Extending RML to Support Permissioned Data Sharing with Multiple Views

Els de Vleeschauwer[1], Gerald Haesendonck[1], Ben De Meester[1] and Pieter Colpaert[1]

[1]*IDLab, Department of Electronics and Information Systems,*
Ghent University – imec, Technologiepark-Zwijnaarde 122, 9052 Ghent, Belgium

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, the Fedora Repository, and more broadly the deployment of data spaces. This leads to more applications of Semantic Web technologies and the Resource Description Framework (RDF) in particular. Permissioned RDF data sharing requires access control on multiple levels of granularity. Such fine-grained access control policies require multiple disjoint and overlapping views. Thus, creating multiple disjoint and overlapping views becomes a requirement for knowledge graph construction processes, needed by existing data owners to transform their existing data produced, stored, and managed in their legacy systems. In this paper, we investigate how to manage permissioned data sharing solutions when access control policies become fine-grained, using a declarative knowledge graph construction process. Our main contribution is technically supporting multiple views by introducing and specifying the concept of Dynamic Target in the RDF Mapping Language (RML). Our secondary contribution is applying this to a specific data sharing solution, namely Solid, driven by Onto-DESIDE, a Horizon Europe project on product and material data sharing for increased circular economy. Technically, we solve this by introducing an RML-IO access description to specify the concept of HTTP Request Access with extensible authorization, capable of supporting the Solid protocol. After implementing our RML extensions in RMLMapper and their human-readable YARRRML version in YARRRML Parser, we functionally evaluated that our solution complements the state of the art, and validated our solution on the Onto-DESIDE use cases. 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 the application of our solution to other data sharing solutions, investigating its performance impact, and conducting research on advanced features such as write synchronization with the source data and virtualized permissioned views. We will pursue further alignment with the Solid and RML standards, to increase the method's longevity.

Keywords

Solid, RML, data sharing, access control

1. Introduction

The usage of Semantic Web technologies for data integration has extended from *open* data sharing to *permissioned* data sharing (i.e., requiring authentication and authorization), as exemplified by standardization efforts from, e.g., the Solid project[1], the Fedora Repository[2], and more broadly the increased deployment of data spaces[3]. As this opens doors to more applications of Semantic Web technologies and use of the Resource Description Framework (RDF) [1] in particular, the need for knowledge graph construction processes increases. To achieve such interoperable permissioned data sharing, existing data owners need an RDF transformation process for their existing data produced, stored, and managed in their legacy systems. Today, lack of support for managing fine-grained permissioned data sharing is one of the real-world obstacles that industry actors point to, as exemplified by our motivating use

KGCW'25: 6th International Workshop on Knowledge Graph Construction, June 1st, 2025, Portorož, SLO
els.devleeschauwer@ugent.be (E. de Vleeschauwer); gerald.haesendonck@ugent.be (G. Haesendonck); ben.demeester@ugent.be (B. De Meester); pieter.colpaert@ugent.be (P. Colpaert)
0000-0002-8630-3947 (E. de Vleeschauwer); 0000-0003-1605-3855 (G. Haesendonck); 0000-0003-0248-0987 (B. De Meester); 0000-0001-6917-2167 (P. Colpaert)
© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

[1]https://solidproject.org/
[2]https://fedorarepository.org/
[3]https://www.dataspaces-radar.org/

case: the Onto-DESIDE project[4], a Horizon Europe project on product and material data sharing for increased circular economy.

Permissioned RDF data sharing requires access control on multiple levels of granularity, e.g., privately sharing a specific product's details vs publicly sharing a product portfolio. Specifically, a common case is *data minimization* (i.e., only sharing specific subsets of the same data) tailored to the requesting party. 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, however, legal requirements mandate the disclosure of whether the concentration of certain toxic compounds surpasses the established threshold. Such fine-grained access control policies require multiple disjoint and overlapping views [2]. Thus, creating multiple disjoint and overlapping views becomes a requirement for knowledge graph construction processes, e.g., starting from a single dataset, generating a detailed view per product (multiple disjoint views), combined with a separate summary view over all products (overlapping view).

In this paper, we focus on declaratively defining these views when constructing a knowledge graph. Declarative rules allow to properly inspect and manage views, independent of a specific implementation. Our contribution is an end-to-end declarative solution to specify how non-RDF data from existing data management systems can be shared in multiple, fine-grained views using existing RDF data sharing solutions. Our main research question is how to manage such data sharing solutions when access control policies become fine-grained (e.g., each detailed product view has different access control policies), using declaratively defined views. Our main contribution is technically supporting multiple views by introducing and specifying the concept of **Dynamic Target** in the RDF Mapping Language (RML), one of the most supported declarative mapping languages at this point in time and the only mapping language supporting an export description and thus end-to-end declarative description [3].

Our secondary contribution is applying this to a real-world use case for sharing legacy source datasets in multiple disjoint and overlapping views under different access control policies. We apply our main contribution to a specific data sharing solution, namely Solid (a web decentralization standard designed to give users control over their data) [4] with Web Access Control rules to configure the authorization. This Solid use case is driven by the Onto-DESIDE project goal: accelerate the circular economy by enabling permissioned data sharing using open semantic and technical interoperability standards. Technically, we solve this by complementing the Dynamic Target with an RML-IO access description to specify the concept of **HTTP Request Access** with extensible authorization, capable of supporting the Solid protocol.

We (i) implemented our RML extensions in RMLMapper and their human-readable YARRRML version in YARRRML Parser, (ii) functionally compared our solution to the state of the art, and (iii) 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 for Onto-DESIDE specifically, and in a decentralized data sharing network in general.

The developed RML extensions are generic and can be used in any use case that requires multiple disjoint and overlapping views based on the source data using RML Dynamic Targets, and that requires an end-to-end declarative solution using the HTTP Request Access.

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

In this section, we (i) present the Onto-DESIDE project, our motivating use case; (ii) present declarative mapping solutions to convert existing data to an interoperable format; (iii) discuss permissioned data

[4] https://ontodeside.eu/

sharing solutions, specifically for RDF data; and (iv) discuss related work to map and share existing data using a declarative mapping tool.

In the **Onto-DESIDE project**[5] (Ontology-based Decentralized Sharing of Industry Data in the European Circular Economy) semantic ontologies and Linked Data are combined with the concept of decentralized data sharing to enable data collaboration in the context of circular economy [5]. Circular economy aims to reduce value loss and avoid 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 [6]. 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: RML extensions to enable the end-to-end pipeline from legacy data to permissioned RDF data published in a Solid pod.

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 (in contrast to ad-hoc tools and scripts) [3]. The RDF Mapping Language (RML) – originally extended from the W3C recommended RDB to RDF Mapping Language R2RML [7] – is supported by most knowledge graph construction engines [3] 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. RML is the only mapping language that supports such export descriptions and thus allows declaring an end-to-end solution [3].

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] [9]. 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) [10], decoupling data storage from applications. Individual users can store their data as a web resource in personal online data stores, called 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[8] aims to develop a W3C recommendation based on the work of the Solid Community Group. More broadly, data space initiatives have further intensified the need for (interoperable) data sharing, and specifically, are focussing on the discovery of relevant datasets using a catalogue and access request flow for getting rightful access to a specific dataset [11] (presented as the "control plane"). In that regard, these initiatives can be seen as complementary to the actual data exchange protocol (presented as the "data plane").

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 [12] 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 [13] 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,

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

and comes with a workflow implementation [14] to automate the generation of read-only LDPs from heterogeneous data sources, supporting also virtualized generation of RDF content. Both works use a single configuration 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, discuss to which extent Morph-LDP and the LDP-DL workflow satisfy these requirements, and identify the open issues.

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 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, retaining 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: careful fragmentation positively 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 configuration.

3.2. Complementary Requirements from the State of the Art

Following LDP-DL requirements [13] are not yet covered by Onto-DESIDE's critical features mentioned above.

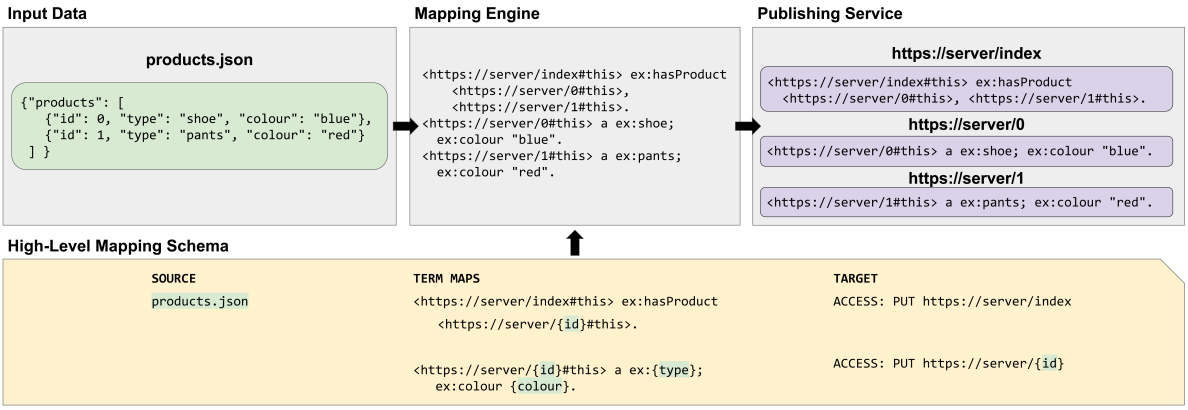

Figure 1: Input data about products is *transformed* to RDF data. This RDF data is *shared* as an index file with an overview of all products (without product details) and one resource per product (with product details). The resources per product allow for access control on product level. In the mapping document, references to the input data are needed to describe the target for the resources per product.

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.

Virtualization [R8] Virtualize data generation, 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[9], 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[10] [15], providing a virtualized knowledge graph by default.

Morph-LDP covers one more requirement [12].

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 via HTTP GET ([R2]), and also writable using HTTP 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.

In Figure 1 we visualize a running example. Input data about products is *transformed* to RDF data. This RDF data is *shared* as an index file with an overview of all products (without product details) and one resource per product (with product details). The resources per product allow for access control on product level.

With the addition of the *logical target* to RML [16], integrated in the new RML specification as part of the RML-IO module[11], 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

[9] https://github.com/noorbakerally/InterLDP
[10] https://github.com/oeg-upm/morph-rdb
[11] http://w3id.org/rml/io/spec/

target must be accessed when exporting RDF triples. To the best of our knowledge, there is no access specification nor implementation for authenticated access to 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 specification with extensible authentication method (Section 4.1), and (ii) the introduction of dynamic logical targets (Section 4.2).

4.1. HTTP Request Access

In RML, access descriptions are specified in the RML-IO-Registry[12]: an RML extension point open to access descriptions specified by the Knowledge Graph Construction Community. The Web of Things access description [13] is an example of such access description, describing access to Web APIs. However, it does not include the authentication nor the access to auxiliary files needed to support the Solid use case. Moreover, its deeply nested structure increases the language complexity without adding value to the Solid use case.

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 HTTP-accessible resources with support for, amongst others, Solid authentication methods (Figure 2)[14].

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 [4]. 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 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* at https://w3id.org/imec/rml/specs/access/httprequest/20250423/, modeling the access to a web resource as an 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 the following main classes: the *HTTP Request* (`rmle:HttpRequest`) and the *User Authentication* (`rmle:UserAuthentication`).

HTTP Request For each *HTTP Request* the method name, headers, and user authentication details can be configured. 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 The *User Authentication* resource holds the authentication details. The required authentication information depends on the authentication type. We specified the *CSS client credentials*

[12] https://kg-construct.github.io/rml-io-registry/
[13] https://kg-construct.github.io/rml-io-registry/wot/index.html
[14] Prefixes are omitted but can be found on https://prefix.cc. Specifically, the prefix `rmle` is used for the proposed RML extensions.
[15] https://www.w3.org/TR/HTTP-in-RDF10/

```
1 <logicaltarget001> a rml:LogicalTarget ;
2   rml:serialization formats:Turtle ;
3   rml:target [
4     a rml:Target, rmle:DirectHttpRequest ;
5     htv:absoluteURI "https://server/index" ;
6     htv:methodName "PUT" ;              # Default method for target
7     htv:headers ([
8         htv:fieldName "content-type" ;  # By default derived from serialization
9         htv:fieldValue "text/turtle" ;
10        )] ;                            # This block may be ommitted
11    rmle:userAuthentication <auth> ;
12  ] .
13 <logicaltarget002> a rml:LogicalTarget ;
14   rml:serialization formats:Turtle ;
15   rml:target [
16     a rml:Target, rmle:LinkedHttpRequest ;
17     rmle:linkingAbsoluteURI "https://server/index" ;
18     rmle:linkRelation "acl" ;
19     htv:methodName "PUT" ;               # Default method for target
20     htv:headers ([
21         htv:fieldName "content-type" ;   # By default derived from serialization
22         htv:fieldValue "text/turtle" ;
23         )] ;                             # This block may be ommitted
24     rmle:userAuthentication <auth>
25   ].
26 <auth> a rmle:CssClientCredentialsAuthentication ;
27   rmle:authEmail "hello@server.com" ;
28   rmle:authPassword "abc123" ;
29   rmle:authWebId <https://server/profile/card#me> ;
30   rmle:authOidcIssuer <https://server/> .
```

Figure 2: Logical target descriptions for Solid resources. The direct HTTP request, i.e., the target of `<logicaltarget001>`, adds a new resource to a Solid pod (lines 4–10), using an HTTP PUT request on `https://server/index`. The linked HTTP request, i.e., the target of `<logicaltarget002>`, adds a new ACL file for a resource in a Solid pod (lines 13–18), using an HTTP PUT request to the URI discovered by making an HTTP HEAD request on `https://server/index`, and extracting the value of the HTTP Link header with relation `acl`. Both target descriptions include authentication details (lines 8, 18, and 20–24) to properly execute authenticated HTTP requests.

authentication, describing the information required for authentication using client credentials as specified in the Community Solid Server (CSS) 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 in more than one resource) are needed to satisfy the needs of real world use cases requiring fine-grained access control [2]. 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**

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

based on source data, e.g., define a logical target per product. Existing workarounds[17] allow defining multiple logical targets 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 at https://w3id.org/imec/rml/specs/target/dynamictarget/20250113/. An example in RML mapping rules is presented in Figure 3. The tree-steps solution is elaborated in the following paragraphs.

```
1  <triplesmap001> a rml:TriplesMap;
2    rml:logicalSource <logicalsource001>;
3    rml:subjectMap [
4      rml:template "http://logicaltarget/{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 "http://target/{id}"
11     ];
12   ].
13 <triplesmap002> a rml:TriplesMap;
14   rml:logicalSource <logicalsource001>;
15   rml:subjectMap [
16     rml:template "http://target/{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 "https://server/{id}";
24       rml:termType rml:Literal;
25     ];
26   ];
27   rml:predicateObjectMap [
28     rml:predicate rmle:userAuthentication;
29     rml:objectMap <auth>;
30   ].
31 <triplesmap003> a rml:TriplesMap;
32   rml:logicalSource <logicalsource001>;
33   rml:subjectMap [
34     rml:template "https://server/{id}#this";
35     rml:class "https://example.com/{type};
36     rmle:logicalTargetMap [
37       rml:template "http://logcialtarget/{id}"
38     ];
39   ].
```

Figure 3: Triples maps `<triplesmap001>` and `<triplesmap002>` generate respectively 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) of `<triplesmap003>` generates URIs matching the identifiers of generated logical targets, securing the creation of web resources per product.

Step 1: Generate Logical Targets using RML. All RML statements, also logical targets, are a set of RDF triples. Consequently, it is possible to define RML mapping rules to generate logical targets

[17] https://kg-construct.github.io/rml-io/spec/docs/#conditions

based on source data (Figure 3, 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 possible 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 3 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 3 lines 36–38). The logical target map is aligned with other dynamic mapping constructs 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 3 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.

With this approach, it is also possible to define views on subgroups of products based on source data that is not transformed to RDF. If an actor wants to only share specific products to specific partners, he can manage this in an internal list, e.g., product A and B are shared with partner X, product D is shared with partners Y and Z. The resulting RDF product data should not include information from this internal list, however, the list can be used during the mapping process to determine which product should end up in which view.

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 specified and implemented their human-readable YARRRML versions (Section 5.2), we built a demonstrator to showcase how the requirements are satisfied (Section 5.3), and we applied our solution in the Onto-DESIDE project (Section 5.4).

5.1. Implementation in RMLMapper

We implemented HTTP request access and dynamic targets in RMLMapper, a Java library for executing RML rules to generate RDF data, published at https://github.com/RMLio/rmlmapper-java/releases/tag/v7.3.1. 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[18].

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

We focused the implementation of the HTTP request access for targets[19], satisfying the needs of our motivating use case Onto-DESIDE. HTTP request access for sources[20] 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. The triples for 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.

5.2. Implementation in YARRRML Parser

```yaml
targets:
    - productid:
        source: logicalsource001
        type: directhttprequest
        access: "http://server/$(id)"
        authentication: auth
authentications:
    - auth:
       type: cssclientcredentials
       email: hello@server.com
       password: abc123
       webId: "https://server/profile/card#me"
       oidcIssuer: "https://server/"
mappings:
    - products:
        sources: logicalsource001
        subjects:
            value: "https://server/$(id)#this"
            targets:
                - productid
        predicateobjects:
            - [a, "https://example.com/$(type)"]
```

Figure 4: The YARRRML representation for the dynamic logical target and mapping shown in Figure 3 and for the authentication shown in Figure 2.

The RML descriptions of dynamic logical targets are verbose, in favor of minimizing the introduction of new RML constructs and thus limit the language complexity's increase. To assist developers of RML rules, we have integrated dynamic logical targets in YARRRML [17], a human-readable text-based representation for declarative Linked Data generation rules, that can be used to used to represent RML rules. We provide a specification for Dynamic Targets and HTTP Request Access at https://w3id.org/imec/rml/yarrrml/spec/target/dynamictarget/20250226/ and https://w3id.org/imec/rml/yarrrml/spec/access/httprequest/20250312/, respectively. A YARRRML representation of a dynamic logical target with HTTP request access is shown in Figure 4. We represent dynamic logical targets as targets with an additional `source` and optional `id` key. The value of the other keys of a dynamic target are *dynamic*

[19] https://w3id.org/imec/rml/specs/access/httprequest/#http-request-as-target
[20] https://w3id.org/imec/rml/specs/access/httprequest/#http-request-as-source

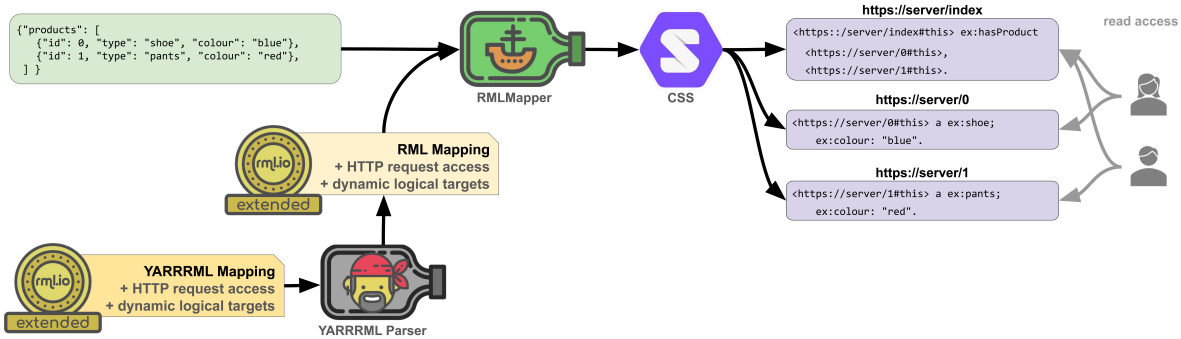

Figure 5: End-to-end pipeline allowing permissioned data sharing from existing source data: YARRRML Parser converts the extended YARRRML mapping to an extended RML mapping. RMLMapper processes heterogeneous data sources using the extended RML mapping, and exports the generated RDF data to overlapping and disjoint resources hosted on a CSS Solid pod.

Table 1
The demonstrator pipeline satisfies all requirements for the Onto-DESIDE use case (R1-R6, put in bold). Virtualization and write access, 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 Virtualization	✓	✓	–
R9 Technical interoperability: write access	✓	–	–

templates instead of *static* strings. The triples maps for generating the dynamic logical targets in RML are constructed by the YARRRML processor that converts the YARRRML rules to RML rules.

We implemented both specifications in YARRRML Parser, a Javascript library to convert YARRRML rules to RML rules, published at https://github.com/RMLio/yarrrml-parser/releases/tag/v1.10.0. Also for this implementation we have focused on the needs of our motivating use case Onto-DESIDE and plan the HTTP request access for sources for a future release.

5.3. Demonstrator

We built a demonstrator to showcase how our solution answers the requirements listed in Section 3, published at https://github.com/RMLio/rml-solid-demonstrator/releases/tag/v1.0.1. 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 YARRRML 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 5 we convert the extended YARRRML mappings to extended RML mappings, and 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 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 the data of these CSV files to ACL rules. Linked HTTP requests (Section 4.1) allow 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 into *virtualization* **[R8]** of the views, and to *synchronize write operations in the exposed views with the source data* **[R9]**. The implementation of virtualization 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).

5.4. 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 [6]. 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, which in turn enables 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. We validated our solution's functionality through workshops with the industry partners of the Onto-DESIDE project, i.a. Lindner, Ragn-Sells and Texon, using the think-aloud method [18]. The industry partners defined evaluation scenarios and approved their implementations. The results of these evaluations are publicly reported through Onto-DESIDE's deliverables[21].

6. Conclusion

To manage permissioned data sharing solutions with multiple overlapping and disjoint views, we facilitate a dynamic data-driven description of publication targets in a declarative Knowledge Graph Construction solution. We presented and implemented two RML extensions: (i) HTTP request access to directly publish RDF as online (Solid) resources; and (ii) dynamic logical targets to create data-driven overlapping and disjoint views. These RML extensions – implemented in RMLMapper and YARRRML Parser – are integrated in a demonstrator, functionally compared to the state-of-the-art and validated with the real-world Onto-DESIDE use cases. All resources presented in this paper are published on Zenodo: https://doi.org/10.5281/zenodo.14507297.

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 active W3C community groups[22] and working group[23], rather than developing a standalone custom solution, we aim to 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 showcasing the generic character of our solution, investigating the performance impact, exploring the unresolved requirements (virtualization and synchronization of write operations), and pursuing the standardization of our RML extensions in the W3C KG-Construct Community Group.

We plan to showcase the generic character of our solution by applying it to other data sharing solutions with access control, e.g., data spaces, and to data organization strategies independent of the access control requirement, e.g., triple pattern fragments.

Performance was not a focus point in this paper, but should be once data itself becomes large/streaming, for which we plan to integrate our previous work on continuous RDF generation using RML and Linked Data Event Streams [19, 20], and the RMLStreamer [21]. For the latter, as we cannot rely on in-memory RML processors, the complexity of prioritizing the dynamic target triple generation must be taken into account.

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. Virtualization of RDF requires investigating trade-offs between storage space, bandwidth, and response times, and the implementation of adapted Solid servers.

Finally, we will strive for the standardization of our RML extensions within the KGC 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).

[21] https://ontodeside.eu/wp-content/uploads/2024/10/Onto-DESIDE_WP6_Deliverable_D6.8_final.pdf
[22] https://www.w3.org/community/kg-construct/
[23] https://www.w3.org/groups/wg/lws/

References

[1] R. Cyganiak, D. Wood, M. Lanthaler, RDF 1.1 Concepts and Abstract Syntax, Recommendation, World Wide Web Consortium (W3C), 2014. URL: https://www.w3.org/TR/rdf11-concepts/.

[2] R. Dedecker, W. Slabbinck, J. Wright, P. Hochstenbach, P. Colpaert, R. Verborgh, What's in a Pod? A Knowledge Graph Interpretation For The Solid Ecosystem, in: M. Saleem, A.-C. Ngonga Ngomo (Eds.), QuWeDa 2022: 6th Workshop on Storing, Querying and Benchmarking Knowledge Graphs, volume 3279, 2022, pp. 81–96. URL: https://ceur-ws.org/Vol-3279/paper6.pdf.

[3] D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester, A. Dimou, Declarative RDF graph generation from heterogeneous (semi-)structured data: A systematic literature review, Journal of Web Semantics (2022). doi:10.1016/j.websem.2022.100753.

[4] S. Capadisli, T. Berners-Lee, R. Verborgh, K. Kjernsmo, Solid Protocol, Technical Report, Solid Community Group, 2022. URL: https://solidproject.org/TR/protocol.

[5] 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), volume 3291, CEUR, 2022, p. 8. URL: https://ceur-ws.org/Vol-3291/paper1.pdf.

[6] G. De Mulder, E. de Vleeschauwer, B. De Meester, P. Colpaert, O. Hartig, 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, 2024. URL: https://ceur-ws.org/Vol-3753/paper4.pdf.

[7] S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, Working Group Recommendation, World Wide Web Consortium (W3C), 2012. URL: http://www.w3.org/TR/r2rml/.

[8] A. Iglesias-Molina, D. Van Assche, J. Arenas-Guerrero, B. De Meester, C. Debruyne, S. Jozashoori, P. Maria, F. Michel, D. Chaves-Fraga, A. Dimou, 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), Lecture Notes in Computer Science, Springer, Cham, 2023, pp. 152–175. doi:10.1007/978-3-031-47243-5_9.

[9] P. Haase, D. M. Herzig, A. Kozlov, A. Nikolov, J. Trame, metaphactory: A platform for knowledge graph management, Semantic Web 10 (2019) 1109–1125. doi:10.3233/SW-190360.

[10] S. Speicher, A. Malhotra, J. Arwe, Linked Data Platform 1.0, Recommendation, World Wide Web Consortium (W3C), 2015. URL: http://www.w3.org/TR/ldp/.

[11] S. Steinbuss, P. Koen, M. Kollenstart, J. Marino, J. Pampus, A. Turkmayali, A. Weiß, Dataspace Protocol 2025-1-RC1, Technical Report, 2025. URL: https://eclipse-dataspace-protocol-base.github.io/DataspaceProtocol/.

[12] N. Mihindukulasooriya, F. Priyatna, O. Corcho, R. García-Castro, M. Esteban-Gutiérrez, morph-LDP: An r2rml-based linked data platform implementation, in: Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 418–423. doi:10.1007/978-3-319-11955-7_59.

[13] N. Bakerally, A. Zimmermann, O. Boissier, LDP-DL: A Language to Define the Design of Linked Data Platforms, in: A. Gangemi, R. Navigli, M.-E. Vidal, P. Hitzler, R. Troncy, L. Hollink, A. Tordai, M. Alam (Eds.), The Semantic Web, Springer International Publishing, Cham, 2018, pp. 33–49. doi:10.1007/978-3-319-93417-4_3.

[14] N. Bakerally, A. Zimmermann, O. Boissier, A Workflow for Generation of LDP, in: A. Gangemi, A. L. Gentile, A. G. Nuzzolese, S. Rudolph, M. Maleshkova, H. Paulheim, J. Z. Pan, M. Alam (Eds.), The Semantic Web: ESWC 2018 Satellite Events, Springer International Publishing, Cham, 2018, pp. 142–147. doi:10.1007/978-3-319-98192-5_27.

[15] F. Priyatna, O. Corcho, J. Sequeda, Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph, in: Proceedings of the 23rd international conference on Worldwide Web (WWW), WWW '14, Association for Computing Machinery, Seoul, Korea, 2014, pp. 479–490. URL: https://oa.upm.es/36819/1/INVE_MEM_2014_194373.pdf. doi:10.1145/2566486.2567981.

[16] D. Van Assche, G. Haesendonck, G. De Mulder, T. Delva, P. Heyvaert, B. De Meester, A. Dimou,

Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation, in: Web Engineering, 21st International Conference, ICWE 2021, Proceedings, 2021, pp. 337–352. doi:10.1007/978-3-030-74296-6_26.

[17] P. Heyvaert, B. De Meester, A. Dimou, R. Verborgh, Declarative Rules for Linked Data Generation at your Fingertips!, in: A. Gangemi, A. L. Gentile, A. G. Nuzzolese, S. Rudolph, M. Maleshkova, H. Paulheim, J. Z. Pan, M. Alam (Eds.), The Semantic Web: ESWC 2018 Satellite Events: ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers 15, volume 11155 of *Lecture Notes in Computer Science*, Springer, Springer, Cham, Heraklion, Crete, Greece, 2018, pp. 213–217. doi:10.1007/978-3-319-98192-5_40.

[18] M. W. van Someren, Y. Barnard, J. Sandberg, The think aloud method: A practical guide to modelling cognitive processes, Knowledge-based systems, Londen: Academic Press, 1994. URL: http://hdl.handle.net/11245/1.103289.

[19] D. Van Assche, S. M. Oo, J. A. Rojas, P. Colpaert, Continuous generation of versioned collections' members with RML and LDES, in: Proceedings of the 3rd International Workshop on Knowledge Graph Construction (KGCW 2022) co-located with 19th Extended Semantic Web Conference (ESWC 2022), 2022. URL: https://ceur-ws.org/Vol-3141/paper3.pdf.

[20] G. Haesendonck, B. De Meester, J. A. Rojas Meléndez, D. Van Assche, P. Colpaert, YARRRML + LDES: Simultaneously Lowering Complexity from Knowledge Graph Generation and Publication, in: Proceedings of the ISWC 2023 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 22nd International Semantic Web Conference (ISWC 2023), 2023. URL: https://ceur-ws.org/Vol-3632/ISWC2023_paper_403.pdf.

[21] Sitt Min Oo, G. Haesendonck, B. De Meester, A. Dimou, RMLStreamer-SISO: An RDF Stream Generator from Streaming Heterogeneous Data, in: U. Sattler, A. Hogan, M. Keet, V. Presutti, J. P. A. Almeida, H. Takeda, P. Monnin, G. Pirrò, C. d'Amato (Eds.), The Semantic Web – ISWC 2022, Springer, Springer International Publishing, Cham, 2022, pp. 697–713. doi:10.1007/978-3-031-19433-7_40.