines for generating RDF data from heterogeneous streaming data sources, which will be elaborated in Chapter 4.3.

Listing 4.1: Example of a SPARQL query of a medication.

```
SELECT ?medication
WHERE {
  #Basic graph pattern consisting of 2 triple patterns.
    ?diagnosis example:name ”Cancer” .
    ?medication example:canTreat ?diagnosis .
}
```

medication
Radiation therapy

Table 4.1: Result of executing the SPARQL query in Listing 4.1

4.2 RDF Mapping Language

RDF Mapping Language[10] is a superset of the W3C's R2RML[11] which maps relational databases to RDF datasets. RML improves upon R2RML by expressing mapping rules from heterogeneous data sources and transforming them to RDF datasets whereas R2RML could only consume data from relational databases. RML mapping file is composed of one or more *triples maps*, which in turn consist of *subject*, *predicate* and *object* term maps. As the names imply, the term maps are used to map elements of the data sources to their respective terms in an RDF triple. The definitions of these maps are similar to the specifications in R2RML[12].

Logical sources could be defined by specifying the *source, logical iterator* and zero or one *reference formulation* property. The logical sources in the default RML mapping file are bounded data, where the data already exists and has a predetermined size.

RML also supports defining relationships amongst the different triple maps through the use of *rr:parentTriplesMap*, *rr:joinCondition*, *rr:child and rr:parent* properties. Different triple maps might come from separate logical sources, therefore, referencing across these triple maps will require applying the join operator across multiple logical sources.

Listing 4.2: An example of an RML mapping file[12].

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix rml: <http://semweb.mmlab.be/ns/rml#>.
@prefix ex: <http://example.com/ns#>.
@prefix ql: <http://semweb.mmlab.be/ns/ql#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@base <http://example.com/ns#>.

<#TransportMapping>
  rml:logicalSource [
    rml:source "Transport.xml" ;
    rml:iterator "/transport/bus/route/stop";
    rml:referenceFormulation ql:XPath;
  ];

  rr:subjectMap [
    rr:template
      "http://trans.example.com/stop/{@id}";
    rr:class ex:Stop
  ];

  rr:predicateObjectMap [
    rr:predicate rdfs:label;
    rr:objectMap [
      rml:reference "."
    ]
  ].
```

4.3 Query based Engines

When we think of interacting with a data source, querying the data source with a query language seems natural since that is the common method to interact with a traditional database. Allowing users to query the sources without explicitly defining the mapping semantics hides the mapping or data transformation details from the user. Moreover, it eases the transition to developing applications using RDF data since the syntaxes are similar to existing query languages.

From what we know, there exists two state-of-the-art query based engines for transforming unbounded non-RDF data to RDF data; SPARQL-Generate and RDF-Gen, the latter diverging a lot from the underlying SPARQL syntaxes. These engines will be elaborated more in the following sections.

4.3.1 SPARQL-Generate

SPARQL-Generate [13] was proposed as an alternative to then existing methods of transforming non-RDF data. The language is based on an extension of SPARQL 1.1 query language, to leverage its expressiveness and extensibility. Furthermore, this allows SPARQL-Generate to be implemented on top of any existing SPARQL query engines. The reference implementation in the paper was based on Apache Jena's SPARQL 1.1 engine. Due to the use of existing SPARQL 1.1 query language, experienced knowledge engineers could use SPARQL-Generate to improve the generation of RDF data in their existing workflow.

SPARQL-Generate supports the consumption of heterogeneous data sources by exposing the *binding* and *iterator functions API*. Therefore, covering a new data format or data source could be accomplished by implementing the corresponding *binding* and *iterator functions*. Currently, as of writing this paper, the reference implementation supports the consumption of data sources which are unbounded and in a streaming environment. For example, WebSocket and MQQT are currently supported in the latest version of SPARQL-Generate.

Although there is a support for data stream processing, the paper did not go into details about the application of multi-stream operators like joins when involving multiple streaming data sources. Hence, we could assume that SPARQL-Generate either process the whole dataset in memory or keeps a fixed window of data on which multi-stream operators are applied. Furthermore, the authors mentioned that SPARQL-Generate could use the SPARQL 1.1 operators such as *join operators*. From this, we could derive that SPARQL-Generate will apply *join* on the records by delegating it to the underlying SPARQL engine. Therefore, there is a lack of dynamic windowing schemes to efficiently exploit the characteristics of the streaming data sources.

4.3.2 RDF-Gen

RDF-Gen[14] is also based on SPARQL-like syntaxes. However, instead of extending SPARQL engines, it provides its own architecture as laid out in Figure 4.1 to meet the demand of real-time processing. Instead of adopting most of the syntaxes from SPARQL, RDF-Gen only kept the BGP section of SPARQL query to reduce the size of the transformation specification. Thus, it has the most compact mapping template/query compared to the other methods mentioned in this paper.

RDF-Gen consists of three main components: (a) Data Connector, (b) Triple Generator, and (c) Link Discovery. Data Connector allows close-to-source processing, which is not the case for SPARQL-Generate. Triple Generator handles the rapid generation of RDF triples by making use of template graphs and variable vectors. The Link Discovery component solves the problem of link discovery problem which is defined as follows: Given two data sources S and T, the problem is to find the pairs of elements in $S \times T$ that are related to each other (e.g. following the predicate of *owl:sameAs* property). Since we are concerned with multi-stream operators during RDF generation in this paper, link discovery component could be ignored.

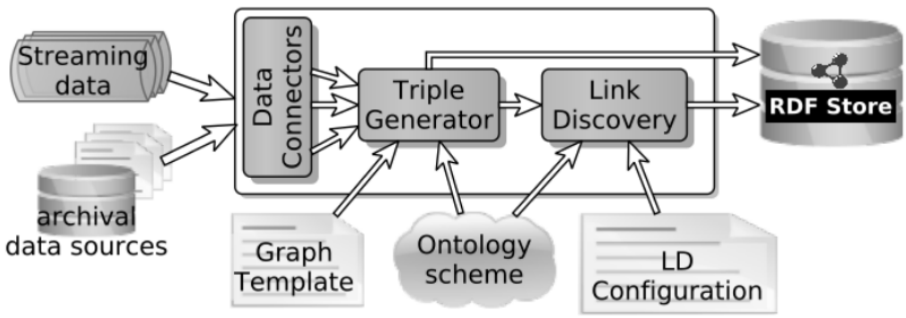

Figure 4.1: Architecture of RDF-Gen[14]

Data Connector

Data Connector has a similar functionality as the *iterator functions* from SPARQL-Generate. It consumes the data sources given a configuration setting. Configuration setting is used to specify the type of data sources, the *window* for processing the incoming data records and also apply functions on the incoming data elements. Data Connector can thus be defined as a mapping function as in Definition 4.1.

Definition 4.1 (Data Connector record [14]). Given a set of data sources $D = \{d_1, d_2, \ldots, d_n\}$ and a mapping function $F = \mu_f(d_i, e) \mid \forall d_i \in D$ with e, a data element of a data source d_i, and f, a filter function. Data Connector generates a record $R = \mu_f(d_i, e) \iff$ all the attributes of e satisfy the filter function f. By default, the filter function f just returns true.

Using the Definition 4.1, we could now also apply an equi-join operator on the data sources. Formally, we could generate a new triple $R = \mu_{f_i}(d_i, e_i) \bowtie \mu_{f_j}(d_j, e_j)$ where e_i and e_j have common attributes under the filter functions f_i and f_j.

The processing is done on individual records, leading to RDF-Gen treating streaming and archival data sources the same way — as "streams" of records. Due to the record-by-record processing, the framework also has a very low memory usage.

Triple Generator

As indicated in Figure 4.1, Triple Generator consumes the output records of the Data Connectors to convert them into RDF triples. A vector of variables V, an RDF graph template G, akin to the basic graph pattern from SPARQL 1.1, and a set of functions can be used to configure the Triple Generators.

V consists of variables which corresponds to the attributes of the generated records from the Data Connector. These variables are referenced in the graph template G, and then used to bind to the attribute value of the record provided by the Data Connector. Therefore, this simple binding of values in a template graph enables Triple Generator to generate RDF triples efficiently and have a high scalability. Generally, to keep the computational time low, the functions will have to be simple in complexity.

Next, we shall work on a small example to understand how Triple Generator works. Listing 4.3 shows an example of a graph template G provided to the Triple Generator to generate RDF triples. In this example, the provided vector of variables is $V = [\text{?diagnosis\_id}, \text{?name}]$. If the incoming record is as shown in Table 4.2, the specified variables, *diagnosis_id* and *name*, will be bound to the values *100* and *Cancer* respectively. Afterwards, the functions *makeUri* and *asString* will be called with the bounded values as arguments and the generated output will be used to generate the RDF triples specified by the graph template. The final generated set of RDF triples is (e.g. in turtle syntax):

```
...
<http://example.com/100>  a example;
                          example:name  "Cancer".
...
```

diagnosis_id	name
100	Cancer

Table 4.2: A sample record generated by the Data Connector.

Listing 4.3: A simple graph template *G* with the functions *asString* and *makeUri*.

```
#BGP for the diagnosis data source
makeUri(?diagnosis_id)  a example:Diagnosis;
                        example:name asString(?name).
```

4.4 Mapping based Engines

Other than approaching the transformation of non-RDF to RDF from the viewpoint of queries, one could also employ mapping languages such as RML. The following subsections will elaborate more on the related state-of-the-art engines which utilizes mapping languages in a streaming environment.

4.4.1 TripleWave

Albeit the abundance of solutions to combine semantic technologies with stream and event processing techniques, there was a lack of engines to disseminate and exchange RDF streams on the Web [15]. TripleWave[15] was conceived to fill this role; to provide the mechanism to publish and spread RDF streams on the Web. It extends R2RML, which only allows ingestion of inputs from relational databases, to also consume other formats such as JSON or CSV (just like RML, a superset of R2RML).

TripleWave generates an RDF stream of JSON-LD format which could be consumed by existing RSP engines for further processing. It also supports joining of multiple stream which could be inferred from its usage of R2RML's usage of *rr:parentTriplesMap* predicate [15].

4.4.2 RMLStreamer

[16]

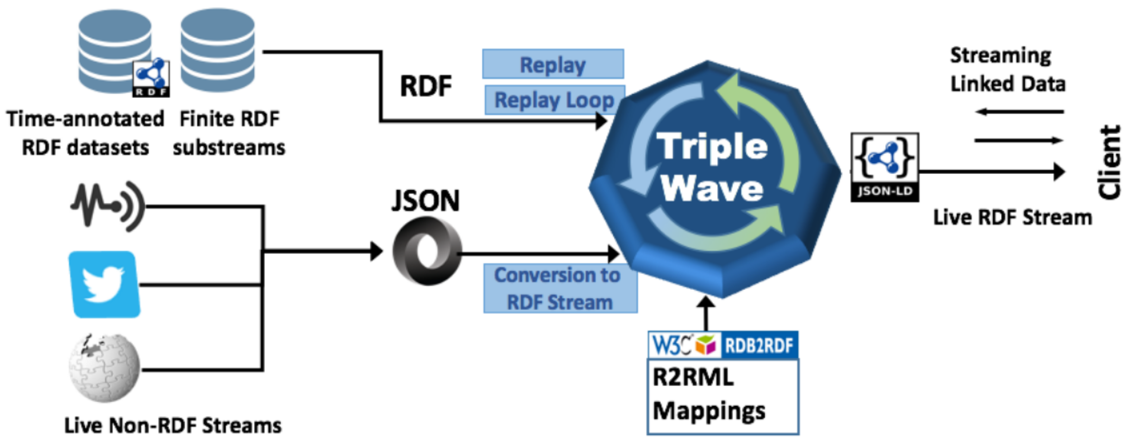

Figure 4.2: Architecture of TripleWave generating RDF streams from non-RDF and RDF data sources[15].

Bibliography

[1] E. Miller, 'An introduction to the resource description framework,' *Bulletin of the American Society for Information Science and Technology*, vol. 25, no. 1, pp. 15–19, 1998. DOI: https://doi.org/10.1002/bult.105. eprint: https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/bult.105. [Online]. Available: https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/bult.105.

[2] J. Weaver and P. Tarjan, 'Facebook linked data via the graph api,' *Semantic Web*, vol. 4, pp. 245–250, 2013.

[3] D. Phuoc, H. Nguyen Mau Quoc, H. Ngo, T. Nhat and M. Hauswirth, 'The graph of things: A step towards the live knowledge graph of connected things,' *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 37, pp. 25–35, Mar. 2016. DOI: 10.1016/j.websem.2016.02.003.

[4] A. Singhal. (2012). 'Introducing the knowledge graph: Things, not strings,' [Online]. Available: https://blog.google/products/search/introducing-knowledge-graph-things-not/ (visited on 25/12/2020).

[5] E. Prud'hommeaux and G. Carothers, 'RDF 1.1 turtle,' W3C, W3C Recommendation, Feb. 2014, https://www.w3.org/TR/2014/REC-turtle-20140225/.

[6] W. Commons. (2014). 'Semantic web stack.' File: `Semantic Web Stack.svg`, [Online]. Available: https://commons.wikimedia.org/wiki/File:Semantic_web_stack.svg.

[7] T. Berners-Lee, J. Hendler and O. Lassila, 'The semantic web,' *Scientific American*, vol. 284, no. 5, pp. 34–43, May 2001. [Online]. Available: http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.

[8] M. Lanthaler, D. Wood and R. Cyganiak, 'RDF 1.1 concepts and abstract syntax,' W3C, W3C Recommendation, Feb. 2014, https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[9] A. Seaborne and E. Prud'hommeaux, 'SPARQL query language for RDF,' W3C, W3C Recommendation, Jan. 2008, https://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[10] A. Dimou, M. V. Sande, P. Colpaert, R. Verborgh, E. Mannens and R. Walle, 'Rml: A generic language for integrated rdf mappings of heterogeneous data,' in *LDOW*, 2014.

[11] R. Cyganiak, S. Das and S. Sundara, 'R2RML: RDB to RDF mapping language,' W3C, W3C Recommendation, Sep. 2012, https://www.w3.org/TR/2012/REC-r2rml-20120927/.

[12] A. Dimou, M. V. Sande, B. D. Meester, P. Heyvaert and T. Delva, 'Rdf mapping language (rml),' IDLab - imec - Ghent University, Specification document, Oct. 2020. [Online]. Available: https://rml.io/specs/rml/.

[13] M. Lefrançois, A. Zimmermann and N. Bakerally, 'A sparql extension for generating rdf from heterogeneous formats,' in *The Semantic Web*, E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler and O. Hartig, Eds., Cham: Springer International Publishing, 2017, pp. 35–50, ISBN: 978-3-319-58068-5.

[14] G. Santipantakis, K. Kotis, G. Vouros and C. Doulkeridis, 'Rdf-gen: Generating rdf from streaming and archival data,' Jun. 2018, pp. 1–10, ISBN: 978-1-4503-5489-9. DOI: 10.1145/3227609.3227658.

[15] A. Mauri, J.-P. Calbimonte, D. Dell'Aglio, M. Balduini, M. Brambilla, E. Della Valle and K. Aberer, 'Triplewave: Spreading rdf streams on the web,' Oct. 2016, pp. 140–149, ISBN: 978-3-319-46546-3. DOI: 10.1007/978-3-319-46547-0_15.

[16] G. Haesendonck, W. Maroy, P. Heyvaert, R. Verborgh and A. Dimou, 'Parallel rdf generation from heterogeneous big data,' Jul. 2019, pp. 1–6. DOI: 10.1145/3323878.3325802.

Appendices

List of Listings

List of Figures

List of Tables