Leveraging Web of Things W3C recommendations for knowledge graphs generation

Dylan Van Assche(✉)[0000−0002−7195−9935], Gerald Haesendonck[0000−0003−1605−3855], Gertjan De Mulder[0000−0001−7445−1881], Thomas Delva[0000−0001−9521−2185], Pieter Heyvaert[0000−0002−1583−5719], Ben De Meester[0000−0003−0248−0987], and Anastasia Dimou(✉)[0000−0003−2138−7972]

IDLab, Department of Electronics and Information Systems, Ghent University – imec
Technologiepark-Zwijnaarde 122, 9052 Ghent, Belgium
`{firstname.lastname}@ugent.be`

Abstract. Constructing a knowledge graph with mapping languages, such as RML or SPARQL-Generate, allows seamlessly integrating heterogeneous data by defining access-specific definitions for e.g., databases or files. However, such mapping languages have limited support for describing Web APIs and no support for describing data with varying velocities, as needed for e.g., streams, neither for the input data nor for the output RDF. This hampers the smooth and reproducible generation of knowledge graphs from heterogeneous data and their continuous integration for consumption since each implementation provides its own extensions. Recently, the Web of Things (WoT) Working Group released a set of recommendations to provide a machine-readable description of metadata and network-facing interfaces for Web APIs and streams. In this paper, we investigated (i) how mapping languages can be aligned with the newly specified recommendations to describe and handle heterogeneous data with varying velocities and Web APIs, and (ii) how such descriptions can be used to indicate how the generated knowledge graph should be exported. We extended RML's Logical Source to support WoT descriptions of Web APIs and streams, and introduced RML's Logical Target to describe the generated knowledge graph reusing the same descriptions. We implemented these extensions in the RMLMapper and RMLStreamer, and validated our approach in two use cases. Mapping languages are now able to use the same descriptions to define the input data but also the output RDF. This way, our work paves the way towards more reproducible workflows for knowledge graph generation.

1 Introduction

Mapping languages, such as the RDF Mapping Language (RML) [6], allow defining mapping rules to describe how to generate a knowledge graph from heterogeneous data. This is achieved by aligning the mapping rules with access-specific definitions for e.g., databases or files, to integrate data from heterogeneous formats, e.g., CSV, XML, JSON. However, we observe that: (i) data velocity is not well supported in mapping languages and corresponding processors, compared to data variety and volume [8,18,20]; and, (ii) the characteristics and destination of the generated knowledge graph remain unexplored.

Mapping languages declaratively describe how to integrate heterogeneous data without considering their data velocity, e.g., when new data is available for retrieval. Consequently, processors cannot generate knowledge graphs from data sources with varying data velocities, such as streams, as they lack the descriptions that determine their execution. This results in non-reproducible knowledge graph generation, because processors have each their own (use case-depending) approach to deal with data sources with varying data velocities.

Mapping languages only partly align with Web APIs and streams descriptions. When they do, they are limited to a set of protocols and do not describe how authentication against Web APIs and streams should be performed. Existing approaches describe access Web APIs, but only for a subset of the HTTP protocol, to retrieve data from Web APIs, while other protocols, e.g., MQTT or CoAP, and use cases of Web APIs are not considered. Because of this, additional steps outside the processor are needed to use other protocols. If authentication is needed, data cannot be retrieved from Web APIs, as the processors do not know how to handle authentication from the access description in mapping rules.

Last, mapping languages only define how a knowledge graph should be generated from heterogeneous data, but not how a knowledge graph should be exported and handled afterwards. Each processor has its own approach to retrieve this information, using e.g. a configuration file or command line arguments as this information is not declaratively described in the mappings. Furthermore, the velocity of the input data, also influences the output velocity when exporting knowledge graphs. Thus, it is necessary to consider the data velocity when retrieving the input data as well when exporting a knowledge graph.

We address the aforementioned issues by leveraging the recent W3C recommendations of the Web of Things (WoT) Working Group [2,10,13]. On one hand, we adapt the data source descriptions to describe how processors can *access and process Web APIs and streams with the WoT W3C recommendations*. On the other hand, we introduce a *target description* which declaratively describes how a knowledge graph should be exported. The target description defines in which format and where the knowledge graph is exported. Since the target description reuses same access descriptions as the input data sources, the generated knowledge graph can be exported in various ways, e.g., file dumps or triple stores.

We apply our proposed approach to the RDF Mapping Language (RML) [6]. Our contributions are: (i) **RML's Logical Source adaptation to the new WoT W3C recommendations** to support more data structures, data velocity and authentication; (ii) **RML's Logical Target introduction** to define how the knowledge graph should be handled and exported; (iii) **Implementation of our proposed approach** in the RMLMapper[1] and RMLStreamer[2]; and (iv) **Validation of our approach** in two use cases: ESSENCE and DAIQUIRI.

[1] https://github.com/RMLio/rmlmapper-java
[2] https://github.com/RMLio/rmlstreamer

Lack of access to data with different velocities and knowledge graph's characteristics' descriptions hampers the knowledge graphs' *reproducible generation* from heterogeneous data. It also hampers their *continuous integration for consumption* as additional steps are needed to retrieve the data and transform these in an appropriate format. Our proposed approach shows how mapping languages can use same descriptions for input data and output knowledge graph. This reduces the processor's implementation costs, as the same descriptions are reused for both input and output and all processors follow the same descriptions, resulting in more reproducible knowledge graph generation.

Section 2 describes the state of the art and Section 3 our motivating use cases and issues encountered in our use cases. Section 4 explains how we aligned the WoT W3C recommendations with RML and how we implemented our approach in the RMLMapper and RMLStreamer. We validate our approach in Section 5 with two real-life use cases. In Section 6, we discuss conclusions and future work.

2 State of the Art

In this Section, we describe our related work (Section 2.1), and introduce the Web of Things W3C recommendations and RML (Section 2.2).

2.1 Related Work

We outline vocabularies (Table 1) to describe Web APIs and streams, and investigate how current approaches use these vocabularies to access Web APIs, deal with streams' varying velocities and export the generated knowledge graph.

Vocabularies for Web APIs and streams. Data sources on the Web come in various forms and protocols while sharing common practices for identifying resources or authentication schemes. Various vocabularies exist to describe access to Web APIs and streams, e.g., Hydra, DCAT, HTTP, VoCaLS, and OWL-S.

The Hydra vocabulary [14] is proposed by the Hydra W3C Community Group to describe Web APIs but it is not a W3C recommendation. The Hydra vocabulary describes Web APIs but does not describe how a processor must perform authentication against Web APIs or use protocol-specific features.

DCAT [16] is a W3C recommendation to describe data catalogs on the Web. DCAT only describes datasets in a DCAT data catalog without covering protocol-specific features or authentication.

The HTTP W3C vocabulary [11] describes the HTTP protocol and can be used to describe HTTP Web APIs. However, the HTTP W3C vocabulary is limited to a single protocol, namely HTTP, does not describe how processors must perform authentication against Web APIs, nor does it describe streams.

OWL-S is a W3C member submission to semantically describe Web services [17] such as Web APIs and streams. OWL-S consists of Service Profiles to describe what the service does, Service Models to specify how it works, and Service Grounding which describes how to access the service. OWL-S' Service

Grounding leverages the Web Services Description Language [4] to describe access to Web services. Although, OWL-S can describe access to Web services, it does not cover authentication and never became a W3C recommendation.

VoCaLS [22] is a vocabulary and catalog description for data streams. It extends the DCAT W3C recommendation to describe streams without being limited to a specific stream protocol. VoCaLS can be used to describe access to data streams, but not other Web APIs.

The Web of Things (WoT) W3C Working Group recently released recommendations for describing IoT devices on the Web [2, 10, 13] by providing an abstract layer to access Internet of Things (IoT) devices. WoT uses a similar approach as OWL-S by applying binding templates to bind this layer to an underlying protocol used by an IoT device. New protocols can be added by defining a new binding template without influencing the access abstraction layer [13]. We leverage the WoT W3C recommendations to showcase how processors can access Web APIs and streams without depending on a specific protocol.

Vocabulary	Protocol independent	Authentication	Web APIs	Streams	W3C Recommendation
Hydra	✓	✗	✓	✗	✗
DCAT	✗	✗	✓	✗	✓
HTTP W3C	✓	✗	✓	✗	✓
OWL-S	✗	✗	✓	✓	✗
VoCaLS	✓	✗	✗	✓	✗
WoT W3C	✓	✓	✓	✓	✓

Table 1. Existing vocabularies for describing access to Web APIs and streams.

Mapping languages. Existing mapping languages share same principles for describing input data sources by defining iterators and access descriptions for the data sources and leave the characteristics of exporting generated knowledge graphs up to the implementation. Most mapping languages, e.g., RML and SPARQL-Generate, reuse existing specifications e.g. R2RML and SPARQL respectively, to define a mapping language for generating knowledge graphs from heterogeneous data sources. RML [6] broadens the scope of R2RML [5] from relational databases to heterogeneous data sources using RML's Logical Source, while still being backwards compatible. SPARQL-Generate [15] extends SPARQL [9] instead of R2RML to integrate heterogeneous data sources into knowledge graphs with iterators to access and iterate over the data sources.

Such mapping languages describe how processors should access various heterogeneous data sources except for Web APIs and streams. RML leverages the Hydra vocabulary to provide access to Web APIs [7], SPARQL-Generate and xR2RML define each their own approach to accomplish this [15, 19]. However, they can only perform HTTP GET requests without authentication to retrieve data from the Web. D2RML argues it is needed to describe access to Web APIs in more detail [3] but D2RML can only describe HTTP requests using the W3C HTTP vocabulary, other protocols are not supported.

Data velocity. In recent years, there has been an increasing interest in generating knowledge graphs from data with different velocities than static data, such as data streams. Several approaches were introduced, for example: TripleWave [18], RDF-Gen [20], SPARQL-Generate [15], and Chimera [21]. However, these approaches do not declaratively describe how different data velocities must be handled during the knowledge graph generation.

TripleWave uses R2RML mappings for specifying the subject, predicate and object of the generated RDF triples, but handles the data velocity problem in its processor through a wrapper. This wrapper is mostly use case specific and not reusable for other use cases. SPARQL-Generate provides access to data streams, but delegates the processing and handling of the different data velocities to the underlying SPARQL engine. RDF-Gen claims it can access and process data streams but does not mention how different data velocities are handled. CARML[3] also access streams by extending RML with its own extension, a single access description for streams (`carml:Stream`) but only describes the name of the stream to use. Recently, a data transformation framework Chimera was proposed [21] which allows to uplift data into a knowledge graph using RML and lower this knowledge graph later on in various data formats through Apache Velocity templates and SPARQL queries [21]. Chimera leverages Apache Camel's Routes [21] for constructing its data processing pipelines. Because of this, Chimera can have multiple input and output channels and access data sources included in the Apache Camel framework, such as Web APIs or streams. However, no declarative access description is available to describe Web APIs and streams; instead, the `rml:source` property in RML's Logical Source refers to a generic InputStream, an extension of RML used in Chimera, which only specifies the name of the InputStream to use as data source.

RDF output. Mapping languages has not yet determined how the serialisation, storage or velocity of the generated knowledge graph (output) should be handled, exported, and described. Each processor of a mapping language has its own way to handle the knowledge graph after its generation. Processors mainly use command line arguments (e.g., RMLMapper, RMLStreamer, SPARQL-Generate), or configuration files (e.g., RMLMapper, SPARQL-Generate, Chimera) to specify a single target such as a local file, access configuration of the SPARQL endpoint containing the knowledge graph, or Kafka stream.

Exporting knowledge graphs to multiple output targets is not considered by mapping languages, nor is generating a knowledge graph as a stream. While processors such as SPARQL-Generate [15], RDF-Gen [20], and TripleWave [18] can export their knowledge graphs as a stream during generation, these processors do not enrich existing knowledge graphs but recreate the knowledge graph from scratch when new data is retrieved. R2RML-Parser [12] avoids the former, but it

[3] https://github.com/carml/carml

only focuses on relational databases. In case multiple sets of targets are needed, the same mapping rules need to be executed multiple times, one set for each target. Since there is no declarative way for specifying where the output must be directed, processors cannot send parts of a knowledge graph to different or multiple output targets. Furthermore, these existing approaches lack the ability to describe if compression should be applied when exporting a knowledge graph.

2.2 Preliminaries

W3C Web of Things. A set of W3C recommendations were published by the W3C Web of Things Working Group for describing IoT devices and their capabilities such as interfaces, security, or protocols [2, 10, 13]. This way, machines can retrieve metadata about IoT devices (Listing 1.1 lines 3-4), understand how to interact with them (lines 8-10). The WoT W3C recommendations also describe which security practices must be applied when interacting with the IoT device (lines 5-6). These recommendations do not enforce a certain protocol, instead, they provide an abstraction layer that describes the protocol that must be used to interact with the device. External vocabularies, such as the W3C HTTP vocabulary [11], are leveraged to describe protocol-specific options. This way, new protocols can be added without changing the recommendation.

Listing 1.1. WoT Thing Description in JSON-LD for an MQTT illumance sensor

```
 1 {
 2   "@context": "https://www.w3.org/2019/wot/td/v1",
 3   "title": "MyIlluminanceSensor",
 4   "id": "urn:dev:ops:32473-WoTIlluminanceSensor-1234",
 5   "securityDefinitions": {"nosec_sc": {"scheme": "nosec"}},
 6   "security": ["nosec_sc"],
 7   "events": {  "illuminance": { "data":{"type": "integer"},
 8     "forms": [ {
 9       "href": "mqtt://example.com/illuminance", "contentType" : "text/plain",
10       "op" : "subscribeevent" } ] } }
11 }
```

RDF Mapping Language (RML). RML [6] broadens R2RML's scope and covers mapping rules from data in different (semi-)structured formats, e.g., CSV, XML, JSON which define how heterogeneous data is transformed in RDF.

Listing 1.2. RML mapping definitions

```
1 <#Mapping> rml:logicalSource <#InputX> ;
2   rr:subjectMap [ rr:template "http://ex.com/{ID}"; rr:class foaf:Person ];
3   rr:predicateObjectMap [ rr:predicateMap [ rr:constant foaf:knows ];
4     rr:objectMap [ rr:parentTriplesMap <#Acquaintance> ] ].
5 <#Acquaintance> rml:logicalSource <#InputY> ;
6   rr:subjectMap [ rml:reference "acquaintance"; rr:termType rr:IRI;
7     rr:class foaf:Person ] .
```

The main building blocks of RML are Triples Maps (Listing 1.2: line 1). A Triples Map defines how triples of the form subject, predicate, and object, will be generated. A Triples Map consists of three main parts: the Logical Source, the Subject Map, and zero or more Predicate-Object Maps. The Subject Map (line 2,

6) defines how unique identifiers (URIs) are generated for the mapped resources and is used as the subject of all RDF triples generated from this Triples Map. A Predicate-Object Map (line 3) consists of Predicate Maps, which define the rule that generates the triple's predicate (line 3) and Object Maps or Referencing Object Maps (line 4), which define how the triple's object is generated. The Subject Map, the Predicate Map, and the Object Map are Term Maps, namely rules that generate an RDF term (an IRI, a blank node or a literal). A Term Map can be a constant-valued term map (`rr:constant`, line 3) that always generates the same RDF term, or a reference-valued term map (`rml:reference`, line 6) that is the data value of a referenced data fragment in a given Logical Source, or a template-valued term map (`rr:template`, line 2) that is a valid string template that can contain referenced data fragments of a given Logical Source.

3 Motivation

In this Section, we introduce our motivating use cases, ESSENCE and DAIQUIRI (Section 3.1), and derive open issues (Section 3.2) with existing mapping languages which we encountered while trying to address these use cases.

3.1 Motivating use cases: ESSENCE & DAIQUIRI

We describe here our motivating use cases, ESSENCE and DAIQUIRI.

In ESSENCE[4], we had requirements related to data access, authentication and knowledge graph export during knowledge graph generation. ESSENCE focuses on data storytelling in smart cities with IoT sensors. These IoT sensors provides information about the weather or traffic in the city and their measurements are available through multiple Web APIs. We need to generate knowledge graphs from measurements of these sensors[5], such as rain sensors, water flow meters, and vehicle counters[6], and the generated knowledge graphs are published in a triple store. The measurements are available through various Web APIs, each with their own way of authentication. The generated knowledge graphs are exported to a triple store to be consumed by other partners, and are also stored locally to create backups.

In DAIQUIRI[7], we found requirements for data access, data velocity and exporting knowledge graph to various targets. DAIQUIRI is also a use case on data storytelling but for sports games, such as cycling or hockey. Athletes are tracked through sensors to provide sport analysts interesting facts in real time about the game. We integrate several sport tracking sensors into knowledge graphs and export these graphs for consumption. Data from these sensors are available from multiple infinite streams such as movement speed or heart rate.

[4] https://www.imec-int.com/en/what-we-offer/research-portfolio/essence
[5] https://open-livedata.antwerpen.be/#/org/digipolis/api/weerobservatiecutler-actuelewaarden/v1/documentation
[6] https://telraam-api.net/
[7] https://www.imec-int.com/en/what-we-offer/research-portfolio/daiquiri

Multiple types of tracking sensors are used. Consequently, each sensor has its own data velocity. While in ESSENCE, we exported the graphs to a triple store, in DAIQUIRI we export the generated knowledge graphs as an stream and create local backups on disk. The generated knowledge graph is continuously enriched.

3.2 Open Issues

In this Section, we describe open issues we encountered in our motivating use cases (Section 3.1). While these issues are inspired by our use cases, we generalize them in this Section aiming to tackle them with generic solutions. The Knowledge Graph Construction (KGC) Community Group also has a list of unsolved challenges for mapping languages. Several issues we encounter in our use cases were also highlighted by other researchers and companies[8]

Open Issue 1. Streams. Since mapping languages do not describe access to data with different velocities, processors implemented their own extensions, even for the same mapping language, e.g., RML[9]. We encountered this issue in our use cases when retrieving sensor measurements through Web APIs and streams which required a use case specific preprocessing step to overcome this obstacle. This issue is encountered and acknowledged by the KGC Community Group as well in their mapping challenges[10], verifying that this issue goes beyond our use cases. Mapping languages need to describe access to data with different velocities and indicate to processors how to handle data with different data velocities.

Open Issue 2. Web APIs. While mapping languages have preliminary support for Web APIs [7, 15, 19], they do not consider defining authentication, or protocol-specific features such as custom HTTP headers or other HTTP methods besides HTTP GET. As mentioned in Section 3.1, we encountered this issue when accessing Web APIs in our ESSENCE use case. These Web APIs required authentication with a custom HTTP header. We had to create a use case specific preprocessing step to authenticate with the Web APIs and retrieve the data. Mapping languages need to describe in detail how Web APIs must be accessed by processors to avoid such preprocessing steps.

Open Issue 3. Description of the generated knowledge graph. Mapping languages do not describe how a processor must export a knowledge graph. In both ESSENCE and DAIQUIRI, we had to store and publish the generated knowledge graph of sensor measurements. Thus, processors cannot determine from the mapping rules the serialization of a graph or where it must be exported. Therefore, we created a postprocessing step in our use cases to export the generated knowledge graph. There is a need for mapping languages to describe the characteristics of exporting a knowledge graph as RDF.

[8] https://github.com/kg-construct/mapping-challenges/issues
[9] CARML's Stream: https://github.com/carml/carml
RMLStreamer's RML extension: https://github.com/RMLio/rmlstreamer
Chimera's InputStream: https://github.com/cefriel/chimera
[10] https://github.com/kg-construct/mapping-challenges/issues/7

4 Approach

In this Section, we describe how we leveraged WoT W3C recommendations to extend RML's Logical Source (Section 4.1) and introduce RML's Logical Target (Section 4.2) to solve the open issues we discussed in Section 3.2.

4.1 WoT W3C recommendations as data access description

We leverage the WoT W3C recommendations as data source description in RML to describe how processors access Web APIs and streams and perform authentication, if needed (Open Issues 1 & 2). The access description of the Web API or stream is described as `td:PropertyAffordance` (Listing 1.3: lines 3-9, 18-23) which consists of an abstraction layer and protocol bindings. The abstraction layer specifies the location of the resource (Listing 1.3: lines 4, 19), the content type of the data (Listing 1.3: lines 5, 20), and if the property can be read (Listing 1.3: lines 6, 21). A `td:PropertyAffordance` can be combined with other protocol-specific vocabularies through binding templates [13], e.g. the HTTP W3C vocabulary [11] (Listing 1.3: lines 7-9, 23). This way, we describe common information, e.g., resource location, content-type, etc. in a generic way and describe protocol-specific features in the mapping rules.

Listing 1.3. WoT based access description for performing an HTTP GET request with authentication through an API key in a custom HTTP header and subscribing to an MQTT stream with authentication embedded in the message body

```
1  <#WoTWebAPISecurity> a wotsec:APISecurityScheme;
2    wotsec:in "header"; wotsec:name "apikey".
3  <#WoTWebAPISource> a td:PropertyAffordance;
4    td:hasForm [ hctl:hasTarget "http://example.com/data.json";
5      hctl:forContentType "application/json";
6      hctl:hasOperationType td:readproperty;
7      htv:headers ([ htv:fieldName "User-Agent";
8        htv:fieldValue "Mapping language processor"; ]);
9      htv:methodName "GET"; ].
10 <#WoTWebAPI> a td:Thing ;
11   td:hasSecurityConfiguration <#WoTWebAPISecurity>;
12   td:hasPropertyAffordance <#WoTWebAPISource>.
13 <#LogicalSource1> a rml:logicalSource;
14   rml:source <#WoTWebAPISource>;
15   rml:referenceFormulation ql:JSONPath; rml:iterator "$".
16 <#WoTMQTTSecurity> a wotsec:BasicSecurityScheme;
17   wotsec:in "body".
18 <#WoTMQTTSource> a td:PropertyAffordance;
19   td:hasForm [ hctl:hasTarget "mqtt://example.com/mqtt";
20     hctl:forContentType "application/json";
21     hctl:hasOperationType td:readproperty ;
22     mqv:controlPacketValue "SUBSCRIBE"; ].
23 <#WoTMQTT> a td:Thing ;
24   td:hasSecurityConfiguration <#WoTMQTTSecurity>;
25   td:hasPropertyAffordance <#WoTMQTTSource>.
26 <#LogicalSource2> a rml:logicalSource;
27   rml:source <#WoT_MQTT_source>;
28   rml:referenceFormulation ql:JSONPath; rml:iterator "$".
```

We also use the WoT W3C recommendations to describe the authentication of Web APIs and streams. Processors use this information to know how they must authenticate against the Web API or stream to retrieve the data.

The WoT W3C recommendations provide several common authentication descriptions such as `wotsec:APISecurityScheme` (Listing 1.3: lines 1-2) for token based authentication or `wotsec:BasicSecurityScheme` (Listing 1.3: lines 16-17) for authenticating with an username and password. These descriptions not only describe the type of authentication (Listing 1.3: lines 1, 16) but also how the credentials must be provided to the Web API or streams (Listing 1.3: lines 2, 17). This way, we declaratively describe the authentication of Web APIs and streams in the mapping rules. However, the WoT W3C recommendations do not describe the actual credentials such as token, username or password, needed to authenticate with the Web API or stream to avoid leaking the credentials in the WoT descriptions. To overcome this problem, existing vocabularies such as the International Data Spaces Information Model[11] can be used to specify credentials. This way, we declaratively describe the credentials for processors and avoid to leak them by keeping them separated from the mapping rules.

4.2 Introducing RML's Logical Target

We introduce the Logical Target[12] in RML which describes the characteristics of the generated knowledge graph, e.g., serialization format, and target destination of the generated knowledge graph, e.g., storage location (Open Issue 3).

While a Logical Source is part of a Triples Map, a Logical Target is a part of a Term Map specified by `rmlt:logicalTarget` (Listing 1.4: lines 12, 15) which expects a RML Logical Target description. This way, we have fine-grained control over where each triple is exported to (Listing 1.4: lines 12, 15).

We follow the same approach for the output description as RML does for the input description to specify how a target must be accessed and where the knowledge graph must be exported to. We consider the same vocabularies, e.g., VoID [1], SD [23] or WoT, to describe the access to the target destination of the generated knowledge graph as to specify the access to a data source.

A Logical Target describes how a processor accesses a target and the location where the knowledge graph must be exported to with the `rmlt:target`[13] property (Listing 1.4: line 7). This way, we reuse the data source access descriptions used in RML's Logical Source to specify RML's Logical Target. For instance, we use a `void:Dataset` description as data target (Listing 1.4: lines 4-5) in a Logical Target to export the generated knowledge graph to the local disk or a `sd:Service` description to export to a triple store using SPARQL UPDATE queries (Listing 1.4: lines 1-3).

A Logical Target also contains an optional `rmlt:serialization`[14] property (Listing 1.4: line 8) to specify which serialization format must be used to export the generated knowledge graph. The `rmlt:serialization` property reuses the existing W3C `formats` namespace[15] as declarative description of the output

[11] https://w3id.org/idsa/core
[12] https://rml.io/specs/rml-target
[13] http://semweb.mmlab.be/ns/rml-target#
[14] http://semweb.mmlab.be/ns/rml-target#
[15] https://www.w3.org/ns/formats/

RDF format. If no format is specified, the serialization format is N-Quads by default.

We also added an optional `rmlt:compression`[16] property to the domain of RML's Logical Target to describe which compression algorithm is used to save network bandwidth and storage when exporting a knowledge graph (Listing 1.4: line 9). `rmlt:compression` requires an object from the Compression (`comp`) namespace[17]. By specifying the compression algorithm through the `comp` namespace, we declaratively describe the compression algorithms. When the property is not specified, no compression is applied when exporting the knowledge graph.

Listing 1.4. RML Logical Target to export a knowledge graph to local disk as N-Triples with GZip compression & SPARQL endpoint with SPARQL UPDATE.

```
 1  @prefix sd: <http://www.w3.org/ns/sparql-service-description#> .
 2  <#SPARQLUPDATE> a sd:Service;
 3    sd:endpoint  <http://example.com/sparql-update>;
 4    sd:supportedLanguage sd:SPARQL11Update.
 5  <#FileDump> a void:Dataset;
 6    void:dataDump <file:///home/dylan/out.nq>.
 7  <#LogicalTarget1> a rmlt:LogicalTarget;
 8    rmlt:target <#FileDump>;
 9    rmlt:serialization formats:N-Triples;
10    rmlt:compression comp:GZip.
11  <#TriplesMap> a rr:TriplesMap;
12    rr:subjectMap [ rr:template "http://example.com/{name}";
13      rmlt:logicalTarget <#LogicalTarget1> ];
14    rr:predicateObjectMap [ rr:predicate foaf:name;
15      rr:objectMap [ rml:reference "name";
16        rml:logicalTarget [ a rml:LogicalTarget; rml:target <#SPARQLUPDATE> ];
17      ];
18    ].
```

5 Validation

In this Section, we explain how we implemented our approach (Section 5.1) in the RMLMapper and RMLStreamer, and how we applied our approach to two use cases: ESSENCE (Section 5.2) and DAIQUIRI (Section 5.3)

5.1 Implementation

We implemented our approach in two RML processors, the RMLMapper[18] and RMLStreamer[19], to show that our approach can be applied to any implementation following the RML specification. The RMLMapper follows a mapping-driven approach by executing each Triples Map one by one to generate a single knowledge graph. To the contrary, the RMLStreamer uses a data-driven approach by executing the Triples Maps based on the retrieved data records. The knowledge graph is generated continuously as a data stream.

[16] http://semweb.mmlab.be/ns/rml-target#
[17] http://semweb.mmlab.be/ns/rml-compression#
[18] https://github.com/RMLio/rmlmapper-java
[19] https://github.com/RMLio/rmlstreamer

5.2 ESSENCE use case

Initial pipeline We created an initial pipeline (Figure 1) consisting of several scripts and mapping rules to retrieve the measurement data, authenticate against Web APIs, generate knowledge graphs, and export the generated knowledge graphs to multiple targets. The mapping rules describe only how the retrieved data need to be integrated into a knowledge graph. The authentication against the Web API, the data retrieval, and export of the generated knowledge graphs are not declaratively described. Instead, they are handled by the use case specific scripts. Each script was especially written for the ESSENCE use case, and cannot be reused for our other use cases.

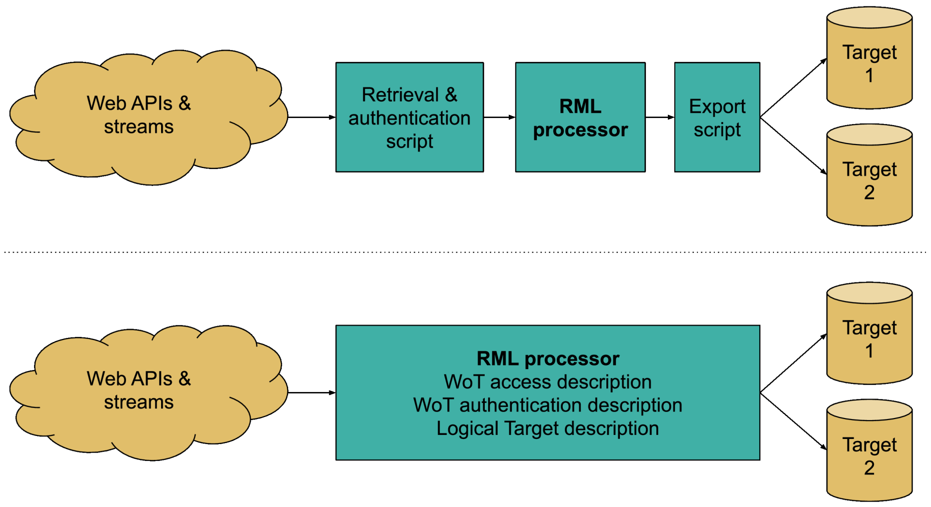

Fig. 1. Above, the initial use case specific pipeline with use case specific scripts to retrieve the data and export the generated knowledge graph. Below, our approach which declaratively describes how the data should be retrieved and how the generated knowledge graph should be exported.

Declaratively described pipeline In our approach[20], we not only declaratively describe how the data must be integrated into a knowledge graph, but also how processors must authenticate against Web APIs, retrieve the data from Web APIs and how processors must export the generated knowledge graphs to multiple targets. This way, the mapping rules not only describe how the data is integrated, but also how the data is accessed and how the knowledge graphs are exported. We replaced our retrieval and authentication script with Web of Things Web API access and authentication descriptions (Section 4.1, Figure 1). This way, the RMLMapper can authenticate to the Web APIs and retrieve the

[20] https://github.com/RMLio/web-of-things-icwe2021

necessary data. Since these descriptions are reusable in other use cases, our approach provides a generic solution for Open Issue 2. Afterward, we replaced the export script as well with Logical Target descriptions (Section 4.2, Figure 1). Consequently, the RMLMapper can export its generated knowledge graph directly to a triple store and local disk for backups. The local backups are also compressed during the export to save disk space (Section 4.2).

5.3 DAIQUIRI use case

Initial pipeline As in ESSENCE, we first created an initial pipeline (Figure 1) consisting of several scripts and mapping rules to retrieve the data from the MQTT stream, integrate the data into knowledge graphs, and export the generated knowledge graphs to multiple targets. The mapping rules only describe how the data is integrated, the actual data retrieval and export of knowledge graphs are handled by use case specific scripts which cannot be reused in other use cases. Since the RMLStreamer only supports Kafka and TCP streams, we created a script to transform the MQTT stream into a Kafka stream and back to an MQTT stream when exporting the knowledge graph.

Declaratively described pipeline By applying our approach to DAIQUIRI[21], we declaratively describe the knowledge graph generation pipeline from retrieving the data until exporting the generated knowledge graphs to multiple targets. We extended the RMLStreamer to support WoT descriptions for accessing MQTT streams which allowed us to remove our initial data retrieval script. The WoT descriptions contain sufficient information for the RMLStreamer to retrieve the data directly (Section 4.1, Figure 1) which solves Open Issue 1. We also reused the same descriptions in a Logical Target for exporting the generated knowledge graphs as an MQTT stream and store compressed backups locally (Section 4.2).

Our approach was validated for both ESSENCE and DAIQUIRI regarding to exporting a knowledge graph by using the same access descriptions for heterogeneous data sources in RML (Open Issue 3).

6 Conclusion

In this paper, we investigated how mapping languages can describe the characteristics of (i) accessing data streams and Web APIs, and (ii) exporting a knowledge graph. We validated our approach with two real-life use cases which showcases that our approach can be used for accessing Web APIs and streams and exporting knowledge graphs. This shows that our approach improves the reproducibility of knowledge graph generation as we not only declaratively describe how the knowledge graph should be generated, but also how the Web APIs and streams should be accessed, and the generated knowledge graph exported. WoT W3C recommendations enable mapping languages to access Web APIs and

[21] https://github.com/RMLio/web-of-things-icwe2021

streams without depending on a specific protocol. More, access descriptions can be leveraged for describing how generated knowledge graphs must be exported.

Only a limited amount of protocol bindings are standardized so far. The WoT Working Group released several recommendations, which are used in this paper, but some parts of the recommendations are still in development such as the protocol bindings. These protocol bindings provide descriptions for protocol-specific options and need to be provided for each protocol separately. However, if no protocol-specific options are needed, the abstraction layer of the WoT W3C recommendations covers the necessary parts to access a Web API or stream.

Further research should be undertaken to investigate how pagination in Web APIs can be handled as our work only covers access of Web APIs in mapping languages. Furthermore, more investigation must be applied to determine that our work covers all possible Web API use cases.

References

1. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing Linked Datasets with the VoID Vocabulary. Interest group note, World Wide Web Consortium (W3C) (2011), https://www.w3.org/TR/void/
2. Charpenay, V., Lefrançois, M., Poveda Villalón, M., Käbisch, S.: Thing Description (TD) Ontology. Working group editor's draft, World Wide Web Consortium (W3C) (2020), https://www.w3.org/2019/wot/td
3. Chortaras, A., Stamou, G.: Mapping Diverse Data to RDF in Practice. In: The Semantic Web – ISWC 2018. Lecture Notes in Computer Science, vol. 11136. Springer, Cham (2018)
4. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.0 (2000)
5. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. Working group recommendation, World Wide Web Consortium (W3C) (2012), http://www.w3.org/TR/r2rml/
6. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In: Proceedings of the 7th Workshop on Linked Data on the Web. CEUR Workshop Proceedings, vol. 1184. CEUR-WS.org (2014)
7. Dimou, A., Verborgh, R., Sande, M.V., Mannens, E., de Walle, R.V.: Machine-interpretable dataset and service descriptions for heterogeneous data access and retrieval. In: Proceedings of the 11th International Conference on Semantic Systems - SEMANTICS '15. ACM Press (2015)
8. Haesendonck, G., Maroy, W., Heyvaert, P., Verborgh, R., Dimou, A.: Parallel RDF generation from heterogeneous big data. In: Proceedings of the International Workshop on Semantic Big Data - SBD '19. ACM Press, Amsterdam, Netherlands (2019)
9. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. Recommendation, World Wide Web Consortium (W3C) (2013), https://www.w3.org/TR/sparql11-query/
10. Kaebisch, S., Kamiya, T., McCool, M., Charpenay, V., Kovatsch, M.: Web of Things (WoT) Thing Description. Working group recommendation, World Wide Web Consortium (W3C) (2020), http://www.w3.org/TR/wot-thing-description/

11. Koch, J., Valesco, C.A., Ackermann, P.: HTTP Vocabulary in RDF 1.0. Working group note, World Wide Web Consortium (W3C) (2017), http://www.w3.org/TR/HTTP-in-RDF10/
12. Konstantinou, N., Spanos, D.E., Houssos, N., Mitrou, N.: Exposing scholarly information as Linked Open Data: RDFizing DSpace contents. The Electronic Library (2014)
13. Koster, M., Korkan, E.: Web of Things (WoT) Binding Templates. Working group note, World Wide Web Consortium (W3C) (2020), http://www.w3.org/TR/wot-binding-templates/
14. Lanthaler, M.: Hydra Core Vocabulary: A Vocabulary for Hypermedia-Driven Web APIs. Unofficial draft, World Wide Web Consortium (W3C) (2019), http://www.hydra-cg.com/spec/latest/core/
15. Lefrançois, M., Zimmermann, A., Bakerally, N.: A SPARQL extension for generating RDF from heterogeneous formats. In: The Semantic Web 14th International Conference, ESWC 2017, Proceedings. Springer International Publishing, Portoroz, Slovenia (2017)
16. Maali, F., Erickson, J.: Data Catalog Vocabulary (DCAT). Recommendation, World Wide Web Consortium (W3C) (2014), https://www.w3.org/TR/vocab-dcat/
17. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic Markup for Web Services. Member submission, World Wide Web Consortium (W3C) (2004), http://www.w3.org/Submission/OWL-S/
18. Mauri, A., Calbimonte, J.P., Dell'Aglio, D., Balduini, M., Brambilla, M., Della Valle, E., Aberer, K.: TripleWave: Spreading RDF streams on the web. In: The Semantic Web – ISWC 2016. Springer International Publishing, Cham (2016)
19. Michel, F., Djimenou, L., Faron-Zucker, C., Montagnat, J.: Translation of Heterogeneous Databases into RDF, and Application to the Construction of a SKOS Taxonomical Reference. In: International Conference on Web Information Systems and Technologies. Springer (2015)
20. Santipantakis, G.M., Kotis, K.I., Vouros, G.A., Doulkeridis, C.: RDF-Gen: Generating RDF from Streaming and Archival Data. In: Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics (2018)
21. Scrocca, M., Comerio, M., Carenini, A., Celino, I.: Turning Transport Data to Comply with EU Standards While Enabling a Multimodal Transport Knowledge Graph. In: International Semantic Web Conference. Springer (2020)
22. Tommasini, R., Sedira, Y.A., Dell'Aglio, D., Balduini, M., Ali, M.I., Le Phuoc, D., Della Valle, E., Calbimonte, J.P.: Vocals: Vocabulary and catalog of linked streams. In: The Semantic Web – ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11137. Springer, Cham (2018)
23. Williams, G.: SPARQL 1.1 Service Description. Recommendation, World Wide Web Consortium (W3C) (2013), https://www.w3.org/TR/sparql11-service-description/