Extending RML to Support Permissioned Data Sharing with Multiple Views

Els de Vleeschauwer[1][0000−0002−8630−3947],
Gerald Haesendonck[1][0000−0003−1605−3855],
Ben De Meester[1][0000−0003−0248−0987], and Pieter Colpaert[1][0000−0001−6917−2167]

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

Abstract. The usage of Semantic Web technologies for data integration has extended from open data sharing to permissioned data sharing, as exemplified by standardization efforts from, for example, the Solid project and the Fedora Repository. These existing efforts have shown that the same data must be made available in multiple disjoint and overlapping views, depending on which access control policies are applied. To achieve such interoperable and permissioned data sharing, existing data owners need a transformation process for their existing data produced, stored, and managed in their legacy systems. This paper presents a method and tooling for sharing a legacy source dataset in multiple Linked Data views under different access control policies. Our approach employs the RDF Mapping Language (RML) to map data from existing structures to RDF, extends it with dynamic target descriptions to support any Linked Data publication strategy, and stores the mapped data on Solid pods for permissioned sharing. After functionally evaluating that our solution complements the state of the art, we validated our solution in Onto-DESIDE, an EU project on circular economy, demonstrating how it enables the creation of permissioned views over legacy system data, lowering the barriers to participate in a circular value network. As our RML extensions support standardized HTTP access to any web resource, and the dynamic generation of any publication target, they are generic enough to support any (permissioned) Linked Data publication strategy, outside of Solid and Onto-DESIDE. Future work includes research on write synchronization with the source data and on-the-fly-generation of permissioned Linked Data views. We will pursue further alignment with Solid, and take-up within RML, to increase the method's longevity.

Keywords: Solid · RML · data sharing · access control

Resource Types: Two specifications, one vocabulary, one implementation and one demonstrator

URL: `https://w3id.org/imec/rml/publications/eswc2025/rml-solid/`

Licenses: CC BY 4.0 and MIT

DOI: 10.5281/zenodo.14507298

1 Introduction

The usage of Semantic Web technologies for data integration has extended from open data sharing to permissioned data sharing, as exemplified by standardization efforts from, for example, the Solid project[1] and the Fedora Repository[2]. These existing efforts have shown that the same data must be made available in multiple disjoint and overlapping views [5], depending on which access control policies are applied.

To achieve such interoperable and permissioned data sharing, existing data owners need a transformation process for their existing data produced, stored, and managed in their legacy systems. Today, lack of support for permissioned data sharing in a secure and automated way is one of the real-world obstacles that industry actors point to, as exemplified by our motivating use case: the Onto-DESIDE project[3], an EU project on circular economy. Specifically, *data minimization* (i.e. only sharing specific subsets of the same data) tailored to the requesting party is a common use case. This is not only enforced in personal data sharing legislation (e.g. GDPR Art. 5.1(c)), but also needed and enforced in industrial data sharing (e.g. REACH Art. 7), e.g. the precise chemical composition of a product might contain intellectual property critical to a company's competitive edge, but the thresholds of certain (toxic) compounds need to be shared by law.

This paper presents a method and tooling for sharing legacy source datasets in multiple disjoint and overlapping Linked Data views under different access control policies. The views are published as interoperable Resource Description Framework (RDF) resources following the Solid protocol, a web decentralization standard designed to give users control over their data. For the transformation of the existing data to RDF data we use the RDF Mapping Language (RML). To enable the description of the end-to-end pipeline, from existing data to resource on the Solid pod, we propose an access description for HTTP requests in RML. To define data publication strategies of which the URI strategy can be defined both statically as based on the source data, we extend the RML logical target with a dynamic logical target. This adds design flexibility needed for fine-grained access control over the published views.

We implemented our RML extensions in RMLMapper, functionally compared our solution to the state of the art, and validated it in Onto-DESIDE, demonstrating how it enables the creation of permissioned views over legacy system data, lowering the barriers to participate in a circular value network.

Our solution can be used in any use case for permissioned sharing of existing data, beyond the Onto-DESIDE project. The developed RML extensions are generic – applicable to protocols other than Solid – to describe any pipeline from or to web resources reachable with HTTP requests, and to publish towards any target type, generated dynamically taking the source data into account.

[1] `https://solidproject.org/`
[2] `https://fedorarepository.org/`
[3] `https://ontodeside.eu/`

After discussing related work (Section 2), we analyze the requirements of the state of the art and of our use case (Section 3), and describe our approach (Section 4). Finally, we validate our solution (Section 5) and conclude (Section 6).

2 Related Work

Related work has shown that to transform existing data to RDF, using declarative methods has become best practice and mature [11], as declarative methods are independent of the use case compared to ad-hoc tools and scripts. Hence, we focus our work and related work investigation on declarative methods. In this section, we (i) present the Onto-DESIDE project, our motivating use case; (ii) discuss Semantic Web technologies for permissioned data sharing; (iii) present declarative mapping solutions to convert existing data to an interoperable format; and (iv) discuss proposals to share open existing data via standards using a declarative mapping tool.

In the **Onto-DESIDE project**[4] (Ontology-based Decentralized Sharing of Industry Data in the European Circular Economy) semantic ontologies and Linked Data are combined with the concept of digital twins to enable data collaboration in the context of circular economy [3]. Circular economy aims at reducing value loss and avoiding waste by circulating material or product parts before they become waste. Today, the main obstacle to create new circular value networks is the lack of support for secure, quality-assured, and automated data sharing. Additionally, inconsistent terminology and undefined concepts make it challenging to establish such circular value networks. The Open Circularity Platform is the technical solution for the Onto-DESIDE project [4]. This platform includes a *transform*, *share*, *query*, and *view* step, to automate the sharing of existing data in a semantically and technically interoperable manner while data owners retain the control over their data. In this paper, we present new contributions to the *transform* and *share* step of the Open Circularity Platform.

As a basis for the *transform* step, **mapping languages describe the conversion of heterogeneous (semi-)structured data to semantic interoperable RDF data**, independent of the application executing the mapping. The RDF Mapping Language (RML) – originally extended from the W3C recommended RDB to RDF Mapping Language R2RML[5] – is supported by most implementations [11] and gathered a large community of contributors and users, resulting in a new modular specification for RML developed by the W3C Community Group on Knowledge Graph Construction [8]. In RML, RDF triple generation is defined using *triples maps*. Within these triples maps, *expressions* are used to fill in data values in specified places in the triples. On which data these triples are generated, is specified using a *logical source*: a construct to describe how to extract *logical iterations* out of heterogeneous data sources. What should happen with the generated triples, is specified using a *logical target*: a construct to describe how to export the triples after their generation.

[4] `https://ontodeside.eu`
[5] `https://www.w3.org/TR/r2rml/`

As a basis for the *share* step, several **Linked Data sharing technologies foresee (customized) solutions for access control**, e.g. customized SPARQL endpoints such as GraphDB[6] and Linked Data platforms such as Metaphactory[7] [6]. The Solid project and the Fedora Repository are web decentralization projects based on open web standards, e.g. the W3C recommended Linked Data Platform (LDP)[8], decoupling data storage from applications. Individual users can store their data as a web resource in personal online data stores (pods), which they fully control. Users decide who has access to their resources, granting permissions to specific resources or groups of resources (containers) on their pod. The recently started W3C Working Group on Linked Web Storage[9] aims to develop a W3C recommendation based on the work of the Solid Community Group.

Several works exist to **create interoperable sharing of existing data**, i.e. presenting an all encompassing solution for the *transform* and *share* step. Morph-LDP [9] is an R2RML-based LDP implementation that exposes relational data as read/write Linked Data for LDP-aware applications, whilst allowing legacy applications to continue using their relational databases. The design of the resulting LDP is predefined: one container resource per triples map, one RDF resource per subject. LDP-DL [1] is a language to define the design of LDPs, integrating SPARQL-Generate (a SPARQL-based declarative mapping language) to transform heterogeneous data sources to RDF data. LDP-DL allows to specify how the LDP structure is built, offering more flexibility than Morph-LDP in that regard, and comes with a workflow implementation [2] to automate the generation of read-only LDPs from heterogeneous data sources, supporting also on-the-fly generation of RDF content. Both works use a single configuration document to declaratively describe the full LDP deployment.

3 Requirements Analysis

In this section, we first describe the requirements from the Onto-DESIDE project, our motivating use case. Next, we present complementary requirements derived from the state of the art, i.e. Morph-LDP and LDP-DL. Although these complementary requirements are not critical to the Onto-DESIDE project, they offer opportunities for future enhancements. We explain all requirements from the perspective of actors seeking to share data within the Open Circularity Platform to create a circular value network, discuss to which extent Morph-LDP and the LDP-DL workflow satisfy these requirements, and identify the open issues.

[6] `https://graphdb.ontotext.com/documentation/10.7/access-control.html`
[7] `https://help.metaphacts.com/resource/Help:Security`
[8] `https://www.w3.org/TR/ldp/`
[9] `https://www.w3.org/2024/09/linked-web-storage-wg-charter.html`

3.1 Requirements from the Onto-DESIDE project

Within the Onto-DESIDE project, our solution aims to automate the sharing of existing data in a semantically and technically interoperable manner while data owners retain the control over their data. This results in following requirements.

Semantic Interoperability [R1]: Align to a common data model to combine the data from different actors within a circular value network. Both LDP-DL and Morph-LDP leverage a mapping language, resp. SPARQL-Generate and R2RML, to transform the source data to any ontology using RDF.

Technical Interoperability: Read Access [R2]: Do not transfer your data to a central data platform, but share it in a decentralized way, to retain control. Decentralized data must be reachable over the web with standardized operations. Both LDP-DL and Morph-LDP expose the data using LDP and thus support decentralized read operations via HTTP requests.

Access Control [R3]: Do not share confidential or sensitive data with every other actor, but restrict the access to certain parts of the data to specific users. Access control was not required nor supported by LDP-DL and Morph-LDP, and it is not inherent to LDP. Solving this open issue is the main contribution of this paper.

Flexible design [R4] Freely choose the fragmentation of the shared data, i.e. which data is exposed per read operation: this impacts the performance of applications using the data, and the possibilities to control the access to this data. In Morph-LDP the design of the resulting LDP is predefined: all triples with the same subject are exposed with the same read operation. In contrast, LDP-DL allows for a fully customizable LDP structure design.

Heterogeneous Data Sources [R5]: Support any existing data format: as each actor has independently chosen the data format fitting its own needs (e.g. CSV, JSON, or SQL), this is heterogeneous by nature. LDP-DL leverages the capability of SPARQL-Generate to handle any source data format. Morph-LDP only supports relational databases as source format.

Automated generation [R6]: Automate the generation of a data sharing solution. Existing LDP or Solid implementations require much manual development, time, and expertise that the actors may not want to invest when putting in place their data sharing solution. Both LDP-DL and Morph-LDP provide the tools for automatic deployment from source data to shared RDF data. The entire LDP-compliant structure is automatically generated based on a single design document.

3.2 Complementary Requirements from the State of the Art

Following LDP-DL requirements [1] are not yet covered by Onto-DESIDE's critical features mentioned above.

Reusable design [R7]: Reuse existing designs to expose data in a similar way to enhance integration and access of cross-actor applications. Such applications may exploit any actor's LDP as long as the LDPs use a design and vocabulary known by the application. Both LDP-DL and Morph-LDP configure their data sharing in a standalone model, independent and separate from any implementation. These models can be shared and adapted for deployment over another dataset.

On-the-Fly Generation [R8]: Generate interoperable data on-the-fly, as materializing all data as RDF might be prevented, e.g. by data sharing license restrictions or storage limitations. The LDP-DL workflow includes InterLDP[10], an LDP server that can optionally generate the content of requested resources at query time. This option decreases storage requirements but increases the response time. Morph-LDP is built on top of Morph-RDB[11] [10], providing a virtualized knowledge graph (i.e. on-the-fly generation) by default.

Morph-LDP covers one more requirement [9].

Technical Interoperability: Write Access [R9]: Provide write access, so that an actor can contribute to another actor's data. Morph-LDP exposes the relational data as Linked Data, readable through HTTP GET operation ([R2]), and also writable using PUT, POST, PATCH, and DELETE.

4 Approach

For our solution, we integrate two emerging standards: the RDF Mapping Language (RML) and Solid. Both RML and Solid have an active W3C Community Group of contributors, users, and developers of compliant tools. Aligning with or even performing an integration with these standards increases the longevity of our solution and supports the development of those standards. RML can *transform* the existing data to RDF data. RDF data can be *shared* with access control on a Solid pod.

With the addition of the *logical target* to RML [12], we can describe and execute a complete knowledge graph publication pipeline with RML rules. Each logical target has exactly one *target access description*, describing how a target must be accessed when exporting RDF triples. To the best of our knowledge, there is no target specification nor implementation to access Solid pods in RML. Additionally, the current *static* logical targets in RML do not offer the flexibility to define data-driven views, needed for fine-grained access control. To resolve these two open issues, we propose: (i) an HTTP request access vocabulary (Section 4.1), and (ii) the introduction of dynamic logical targets (Section 4.2).

[10] `https://github.com/noorbakerally/InterLDP`
[11] `https://github.com/oeg-upm/morph-rdb`

4.1 HTTP Request Access

To describe an end-to-end pipeline from heterogeneous data sources to RDF resources with access control on a Solid Pod, we provide an RML access description to Solid resources (fig. 1)[12].

```
 1 <products_logical_target> a rml:LogicalTarget ;
 2   rml:serialization formats:Turtle ;
 3   rml:target [
 4     a rml:Target, rmle:DirectHttpRequest ;
 5     htv:absoluteURI "https://manufacturer-pod/products" ;
 6     htv:methodName "PUT" ;
 7     rmle:contentTypeHeader "text/turtle" ;
 8     rmle:userAuthentication <auth>
 9   ] .
10 <products_acl_logical_target> a rml:LogicalTarget ;
11   rml:serialization formats:Turtle ;
12   rml:target [
13     a rml:Target, rmle:LinkedHttpRequest ;
14     rmle:linkingAbsoluteURI "https://manufacturer-pod/products" ;
15     rmle:linkRelation "acl" ;
16     htv:methodName "PUT" ;
17     rmle:contentTypeHeader "text/turtle" ;
18     rmle:userAuthentication <auth>
19   ].
20 <auth> a rmle:CssClientCredentialsAuthentication ;
21   rmle:authEmail "hello@manufacturer.com" ;
22   rmle:authPassword "abc123" ;
23   rmle:authWebId <https://manufacturer/profile/card#me> ;
24   rmle:authOidcIssuer <https://manufacturer-pod/> .
```

Fig. 1. Logical target descriptions for Solid resources. The HTTP request adds a new resource to a Solid pod (lines 4–8). The linked HTTP request adds a new ACL file for a resource in a Solid pod (lines 13–18). Both target descriptions include authentication details (lines 8, 18, and 20–24) to properly execute authenticated HTTP requests.

The Solid protocol specifies how Solid resources are accessed, e.g. with an HTTP PUT request you can add data to a resource with a specific URL. To enable permissioned data sharing, access rules for a resource are provided in an auxiliary resource, e.g. an Access Control List (ACL) resource[13]. Clients discover the ACL resource associated with a resource by making an HTTP HEAD request on the URI of the resource to which the ACL resource applies. The URI of the

[12] Prefixes are omitted but can be found on `https://prefix.cc`. Specifically, the prefix `rmle` is used for the proposed RML extensions.

[13] `https://solidproject.org/TR/protocol#web-access-control`

ACL resource is exposed with the relation `acl` in the HTTP Link header. Unless public write and control access is specified for a resource, you need to provide authentication details to manipulate that resource and its access rules.

We provide an *HTTP Request Access Specification*[14], modeling the access to a web resource as a simplified HTTP request with optional authentication, extensible to any authentication type. Our specification reuses the *HTTP Vocabulary in RDF 1.0* [15], and adds a custom vocabulary (i) for access to linked resources (i.e. discoverably via the Link header) and (ii) for providing sufficient authentication details to execute authenticated HTTP requests. The HTTP request access specification covers following main classes: the *HTTP request* (`rmle:HttpRequest`) and the *user authentication* (`rmle:UserAuthentication`).

HTTP Request For each HTTP request the method name, accept header, content-type header, and user authentication details can be configured. The specification includes defaults for the method name, accept header and content-type of the HTTP request. When using the HTTP request for an RML target, the body of the HTTP request contains the generated RDF triples, the default method name is `PUT` and the default content-type header is derived from the serialization format of the logical target (e.g. the serialization format `formats:Turtle` will default the content-type header to `text/turtle`).

The *direct HTTP request* (`rmle:DirectHttpRequest`) is a subclass of the HTTP request, and describes access to a web resource with the absolute URI of the web resource.

The *linked HTTP request* (`rmle:LinkedHttpRequest`) is a subclass of the HTTP request, and describes how to access a web resource linked from another web resource, via the HTTP Link header with a specified relation. One use case for a linked HTTP request is access to an ACL resource, linked from a resource using the `acl` relation in the HTTP Link header.

User Authentication A resource of the type *user authentication* holds the authentication details. The required authentication information depends on the authentication type. We specified the *CSS client credentials authentication*, describing the information required for authentication using client credentials as specified in the Community Solid Server version v7[16], as this authentication type covers the needs of our motivating use case. However, the specification is extensible to any other authentication type.

4.2 Dynamic Logical Targets

In Solid, access control is managed on resource level, i.e. per URI. Multiple overlapping and disjoint views over the same data (i.e. exposing the same data

[14] `https://rml.io/specs/access/httprequest/20241212/`

[15] `https://www.w3.org/TR/HTTP-in-RDF10/`

[16] `https://communitysolidserver.github.io/CommunitySolidServer/7.x/usage/client-credentials/`

in more than one resource) are needed to satisfy the needs of real world use cases requiring fine-grained access control [5]. For example, if product data is exposed as one resource, e.g. with the URI *https://manufacturer-pod/products*, users will either get access to all product data, or no product data. If you manage the access to your products on product level, you need to expose resources per product, e.g. with URIs *https://manufacturer-pod/product1* and *https://manufacturer-pod/product2*.

In RML, a term map can specify one or multiple logical targets. Triples containing a term generated by a term map, are exported to all logical targets specified in that term map. Although this allows a many-to-many relation between the generated triples and the (static) targets they end up in (and thus supports overlapping and disjoint views), there is **no solution in RML to define a logical target *based on source data***, e.g. define a logical target per product. Existing workarounds[17] allow defining multiple logical target based on a static set of cases, however, whenever a product is added to the source data, the mapping file must be updated.

To create such multiple data-driven views we introduce *dynamic logical targets* as a new concept in RML. We propose a three-step solution to define logical targets based on source data, published as the RML Dynamic Targets Specification[18]. An example in RML mapping rules is presented in Figure 2.

Step 1: Generate Logical Targets using RML. RML mappings are RDF graphs: logical targets are a set of RDF triples. Consequently, it is possible to define RML mapping rules to generate logical targets based on source data (Figure 2, lines 1–12, 13–30). Any property of a logical target, defined with any vocabulary, can be generated based on source data. This is inherently possibly in RML, without adding any RML language constructs.

Step 2: Introduce a Self-Referencing Logical Target. To use these generated logical targets during the same mapping process, we introduce a predefined instance of a logical target, i.e. `rmle:ThisMapping` (Figure 2 lines 6 and 18). This logical target indicates that the triples generated by the connected term map have to be added to the RML mapping rules that are part of the current mapping process, as additional RML rules. This is an alternative for executing two consecutive mapping processes, keeping all configuration inside one document and allowing for optimizations of the mapping process, e.g. grouping mappings with the same data source.

Step 3: Introduce a Logical Target Map. In RML, logical targets are identified via a blank node or named node. We introduce a logical target map (`rmle:LogicalTargetMap`), a subclass of an *expression map*, to dynamically generate these identifiers using values of the logical iteration (Figure 2 lines 36–38). The logical target map is aligned with other dynamic mapping constructs

[17] `https://kg-construct.github.io/rml-io/spec/docs/#conditions`
[18] `https://rml.io/specs/target/dynamictarget/20241212/`

```
 1 <tm_lt_product> a rml:TriplesMap;
 2   rml:logicalSource <products_source>;
 3   rml:subjectMap [
 4     rml:template "lt/{product_id}";
 5     rml:class rml:LogicalTarget;
 6     rml:logicalTarget rmle:ThisMapping;
 7   ];
 8   rml:predicateObjectMap [
 9     rml:predicate rml:target;
10     rml:objectMap [rr:template "t/{product_id}"
11     ];
12   ].
13 <tm_t_product> a rml:TriplesMap;
14   rml:logicalSource <products_source>;
15   rml:subjectMap [
16     rml:template "t/{product_id}";
17     rml:class rml:Target, rmle:DirectHttpRequest;
18     rml:logicalTarget rmle:ThisMapping;
19   ];
20   rml:predicateObjectMap [
21     rml:predicate htv:absoluteURI;
22     rml:objectMap [
23       rml:template "http://manufacturer/products/{product_id}";
24       rml:termType rml:Literal;
25     ];
26   ];
27   rml:predicateObjectMap [
28     rml:predicate rmle:userAuthentication;
29     rml:objectMap <auth>;
30   ].
31 <tm_products> a rml:TriplesMap;
32   rml:logicalSource <products_source>;
33   rml:subjectMap [
34     rml:template "http://manufacturer/products/{product_id}";
35     rml:class ex:product;
36     rmle:logicalTargetMap [
37       rml:template "lt/{product_id}"
38     ];
39   ].
```

Fig. 2. Two triples maps generate logical targets and targets based on source data (lines 1–30). The logical target of these triples maps is `rmle:ThisMapping` (lines 6 and 18), indicating that the generated triples should be added to the mapping document in which the triples maps are located. The logical target map (line 36) generates URIs matching the identifiers of generated logical targets, securing the creation of web resources per product.

in RML, e.g. *term maps*. The existing predicate `rml:logicalTarget` can be considered as a *constant shortcut property* for a *constant valued* logical target map.

Result. With a CSV file as data source, each row containing a product and its properties, the RML rules from Figure 2 generate one logical target description per row, i.e. per product. These target descriptions are added to the initial RML mapping rules. The identifiers of the generated targets match the identifiers generated by the logical target map (lines 36–38), exporting the generated RDF data about the products to one resource per product. More products, i.e. more rows in the source data, result in more (dynamically generated) logical targets and more resources in the Solid pod, using the same RML mapping rules. This result is achieved with a minimized addition of new RML constructs (one instantiated logical target and one new predicate), aligned with the existing RML vocabulary, and supporting the dynamic generation of any logical target type.

5 Validation

To validate our approach, we provide an implementation to execute the proposed RML constructs (Section 5.1), we built a demonstrator to showcase how the requirements are satisfied (Section 5.2), and we applied our solution in the Onto-DESIDE project (Section 5.3).

5.1 Implementation in RMLMapper

We implemented HTTP request access and dynamic targets in RMLMapper[19], a Java library capable of executing RML rules to generate RDF data, as part of its version 7.2.0 release. RMLMapper's MIT-licensed codebase, continuously maintained by imec and Ghent University, has seen over 40 stable releases since in 2018 and has been downloaded more than 180,000 times[20].

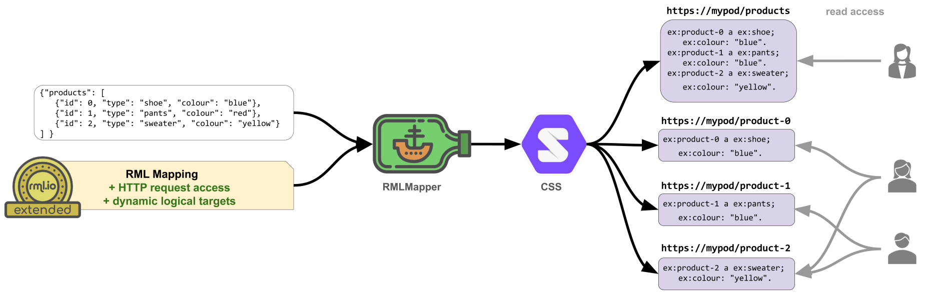
We focused the implementation of the HTTP request access to targets, satisfying the needs of our motivating use case Onto-DESIDE. HTTP request access for sources is planned for a future release. Our implementation includes the optional authentication with CSS Client Credentials, executed as HTTP requests to the CSS server to obtain the needed access tokens. The access tokens are included in the final HTTP PUT request exporting the generated triples to the targeted web resource.

For the implementation of the dynamic targets, we reused the code for the (dynamic) implementation of term maps to generate the identifiers of the logical target map, during the same iteration as the generation of the connected term maps. Thanks to the in-memory architecture of RMLMapper the instantiation of targets can be postponed till the end of the mapping process. The triples for

[19] https://github.com/RMLio/rmlmapper-java

[20] https://tooomm.github.io/github-release-stats/?username=rmlio&repository=rmlmapper-java

the self-referencing logical target `rmle:thisMapping`, i.e. the dynamic target descriptions, are generated simultaneously with all other triples. At the end of the mapping process, RMLMapper adds the triples for `rmle:thisMapping` to the RML mapping rules, instantiates the target connections using the (extended) mapping rules, and directs the remaining triples to the appropriate targets. The implementation in other (not in-memory) RML processors might demand a prioritization of the triple generation for `rmle:thisMapping`, to enable intermediate exports to (dynamic) targets.



```
{"products": [
   {"id": 0, "type": "shoe", "colour": "blue"},
   {"id": 1, "type": "pants", "colour": "red"},
   {"id": 2, "type": "sweater", "colour": "yellow"}
] }
```

https://mypod/products

```
ex:product-0 a ex:shoe;
   ex:colour: "blue".
ex:product-1 a ex:pants;
   ex:colour: "blue".
ex:product-2 a ex:sweater;
   ex:colour: "yellow".
```

https://mypod/product-0

```
ex:product-0 a ex:shoe;
   ex:colour: "blue".
```

https://mypod/product-1

```
ex:product-1 a ex:pants;
   ex:colour: "blue".
```

https://mypod/product-2

```
ex:product-2 a ex:sweater;
   ex:colour: "yellow".
```

Fig. 3. End-to-end pipeline allowing permissioned data sharing from existing source data: RMLMapper processes heterogeneous data sources using an extended RML mapping, and exports the generated RDF data to overlapping and disjoint resources hosted on a CSS Solid pod.

5.2 Demonstrator

We built a demonstrator[21] to showcase how our solution answers the requirements listed in Section 3. We mapped the different aspects of our demonstrator to the requirements (Table 1). Our demonstrator simulates the sharing of data by three manufacturers with ten users with different access rights to the manufacturers' data. Per manufacturer we created (i) heterogeneous source data about products and their properties, (ii) a CSV file to manage the access control, (iii) an extended RML mapping (including the RML extensions described in this paper), and (iv) a Solid pod hosted on a Community Solid Server. To simulate external users, we created ten additional Solid pods. With the pipeline shown in Figure 3 we execute the extended RML mappings to convert the source data and access control data to RDF data and to publish the RDF data and access control rules on the Solid pods of the manufacturers.

RML handles *heterogeneous source data* [R5], with logical sources to describe the access to the sources, and expressions to extract the values from the sources. The demonstrator's source data of the first and third manufacturer is spread

[21] https://github.com/RMLio/rml-solid-demonstrator

Table 1. The demonstrator pipeline satisfies all requirements for the Onto-DESIDE use case. Write access and on-the-fly-generation, complementary requirements derived from the state of the art, will be considered for future enhancements.

	Morph-LDP	LDP-DL	RML+SOLID
R1 Semantic interoperability	✓	✓	✓
R2 Technical interoperability: read access	✓	✓	✓
R3 Access control			✓
R4 Flexible design		✓	✓
R5 Heterogeneous data sources		✓	✓
R6 Automated generation	✓	✓	✓
R7 Reusable design	✓	✓	✓
R8 On-the-fly-generation	✓	✓	
R9 Technical interoperability: write access	✓		

over two files, with heterogeneous file formats (CSV and JSON) and heterogeneous labels (e.g. `ProductID` and `product_id`). The second manufacturer has one source file in XML format, with different labels (e.g. `articlenumber`).

RML secures the *semantic interoperability* [R1], mapping the source data to any RDF ontology to secure a common understanding of the data. The demonstrator's three manufacturers map their source data to the same ontology. The first and third manufacturer additionally map their source data to another ontology.

With the addition of HTTP request access (Section 4.1), RML allows to describe the export of the RDF data to resources on a Solid pod via logical targets, thus securing *automated generation* [R6]. Once the RML mapping is created, the demonstrator pipeline generates the RDF data and the corresponding resources on the manufacturer's Solid pod with one command. Updates in the source data are handled with a new pipeline run.

With the addition of *dynamic targets*(Section 4.2), our solution extends the *design flexibility* [R4] of RML: the data can be exposed in multiple views of varying granularity, using also source data to create the URIs of the resources. The demonstrator presents five examples of granularity, ranging from a single resource containing all properties of all products to individual resources for each property of each product. These resources expose the manufacturers' source data through overlapping and disjoint views: the resource containing all product properties overlaps with the resources for individual product properties, while the resources for individual product properties are mutually disjoint.

Solid allows *access control* [R3] per resource. In our demonstrator, the manufacturers keep an overview of the access rights to the resources on their Solid pods in a locally stored CSV file. RML maps these data of CSV files to ACL rules. Linked HTTP requests (Section 4.1) allows to describe RML rules for publishing these ACL rules as an ACL resource on the manufacturer's Solid pod, linked to the resource to which they apply.

Solid secures the *technical read interoperability* [R2]. Users with *read access* rights can access the generated RDF data over HTTP. Our demonstrator includes examples of HTTP GET requests, with and without authentication, to retrieve the content of a Solid resource.

The RML mappings are *reusable* [R7]. In our demonstrator the third manufacturer has organized his source data in a similar way as the first manufacturer. He reuses the RML mapping of the first manufacturer, updating it only with the base URI of his Solid pod and with his authentication info.

Our pipeline focuses on interoperable permissioned sharing of existing data: it materializes permissioned views over existing data and supports read access to these views. Further research will be needed to secure *on-the-fly generation* [R8] of the views, and to *synchronize write operations in the exposed views with the source data* [R9]. The implementation of on-the-fly-generation might impact the response time, as observed in the LDP-DL experiments. Morph-LDP has implemented write access to the source data, but limits the supported source format (only relational databases) and hardcoded the design of the exposed views resources (one container resource per triples map, one RDF resource per subject).

5.3 Application in the Onto-DESIDE Project

One of Onto-DESIDE's goals is the Open Circularity Platform: a secure and privacy-preserving decentralized data sharing platform [4]. Our solution was developed to support the *transform* and *share* steps of the Open Circularity Platform, mapping a company's source data to an interoperable representation (i.e. RDF data), and sharing this interoperable representation with others through interoperable interfaces with access control (i.e. Solid pods), respectively. With the addition of *dynamic targets* to RML, the RDF data generated by one RML mapping can result in multiple overlapping or disjoint fine-grained resources, enabling the fine-grained access control needed to handle the real-world Onto-DESIDE use cases. With the *HTTP request access vocabulary*, including Solid specific authentication, the transform and share steps are now executed without any ad-hoc scripts, increasing the reusability of the setup.

We validated our solution by implementing demonstrators for the Onto-DESIDE project, validated and evaluated by the industry partners of the project, making sure that the functionality is sufficient for their real-world use cases. The results of these evaluations are publicly reported through Onto-DESIDE's deliverables[22].

6 Conclusion

To facilitate permissioned and interoperable sharing of existing data in multiple views, we presented and implemented two RML extensions: (i) HTTP request access to directly publish RDF as online (Solid) resources; and (ii) dynamic

[22] https://ontodeside.eu/wp-content/uploads/2024/10/Onto-DESIDE_WP6_Deliverable_D6.8_final.pdf

logical targets to create data-driven overlapping and disjoint views. These RML extensions – implemented in RMLMapper – are integrated in a demonstrator, functionally compared to the state-of-the-art and validated with the real-world Onto-DESIDE use cases. Our solution exceeds the capabilities of the state-of-the-art by adding access control. As we anchored our solution in RML and Solid, two emerging open standards with respectively an active W3C community group[23] and working group[24], rather than developing a standalone custom solution, we increase its longevity.

Our contributions are developed within the scope of the Onto-DESIDE project, where Solid is used to share permissioned data, however, they increase RML's expressiveness in general. Both HTTP request access and dynamic logical targets can be applied outside the Solid context. HTTP request access can be used for source and target access of any web resource reachable with a standard HTTP request. Dynamic logical targets allow for data-driven fine-grained fragmentation, of any target, e.g. file dumps or DCAT datasets.

Future work includes exploring the unresolved requirements (on-the-fly generation and synchronization of write operations), providing a human-readable version of our mapping solution via YARRRML [7], and pursuing the standardization of our RML extensions in the W3C KG-Construct Community Group.

The synchronization of write operations with the source data requires future research on supporting heterogeneous data sources and supporting configurable views (as opposed to Morph-LDP), as well as the implementation of reverse RML processors and adapted Solid servers. On-the-fly generation of RDF requires investigating trade-offs between storage space, bandwidth, and response times, and the implementation of adapted Solid servers.

The RML descriptions of dynamic logical targets are verbose, in favor of minimizing the introduction of new RML constructs. We plan to investigate the specification of a compact human readable text-based representation with YARRRML.

Finally, we will strive for the standardization of our RML extensions within the RML Community Group and upcoming Work Group, to increase the method's longevity.

Acknowledgments. The described research activities were supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10), and the European Union's Horizon Europe research and innovation program under grant agreement no. 101058682 (Onto-DESIDE).

References

1. Bakerally, N., Zimmermann, A., Boissier, O.: Ldp-dl: A language to define the design of linked data platforms. In: Gangemi, A., Navigli, R., Vidal, M.E., Hitzler, P., Troncy, R., Hollink, L., Tordai, A., Alam, M. (eds.) The Semantic Web. pp. 33–49. Springer International Publishing, Cham (2018)

[23] `https://www.w3.org/community/kg-construct/`
[24] `https://www.w3.org/groups/wg/lws/`

2. Bakerally, N., Zimmermann, A., Boissier, O.: A workflow for generation of ldp. In: Gangemi, A., Gentile, A.L., Nuzzolese, A.G., Rudolph, S., Maleshkova, M., Paulheim, H., Pan, J.Z., Alam, M. (eds.) The Semantic Web: ESWC 2018 Satellite Events. pp. 142–147. Springer International Publishing, Cham (2018)
3. Blomqvist, Eva and Lindecrantz, Mikael and Blomsma, Fenna and Lambrix, Patrick and De Meester, Ben: Decentralized digital twins of circular value networks : a position paper. In: García-Castro, Raúl and Davies, John (ed.) Proceedings of the Third International Workshop on Semantic Digital Twins (SeDiT 2022). vol. 3291, p. 8. CEUR (2022)
4. De Mulder, G., de Vleeschauwer, E., De Meester, B., Colpaert, P., Hartig, O.: The Open Circularity Platform: a Decentralized Data Sharing Platform for Circular Value Networks. In: Proceedings of the 2nd International Workshop on Knowledge Graphs for Sustainability (KG4S), Hersonissos, Greece (May 2024)
5. Dedecker, R., Slabbinck, W., Wright, J., Hochstenbach, P., Colpaert, P., Verborgh, R.: What's in a Pod? A Knowledge Graph Interpretation For The Solid Ecosystem. In: Saleem, M., Ngonga Ngomo, A.C. (eds.) QuWeDa 2022: 6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs. vol. 3279, pp. 81–96 (Oct 2022), https://ceur-ws.org/Vol-3279/paper6.pdf
6. Haase, P., Herzig, D.M., Kozlov, A., Nikolov, A., Trame, J.: metaphactory: A platform for knowledge graph management. Semantic Web **10**(6), 1109–1125 (2019)
7. Heyvaert, P., De Meester, B., Dimou, A., Verborgh, R.: Declarative Rules for Linked Data Generation at your Fingertips! In: The Semantic Web: ESWC 2018 Satellite Events: ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers 15. pp. 213–217. Springer (2018), https://2018.eswc-conferences.org/files/posters-demos/paper_297.pdf
8. Iglesias-Molina, A., Van Assche, D., Arenas-Guerrero, J., De Meester, B., Debruyne, C., Jozashoori, S., Maria, P., Michel, F., Chaves-Fraga, D., Dimou, A.: The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF. In: Proceedings of the International Semantic Web Conference (ISWC). pp. 152–175. Lecture Notes in Computer Science, Springer, Cham (2023). https://doi.org/10.1007/978-3-031-47243-5_9
9. Mihindukulasooriya, N., Priyatna, F., Corcho, O., García-Castro, R., Esteban-Gutiérrez, M.: morph-LDP: An r2rml-based linked data platform implementation. In: Lecture Notes in Computer Science, pp. 418–423. Springer International Publishing (2014). https://doi.org/10.1007/978-3-319-11955-7_59
10. Priyatna, F., Corcho, O., Sequeda, J.: Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph. In: Proceedings of the 23rd international conference on Worldwide Web (WWW). pp. 479–490. WWW '14, Association for Computing Machinery, Seoul, Korea (Apr 2014). https://doi.org/10.1145/2566486.2567981, https://oa.upm.es/36819/1/INVE_MEM_2014_194373.pdf
11. Van Assche, D., Delva, T., Haesendonck, G., Heyvaert, P., De Meester, B., Dimou, A.: Declarative RDF graph generation from heterogeneous (semi-)structured data: A systematic literature review. Journal of Web Semantics (2022). https://doi.org/10.1016/j.websem.2022.100753
12. Van Assche, D., Haesendonck, G., De Mulder, G., Delva, T., Heyvaert, P., De Meester, B., Dimou, A.: Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation. In: Web Engineering, 21st International Conference, ICWE 2021, Proceedings. pp. 337–352 (2021).

https://doi.org/10.1007/978-3-030-74296-6_26, https://dylanvanassche.be/assets/pdf/icwe2021-wot-logical-target.pdf