# Declarative RDF graph generation from heterogeneous (semi-)structured data: a Systematic Literature Review

Dylan Van Assche[a,*], Thomas Delva[a], Gerald Haesendonck[a], Pieter Heyvaert[a], Ben De Meester[a] and Anastasia Dimou[b,**]

[a]IDLab, Department of Electronics and Information Systems, Ghent University – imec, Technologiepark-Zwijnaarde 122, 9052 Ghent, Belgium
[b]KU Leuven, Department of Computer Science, Jan-Pieter De Nayerlaan 5, 2860 Sint-Katelijne-Waver, Belgium

ARTICLE INFO

*Keywords*:
knowledge graph construction
schema transformations
data transformations
survey
declarative

ABSTRACT

More and more data in various formats are integrated into knowledge graphs. However, there is no overview of existing approaches for generating knowledge graphs from heterogeneous (semi-)structured data, making it difficult to select the right one for a certain use case. To support better decision making, we study the existing approaches for generating knowledge graphs from heterogeneous (semi-)structured data relying on mapping languages. In this paper, we investigated existing mapping languages for schema and data transformations, and corresponding materialization and virtualization systems that generate knowledge graphs. We gather and unify 52 articles regarding knowledge graph generation from heterogeneous (semi-)structured data. We assess 15 characteristics on mapping languages for schema transformations, 5 characteristics for data transformations, and 14 characteristics for systems. Our survey paper provides an overview of the mapping languages and systems proposed the past two decades. Our work paves the way towards a better adoption of knowledge graph generation, as the right mapping language and system can be selected for each use case.

## 1. Introduction

Over the past decades, the amount of data published on the Web significantly increased, and more and more of these data are integrated into knowledge graphs [61]. Knowledge graphs allow integrating heterogeneous data to make decisions, provide recommendations or deduct new knowledge. Knowledge graphs are leveraged in different domains, e.g., media [111], government [117, 62], search engines [121, 118], commerce [80, 106], social networks [103, 58], etc.

As we witness an increasing use of knowledge graphs [61], it becomes essential to better understand the various aspects of their generation, as well as their strengths and weaknesses. While there are several approaches for generating knowledge graphs, the domain has not been systematically studied so far. Therefore, it remains challenging to identify the most adequate solution for each use case.

Knowledge graphs are generated in different ways. They can be generating via crowdsourcing by collecting contributions from a large group of people, e.g., Wikidata[1], or they can be extracted from (semi-)structured data, for instance tabular (e.g., tables in relational databases or data in CSV format) or hierarchical data (e.g., data in XML or JSON format), or unstructured data, e.g., plain text. In this paper, we focus on the generation of knowledge graphs from heterogeneous (semi-)structured data.

Several approaches for generating knowledge graphs from heterogeneous (semi-)structured data are use-case-specific [121, 80, 106, 103, 58, 118, 24, 43, 56]. However, these approaches cannot be reused. Therefore, in this work, we look into approaches that are *use-case-agnostic*. Such approaches typically rely on *mapping languages* which declaratively describe how knowledge graphs should be generated from heterogeneous (semi-)structured data.

Various studies were conducted in the past but none covered all aspects of knowledge graph generation. A few of these studies are limited to data of a specific format, such as relational databases, e.g., [57, 59, 50, 123], or XML data, e.g., [14]. Others are even more specific and not only do they focus on a certain data format, but also on a particular approach for generating knowledge graphs, such as virtualization, e.g., [133, 134]. Most of these surveys were conducted following an ad-hoc methodology, and only present partial results related to knowledge graph generation.

Recently, two systematic reviews were published which approach the knowledge graph generation as part of the knowledge graph development process but they do not focus on the knowledge graph generation per se. The former [113] is a systematic review of studies on knowledge graph generation and publication. The review only considered two conferences (ESWC and ISWC) and two journals (SWJ and JWS), and only a limited period in time (2015 - 2021). The latter [125] conceptually analyses the literature, identifies the key processes when managing the construction and maintenance of knowledge graphs and provides a synthesis of common steps in knowledge graph development.

Lacking of rigorous surveys, only benchmarks [27, 6, 25] are available for comparing knowledge graphs generation approaches from heterogeneous (semi-)structured data. However, benchmarks evaluate the performance of the systems, but not the declarative mapping language or the schema

*Corresponding authors
*Email addresses:* dylan.vanassche@ugent.be (D. Van Assche); anastasia.dimou@kuleuven.be (A. Dimou)
ORCID(s): 0000-0002-7195-9935 (D. Van Assche); 0000-0001-9521-2185 (T. Delva); 0000-0003-1605-3855 (G. Haesendonck); 0000-0002-1583-5719 (P. Heyvaert); 0000-0003-0248-0987 (B. De Meester); 0000-0003-2138-7972 (A. Dimou)

[1]https://www.wikidata.org, last accessed 07/04/2022

and data transformation. Moreover, benchmarks limit their scope to evaluating systems that implement a single mapping language [25, 6] or a subset of the available systems [27].

There is no survey yet which provides an overview of all existing approaches for knowledge graph generation, such as materializing the complete knowledge graph (*materialization*) or virtualizing parts of the knowledge graph (*virtualization*) nor a survey for all mapping languages proposed.

Therefore, we created this survey to provide an overview of mapping languages and systems to help users choose the mapping language for the data and schema transformations, as well as their systems for a given use case. In this survey, we answer the following research question:

*Which characteristics influence mapping languages and systems for generating RDF knowledge graphs from heterogeneous (semi-)structured data?*

To answer the main research question, the following research subquestions need to be addressed:

RQ1 *Which mapping languages exist to generate knowledge graphs from (semi-)structured heterogeneous data?*

RQ2 *How are schema and data transformations supported by mapping languages and what RDF terms can each mapping language generate?*

RQ3 *How are the mapping languages implemented in different systems to generate RDF graphs?*

With this systematic literature review, we aim to provide an overview of existing works that can help researchers and practitioners to find an optimal solution for their use case. The survey also aims to identify remaining open aspects that are worth further investigation and inspire researchers to experiment further, discover new approaches, and implement new algorithms in their systems.

This paper is structured as follows: in Section 3 we describe the survey methodology we followed in this systematic review. Section 4 provides an overview of different approaches to generate knowledge graphs from heterogeneous (semi-)structured data. In Section 5 we compare different approaches for data transformations. Section 6.2 describes the materialization approaches and their systems. Section 6.3 describe the virtualization approaches and systems. Section 7 discusses the results of the systematic review. Section 8 concludes this survey paper with further work.

## 2. Definitions

In this Section, we review the basic concepts and definitions that we need for the systematic literature review in the rest of the paper.

The problem of transforming data structured under one schema into data structured under a different schema is encountered in several different areas of data management systems. *Schema mappings* are specifications used to describe how data is to be transformed from one representation to another. *Schema mappings* are typically specified using declarative formalisms that describe the correspondence between different schemas at a logical level, without specifying details relevant for the implementation [49]. A *data integration* system combines data residing at different sources, and provides the user with a unified view of these data [88].

A *data integration system* $\mathcal{I}$ is a triple $\mathcal{I} = (\mathcal{T}, \mathcal{S}, \mathcal{M})$ where $\mathcal{T}$ is the *target schema*, expressed in a language $\mathcal{L}_\mathcal{T}$ over an alphabet $\mathcal{A}_\mathcal{T}$, $\mathcal{S}$ is the *source schema*, expressed in a language $\mathcal{L}_\mathcal{S}$ over an alphabet $\mathcal{A}_\mathcal{S}$ [88]. A *schema* is a finite sequence $\mathcal{R} = \langle \mathcal{R}_1, ..., \mathcal{R}_k \rangle$ of distinct relation symbols, each of a fixed arity. Given two disjoint schemas $\mathcal{S}$ and $\mathcal{T}$ with no relation symbols in common, then $\mathcal{S} = \langle \mathcal{S}_1, ..., \mathcal{S}_n \rangle$ is the *source schema* and $\mathcal{T} = \langle \mathcal{T}_1, ..., \mathcal{T}_m \rangle$ the *target schema*. A *schema mapping* is a triple $\mathcal{M} = (\mathcal{S}, \mathcal{T}, \Sigma)$, where $\Sigma$ is a set of formulas of some logical formalism over $\langle \mathcal{S}, \mathcal{T} \rangle$.

The generation of a knowledge graph can be considered as a *mapping* $\mathcal{M}$ between a source schema $\mathcal{S}$, which is the schema of original data source $\mathcal{D}$, and a target schema $\mathcal{T}$, which is the schema of the knowledge graph. In this paper, we focus on the generation of RDF graphs, thus RDF is the language $\mathcal{L}_\mathcal{T}$ of the target schema $\mathcal{T}$ which, on its own turn, is a vocabulary or ontology. The mapping between the source schema $\mathcal{S}$ and the target schema $\mathcal{T}$ follows a certain *schema mapping* whose set of formulas $\Sigma$ is described with a so-called *mapping language*. A *mapping language* describes different types of mappings between a *source schema* and a *target schema* and provides a means for linking a particular data source to its specific mapping policy [4]. In the remaining of this paper, we call the aforementioned mappings *schema transformations* to distinguish them from the *data transformations*. A schema transformation describes how objects are related, and which vocabularies and ontologies are used to annotate the objects [63]. On the contrary, a data transformation describe any changes in the structure, representation or content of data [110].

A system that generates a knowledge graph may be considered as a data integration system $\mathcal{I}$. In the remaining of the paper, we distinguish between *materialization* and *virtualization* systems for knowledge graph generation. In the databases field, a view is a query whose head defines a new database relation [130]. If this relation is not stored, the view is *a virtual view*, while if the results of executing the view are stored, it is a *materialized view* [130]. Similarly in the case of knowledge graphs, a *materialised knowledge graph* is stored, while a *virtual knowlede graph* is not stored.

## 3. Survey methodology

We first searched for recent survey papers to compare different survey methodologies [136, 65, 131, 2] to find the most suitable survey methodology for this paper. We follow the methodology of Kitchenham B. [77], similar to Amrapali et al. [136] to discover and select relevant papers. We choose to follow the same methodology because we also collected and combined papers from various sources such as digital libraries, workshops, journals and conferences while other survey papers focus on only one of these sources. Moreover, they follow existing survey methodologies for their systematic review [78, 101]. They clearly describe how they searched for articles, the source of the articles, and why they

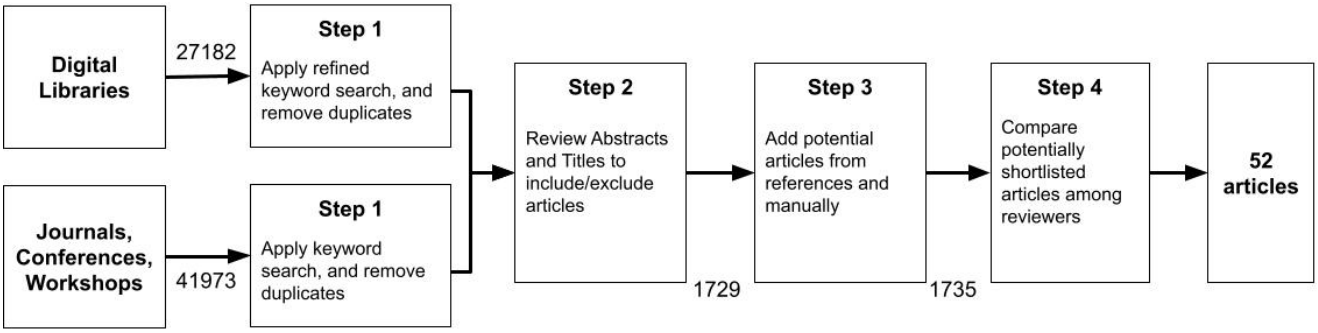


**Figure 1:** Overview of the applied survey methodology to select relevant articles for this survey paper.

were included or not. We extended this survey methodology with an additional step to limit the number of results of digital libraries by refining the keywords.

Three authors independently performed this systematic review following the survey methodology described in [77]. After applying our survey methodology, we retrieved 1735 potentially relevant papers published between 1999 and 2021 from which 52 were relevant for this survey. We studied and unified these papers by categorizing them into the following categories: '*schema transformations*', '*data transformations*', '*materialization systems*', and '*virtualization systems*'. We analyzed 15 characteristics for mapping languages, 5 characteristics for data transformations, and 14 characteristics for systems.

A systematic literature review is performed for several reasons [77]. In this survey paper, we tackle the following reasons: (i) summarizing and comparing existing approaches, (ii) identifying open problems and (iii) conceptualization of various approaches. We compare existing approaches and systems implementing these approaches to generate knowledge graphs from heterogeneous (semi-)structured data and identify open problems. We describe the applied methodology below and visualize it in Figure 1.

## Inclusion and exclusion criteria

We consider the following criteria to include an article in this survey:

- Articles published from the first published recommendation of RDF [79] in 1999 until 2021.
- Articles written in English.
- Articles describing mapping languages and systems related to:
  - Generating knowledge graphs from multiple heterogeneous data sources.
  - Describing systems and optimization algorithms of existing knowledge graph generation approaches.
  - Specifying schema or data transformation for heterogeneous data.

We excluded an article from this survey based on the following exclusion criteria:

- Articles which are not published or peer-reviewed.
- Papers written in another language than English.
- Technical reports, demo papers, posters, PhD consortium papers, and presentations.
- Commercialized approaches and systems.

## Step 1: Collecting articles

We created a list of terms by breaking down our research sub-questions into individual facets and their alternatives, e.g., different spelling, synonyms, and abbreviations. These terms were combined together with boolean `AND` and `OR` operators and used by each reviewer when searching for relevant articles. We avoided to use these terms individually because they were too broad which resulted in an enormous number of results. The following terms and combinations were considered in this systematic review:

- `('knowledge graph' OR 'knowledge graphs') AND ('definition' OR 'definitions')`
- `('RDF' OR 'Resource Description Framework') AND 'heterogeneous data'`
- `('transformation' OR 'transformations') AND 'heterogeneous data'`
- `('answer' OR 'answers' OR 'answering') AND ('query' OR 'queries')`
- `('Resource Description Framework' OR 'RDF') AND ('stream' OR 'streams' OR 'streaming')`
- `('Linked Data' OR 'Linked Open Data' OR 'LOD' OR 'knowledge graph' OR 'knowledge graphs') AND 'generation'`
- `('Linked Data' OR 'Linked Open Data' OR 'LOD' OR 'knowledge graph' OR 'knowledge graphs') AND 'construction'`
- `'SPARQL' AND ('generation' OR 'construction')`
- `('mapping language' OR 'mapping languages') AND ('RDB' OR 'relational database' OR 'relational databases')`
- `('mapping language' OR 'mapping languages') AND 'heterogeneous data'`
- `('mapping language' OR 'mapping languages') AND ('function' OR 'functions')`
- `('mapping language' OR 'mapping languages') AND ('transformation' OR 'transformations')`
- `('mapping language' OR 'mapping languages') AND ('RDF' OR 'Resource Description Framework')`

- `('mapping language' OR 'mapping languages') AND ('knowledge graph' OR 'knowledge graphs' OR 'Linked Data' OR 'Linked Open Data' OR 'LOD')`
- `('mapping language' OR 'mapping languages') AND ('knowledge graph' OR 'knowledge graphs' OR 'Linked Data' OR 'Linked Open Data' OR 'LOD') AND ('generation' OR 'construction')`
- `('mapping language' OR 'mapping languages') AND 'SPARQL'`
- `('Linked Data' OR 'Linked Open Data' OR 'LOD') AND ('mapping language' OR 'mapping languages')`
- `('mapping language' OR 'mapping languages') AND 'R2RML'`
- `('generation' OR 'construction') AND 'R2RML' AND ('knowledge graph' OR 'knowledge graphs' OR 'Linked Data' OR 'Linked Open Data' OR 'LOD')`
- `('stream' OR 'streams' OR 'streaming') AND ('RDF' OR 'Resource Description Framework') AND 'SPARQL'`

### Step 2: Review titles & abstracts

We searched for these terms in the article's title and abstract to avoid a high number of irrelevant articles in comparison to searching in the article's full text [136]. An article is selected when at least one of the combined terms has been found in the title or abstract of an article. We searched for our terms in 42 sources e.g. journals, conferences, and digital libraries. The following sources were used to search for relevant articles for this survey paper:

*Journals*
- Semantic Web Journal (SWJ)
- Journal of Web Semantics (JoWS)
- International Journal of Web Information Systems (IJWIS)
- Future Generation Computer Systems (FGCS)
- Journal on Data Semantics (JDS)
- International Journal on Semantic Web and Information Systems (IJSWIS)
- PeerJ Computer Science
- Information
- International Journal on Digital Libraries (IJDL)
- International Journal of Applied Mathematics and Computer Science (IJAMCS)
- International Journal of Software Engineering and Knowledge Engineering (IJSEKE)

*Conferences*
- International Semantic Web Conference (ISWC)
- European Semantic Web Conference (ESWC)
- International Conference on Semantic Systems (SEMANTiCS/I-SEMANTICS)
- International Conference on Semantic Computing (ICSC)
- International Conference on Knowledge Engineering and Knowledge Management (EKAW)
- International Conference on Knowledge Capture (K-Cap)
- Conference on Information and Knowledge Management (CIKM)
- Knowledge Graph and Semantic Web Conference (KGSWC)
- The Web Conference (WWW)
- Ontologies, Databases and Applications of Semantics (ODBASE)
- Language, Data and Knowledge (LDK)
- International Conference on Web Intelligence (WI-IAT)
- International Conference on Web Information Systems and Technologies (WEBIST)
- International Conference on Web Intelligence, Mining and Semantics (WIMS)
- International Conference on Web Engineering (ICWE)

*Workshops*
- Knowledge Graph Construction Workshop (KGCW)
- Knowledge Graph Building Workshop (KGB)
- Consuming Linked Data Workshop (COLD)
- Large Scale RDF Analytics Workshop (LASCAR)
- Semantic Big Data Workshop (SBD)
- International Semantic Sensor Networks Workshop (SSN)
- Workshop on Linked Data on the Web (LDOW)

*Digital Libraries*
- IEEE Explore
- Springer Link
- Science Direct
- ACM Digital Library
- Google Scholar

We retrieved 1884 articles which are potentially relevant for this survey paper. After removing duplicates, 1729 articles were added to the review list.

### Step 3: Applying additional search strategies

We enriched our review list following additional search strategies to include more relevant articles: (i) reviewing references of potentially relevant articles, and (ii) adding manually relevant articles. We searched for potentially relevant articles through the references of other relevant articles which were in our review list. This way, we discovered 3 more potentially relevant articles [41, 26, 39]. We also manually added 3 more articles which we found relevant for this survey [32, 85, 116]. This result in 1735 articles in total.

### Step 4: Reviewing potential articles for inclusion

We manually reviewed the retrieved potentially relevant articles among reviewers. The reviewers verified the article's relevance for this systematic literature review and applied our inclusion and exclusion criteria. This resulted in 52 relevant articles for this survey paper (Table 1). 18 articles introduce or extend a mapping language for transforming heterogeneous (semi-)structured data into a knowledge graph [98, 86, 36, 30, 132, 87, 83, 81, 128, 15, 46, 90, 41, 51, 135, 104, 19]. 16 articles introduce data transformations or align them with a mapping language for heterogeneous (semi-)structured data [99, 86, 38, 132, 87, 107, 70, 73, 32,

15, 90, 41, 40, 95, 39, 7]. 21 articles describe a materialization system [85, 86, 30, 132, 107, 122, 87, 83, 81, 115, 45, 72, 54, 46, 114, 120, 64, 51, 98, 96, 116], and 18 articles a virtualization system for transforming heterogeneous (semi-)structured data into a knowledge graph [99, 104, 19, 89, 126, 26, 18, 48, 75, 112, 22, 13, 76, 52, 93, 135, 21]. 12 articles were submitted to workshops, 27 to conferences, and 13 to journals.

## 4. Schema transformations

In this Section, we discuss mapping languages as schema transformation descriptions to generate RDF graphs from heterogeneous data. Several schema transformation descriptions exist each with their own set of features and supported data formats and sources. Currently, there is no overview to select the right schema transformation for a given use case, which we address in this Section.

Section 4.1 discusses the categorization we applied for schema transformations, Section 4.2 describes which characteristics we evaluate for each schema transformation. Section 4.5 mentions schema transformations which are worth mentioning but could not be included because of the inclusion and exclusion criteria. The next sections discuss the schema transformations per category.

### 4.1. Categorization

We categorized the mapping languages (Table 1) in two categories: (i) dedicated mapping languages which extend specifications or use a custom syntax, and (ii) repurposed languages which repurpose existing specifications of query or constraint languages as their base for a mapping language.

*Dedicated mapping languages* extend existing mapping language specifications, or provide a custom syntax. On the one hand, a family of mapping languages have as their basis the W3C-recommended RDB to RDF Mapping Language (R2RML) [35] and extend its scope from relational databases to heterogeneous (semi-)structured data. R2RML based mapping languages use Turtle as syntax for writing mapping rules. RML [46], xR2RML [98] and D2RML [30] belong in this category. Other dedicated mapping languages provide their own custom syntax, such as the Dataset Representation (D-REPR) written in YAML [132], and Ontop [135].

*Repurposed languages* repurpose existing specifications which have a different purpose than mapping languages, such as (i) *query languages*, e.g., SPARQL [109] (query-language-driven), and (ii) *constraint languages*, e.g., Shape Expressions (ShEx) [108] (constraint-driven).

*Query-language-driven mapping languages* consider a query language, such as the W3C-recommended SPARQL query language [109], as their basis and extend the query language to generate RDF from heterogeneous (semi-)structured data. The following mapping languages belong in this category: XSPARQL [15, 41], SPARQL-Generate [86, 87], and Facade-X [34].

*Constraint-driven mapping languages* leverage constraint languages to generate RDF graphs from heterogeneous (semi-)structured data. For example, the Shape Expressions Mapping Language (ShExML) [51] leverages ShEx [108].

### 4.2. Characteristics

We derived a set of characteristics from the papers on mapping languages to discuss each characteristic for each mapping language (Table 2). We divided these characteristics into three categories: (i) declarative transformation description, (ii) data access and retrieval, and (iii) RDF specification coverage.

***Declarative transformation description***

Mapping languages declaratively describe how schema and data transformations are applied on input data. The following characteristics are considered:

**S1: Schema transformation** Schema transformations (re-)model the input data, describe relations between data, and which vocabularies to use [63]. For example, the schema transformation describes how a person's age should be modelled by relating the person's age from the input data, to the person, and use the appropriate vocabulary such as `foaf:age`. If the schema transformation is declaratively described by the mapping language [37] to be re-usable for generating any kind of RDF knowledge graph.

**S2: Data transformation** Data transformations describe how to change data into a new representation [110]. For example: the person's birth date is available in the input data, but the schema transformation requires the person's age. The data transformation describes how to transform the birth date into the age which can be used then by the schema transformation, as shown in *S1: Schema transformation*. If data can be transformed or conditions can be applied with the schema transformation [37] to allow cleaning and processing the data. Some schema transformations leverages existing data transformations, these data transformations are discussed in Section 5.

**S3: Export description** Export descriptions describe where and how the generated RDF is exported. If the serialization format e.g. RDF/XML or Turtle, and target e.g., a file or a SPARQL endpoint of a generated RDF graph are described [129]. Describing where the generated RDF is exported to allows better integration with other tools during knowledge graph generation.

**S4: End-to-end** If a mapping language allows describing the entire process declaratively from accessing and transforming the data until exporting the generated RDF, or relies on hard-coded parts to access, transform, or export RDF graphs. This way, the complete generation process is described and re-usable without resorting to scripts customized for a single knowledge graph.

**S5: Web standards integration** If a mapping language integrates with existing web standards by relating to existing W3C recommendations. By relying on standards, implementations implementing them already can adopt a schema transformation more easily.

| Title | Author(s) | Scope |
|---|---|---|
| A Generic Mapping-based Query Translation from SPARQL to Various Target Database Query Languages | Michel F. et al. | ML, DT, VS |
| A middleware framework for scalable management of linked streams* | Le-Phuoc D. et al. | MS |
| A SPARQL Extension for Generating RDF from Heterogeneous Formats | Lefrançois M. et al. | ML, DT, MS |
| DAFO: An Ontological Database System with Faceted Queries | Pankowski T. et al. | VS |
| Declarative Data Transformations for Linked Data Generation: The Case of DBpedia | De Meester B. et al. | DT, VS |
| Detailed Provenance Capture of Data Processing | De Meester B. et al. | ML |
| D2RML: Integrating Heterogeneous Data and Web Services into Custom RDF Graphs | Chortaras A. et al. | ML, MS |
| D-REPR: A Language for Describing and Mapping Diversely-Structured Data Sources to RDF | Vu B. et al. | ML, MS, DT |
| Enabling Ontology-Based Access to Streaming Data Sources | Calbimonte J. et al. | VS |
| Enabling RDF Stream Processing for Sensor Data Management in the Environmental Domain | Llave A. et al. | VS |
| Executing SPARQL queries over Mapped Document Store with SparqlMap-M | Unbehauen J. et al. | VS |
| Exploiting Declarative Mapping Rules for Generating GraphQL Servers with Morph-GraphQL* | Chaves-Fraga D. et al. | VS |
| Flexible RDF Generation from RDF and Heterogeneous Data Sources with SPARQL-Generate | Lefrançois M. et al. | ML, DT, MS |
| Formalisation and Experiences of R2RML-Based SPARQL to SQL Query Translation Using Morph | Priyatna F. et al. | DT, MS |
| FunMap: Efficient Execution of Functional Mappings for Knowledge Graph Creation | Jozashoori, S. et al. | DT |
| FunUL: A Method to Incorporate Functions into Uplift Mapping Languages | Crotti Junior A. et al. | DT |
| GeoTriples: a Tool for Publishing Geospatial Data as RDF Graphs Using R2RML Mappings | Kyzirakos K. et al. | ML, MS |
| GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings | Kyzirakos K. et al. | ML, MS |
| Implementation-independent function reuse* | De Meester B. et al. | DT |
| KR2RML: An Alternative Interpretation of R2RML for Heterogeneous Sources | Slepicka J. et al. | MS |
| LDScript: A Linked Data Script Language* | Corby O. et al. | DT |
| Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation | Van Assche D. et al. | ML |
| Linked Data Integration Framework | Schultz A. et al. | MS |
| Machine-Interpretable Dataset and Service Descriptions for Heterogeneous Data Access and Retrieval | Dimou A. et al. | ML |
| MapSDI: A Scaled-Up Semantic Data Integration Framework for Knowledge Graph Creation | Jozashoori, S. et al. | MS |
| Mapping between RDF and XML with XSPARQL | Bischof S. et al. | ML, DT |
| Mapping Hierarchical Sources into RDF Using the RML Mapping Language | Dimou A. et al. | ML, MS |
| Obi-Wan: Ontology-Based RDF Integration of Heterogeneous Data | Buron M. et al. | VS |
| Ontario: Federated Query Processing against a Semantic Data Lake | M. Endris K. et al. | VS |
| Ontology–based access to temporal data with Ontop: A framework proposal | Güzel Kalayci E. et al. | VS |
| Ontology-Based Data Access: Ontop of Databases | Rodríguez-Muro M. et al. | VS |
| Ontop: Answering SPARQL Queries over Relational Databases | Calvanese D. et al. | VS |
| Ontop-spatial: Ontop of geospatial databases | Bereta K. et al. | VS |
| On the semantics of heterogeneous querying of relational, XML, and RDF data with XSPARQL | Lopes N. et al. | ML, DT |
| Optique: Towards OBDA Systems for Industry | Kharlamov E. et al. | VS |
| Optique: Zooming in on Big Data | Giese M. et al. | VS |
| Parallel RDF Generation from Heterogeneous Big Data | Haesendonck G. et al | MS |
| Querying the Web of Data with XSPARQL 1.1* | Dell'Aglio D. et al. | ML, DT |
| RDF-Gen: Generating RDF from Streaming and Archival Data | Santipantakis G. M. et al. | MS |
| RocketRML - A NodeJS implementation of a use-case specific RML mapper | Umutcan Ş. et al. | MS |
| RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data | Dimou A. et al. | ML, MS |
| R2RML-F: Towards Sharing and Executing Domain Logic in R2RML Mappings | Debruyne C. et al. | DT |
| SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs | Iglesias E. et al. | MS |
| ShExML: improving the usability of heterogeneous data mapping languages for first-time users | García-González H. et al. | ML |
| Sustainable Linked Data Generation: The Case of DBpedia | Maroy W. et al. | DT |
| Squerall: Virtual Ontology-Based Access to Heterogeneous and Large Data Sources | Mami M. et al. | VS |
| The Virtual Knowledge Graph System Ontop | Guohui X. et al. | ML, VS |
| Toward the Web of Functions: Interoperable Higher-Order Functions in SPARQL | Atzori M. et al. | DT |
| Translation of Relational and Non-relational Databases into RDF with xR2RML | Michel F. et al. | ML, MS, VS |
| TripleWave: Spreading RDF Streams on the Web | Mauri A. et al. | MS |
| Turning Transport Data to Comply with EU Standards While Enabling a Multimodal Transport Knowledge Graph* | Scrocca M. et al. | MS |
| XGSN: An Open-source Semantic Sensing Middleware for the Web of Things | Calbimonte J. et al. | VS |

**Table 1**
Overview of selected articles for this survey paper in alphabetical order. Articles with '*' were manually added in step 3. The scope of each article is indicated: Mapping Language (ML), Data Transformation (DT), Materialization System (MS), or Virtualization System (VS).

| Mapping Language | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 | S11 | S12 | S13 | S14 | S15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| RML | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | | ✓ | ✓ | ✓ | ✓ | ✓ | |
| xR2RML | ✓ | | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| D2RML | ✓ | | ✓ | | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | ✓ | |
| D-REPR | ✓ | ✓ | | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓* | ✓ | | |
| XSPARQL | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SPARQL-Generate | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| ShExML | ✓ | | ✓ | | ✓ | ✓ | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | |
| Facade-X | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

**Table 2**
Schema transformation characteristics applied on the discussed mapping languages.
*D-REPR does not support RDF Literal language tags, only support datatypes.
S1: Schema transformation, S2: Data transformation, S3: Export description, S4: End-to-end, S5: Web standards integration, S6: Joins, S7: Intermediate representation, S8: Nested hierarchies, S9: Multi-paths, S10: RDF triples, S11: IRIs, S12: Literals, S13: Blank Nodes, S14: Named Graphs, S15: Collections and containers

***Data processing and composition***

Mapping languages provide access descriptions and descriptions regarding how to process input data in various ways. We consider the following characteristics:

**S6: Joins** If data joins are supported and declaratively described in the schema transformation [37]. Joins combine data from heterogeneous (semi-)structured data into RDF.

**S7: Intermediate representation** If a mapping language uses heterogeneous (semi-)structured data directly or transforms it first into a homogeneous intermediate representation before applying the schema and data transformations [37]. An intermediate representation influences the implementation of a schema transformation because all data must be transformed first into the intermediate representation and afterwards into RDF.

**S8: Nested hierarchies** Nested data structures such as XML or JSON are not tabular like SQL databases or CSV, but hierarchical. If a mapping language can handle hierarchical structures and join these nested data structures or not [37] to support both hierarchical and tabular data.

**S9: Multi-paths** If multiple path expressions – such as JSONPath or XPath – can be used together for accessing nested data encoded in heterogeneous formats [98], e.g., NoSQL databases allow mixing formats, such as JSON inside the NoSQL database. If a mapping language can combine different path expressions or not to refer to nested data in data sources, such as JSON in a CSV column.

***Coverage of the RDF(S) specification***

We cover characteristics from the RDF 1.1 specification and the RDF Schema (RDFS) specification. These characteristics are typically described by each mapping language. Most of these characteristics are supported by each mapping language, forming the base of generating a knowledge graph in RDF.

**S10: RDF triples (subjects, predicates, and objects)** If a mapping language can describe how RDF subjects, predicates and objects should be generated [37].

**S11: IRIs** If a mapping language can describe how a valid IRI conform RFC 3987 should be generated.

**S12: Literals (including language tags and datatypes)** If a mapping language can specify how a Literal with optionally a language tag or datatype should be generated.

**S13: Blank Nodes** If a mapping language can describe the generation of RDF blank nodes [37].

**S14: Named graphs** If a mapping language can describe the generation of an RDF Named Graph to make statements about a set of RDF triples [37].

**S15: Collections and containers** If a mapping language can describe the generation of RDFS collections and containers [98].

## 4.3. Dedicated mapping languages

Dedicated mapping languages are based upon dedicated mapping language specifications such as R2RML [35] or a custom syntax such as YAML or JSON, for transforming heterogeneous (semi-)structured data sources into RDF.

***RDF Mapping Language (2013)***

RDF Mapping Language[2] (RML) [46] describes how RDF is generated from heterogeneous (semi-)structured data by extending the W3C recommendation R2RML [35] (S5) to heterogeneous (semi-)structured data sources. RML introduces a RML Logical Source which describes how the input data should be accessed and referred to using reference formulations, such as JSONPath for JSON or XPath for XML data. Currently, RML supports heterogeneous (semi-)structured data sources, such as relational databases, CSV/TSV, JSON, XML, web APIs, and streams [129, 47].

RML is backwards compatible with R2RML (S1, S4) and leverages R2RML's join support (S6), IRI description (S11), subjects, predicates and objects (S10), named graphs (S14), Literals with language tags, data types (S12), and blank nodes (S13). The input data is used directly, without transforming it into an intermediate format (S7). RML is aligned with FnO [39] (S2, S4) to describe data transformations on the input data (Section 5). Recently, RML was extended with Logical Target to describe how and where the generated RDF must be exported to (S3, S4) [129]. RML cannot handle nested data structures (S8), data structures consisting of multiple data formats (S9), or collections and

[2] https://rml.io/specs/rml/, last accessed 10/11/2021

containers (S15), but an approach [42] is proposed to overcome these limitations.

***xR2RML (2015)***

xR2RML[3] [98] extends R2RML [35] (S1, S5, S10, S11, S12, S13, S14) and RML [46] (S7) with additional data sources, e.g. NoSQL databases, nested data structures, such as XML or JSON (S8) with multiple data formats, and RDFS collections and containers generation (S15). xR2RML allows to combine multiple reference formulations when referring to a data value (S9) which is common among NoSQL databases where a key-value store may contain XML and JSON data as value. xR2RML describes how multiple data sources should be joined (S6) and supports nested data structures (S8) through xR2RML's Nested Term Maps which allow describing RDFS collections and containers (S15). xR2RML describes the RDF Literal's language tag or data type (S12). xR2RML allows to push down data values from a nested data structure to make it accessible in lower levels of the nested data structure. However, xR2RML does not describe where the generated RDF triples must be exported to (S3, S4), and does not describe data transformations (S2, S4).

***D2RML (2018)***

D2RML [30, 29] extends R2RML [35] with a D2RML Logical Source to define how the data source should be accessed and additionally support for transformations, conditions and custom IRI generation functions (S2). D2RML abstracts the data model to use a column and row approach instead of using the input data directly (S7). D2RML supports HTTP web APIs, CSVs, JSON, and XML data. D2RML can describe the generation of subjects, predicates, objects (S10), graphs (S14), blank nodes (S13), Literals with language tags and data types (S12), IRIs (S11), and join multiple data sources during the schema transformation (S6), because it extends R2RML (S1). However, D2RML cannot handle nested data structures (S8), heterogeneous (semi-)structured data (S9), describe the generation of collections and containers (S15), or describing the output location (S3, S4).

***Dataset Representation (2019)***

Dataset Representation (D-REPR) [132] is a mapping language based upon a custom YAML [11] syntax (S5) for transforming heterogeneous (semi-)structured data into RDF. D-REPR supports joins (S6) among spreadsheets, CSV, XML, JSON, (non-)relational databases, and NetCDF files (S8). D-REPR represents all data in a JSON tree based structure instead of the Nested Relational Model (NRM) model used in KR2RML (S7). D-REPR does not support multi-path expressions when referring to data (S9). D-REPR fully describes the schema transformation (S1, S10, S11, S12, S13, S14) and supports data transformations on the input data (S2). However, the output description, named graphs, RDFS collections and containers, and RDF Literal language tags (S15) are not specified in the D-REPR mapping rules (S3, S4).

[3] https://www.i3s.unice.fr/~fmichel/xr2rml_specification_v5.html, last accessed 10/11/2021

## 4.4. Repurposed mapping languages

Repurposed mapping languages re-use syntax from existing specifications such as W3C's SPARQL [109] or W3C's ShEx [108], for transforming heterogeneous (semi-)structured data sources into RDF.

***XSPARQL (2009)***

XSPARQL[4] [15, 41] is a mapping language that combines XQuery [92], JSON-LD [124] and R2RML [35] with the W3C-recommended SPARQL query language [109] (S5) for transforming data in XML, JSON, and relational databases into RDF (S1). XSPARQL not only transforms XML and relational databases into RDF (uplifting), but also the other way around (lowering). XSPARQL inherits all the features from SPARQL (S2, S10, S11, S12, S13, S14). XSPARQL supports joining of nested hierarchy data sources by building on top of SPARQL without an intermediate format (S6, S7, S8). XSPARQL does not support multi-path expressions (S9), or specifies to where the generated triples are exported to (S3, S4). Built-in and custom data transformations are applied using SPARQL Functions (S2). XSPARQL supports relational databases and XML, RDF, or JSON files.

***SPARQL-Generate (2017)***

SPARQL-Generate [86, 87] extends the W3C-recommended SPARQL query language (S5) to transform heterogeneous (semi-)structured data into RDF. SPARQL-Generate supports streaming and binary data sources such as HDT[5], WebSockets[6], and Kafka[7], on top of the data sources supported by RML: relational databases, XML, JSON, and CSV/TSV data.

SPARQL-Generate can be combined with SPARQL Template Transformation Language [31] which can transform RDF graphs into structured text. SPARQL-Generate uses the input data directly during its schema transformation (S7) and can leverage SPARQL functions to execute data transformations when applying its schema transformation (S2). SPARQL-Generate relies on SPARQL to describe the schema transformation (S1), to join heterogeneous (semi-)structured data (S6), handle nested hierarchical data (S8), generation of subjects, predicates, objects (S10), IRIs (S11), named graphs (S14), Literals with language tags or data types (S12), and blank nodes (S13), collections and containers (S15). However, SPARQL-Generate does not specify how the RDF must be exported (S3, S4) or how to handle multi-paths (S9).

***Shape Expressions Mapping Language (2020)***

Shape Expressions Mapping Language[8] (ShExML) is a mapping language based on W3C's Shape Expressions (ShEx) [108] (S5) to transform heterogeneous (semi-)structured data into RDF. ShExML supports relational databases, CSV, XML and JSON files. ShExML describes joins between data sources

[4] https://www.w3.org/Submission/xsparql-language-specification/, last accessed 10/11/2021

[5] https://www.rdfhdt.org/, last accessed 10/11/2021

[6] https://html.spec.whatwg.org/multipage/web-sockets.html#the-websocket-interface, last accessed 10/11/2021

[7] https://kafka.apache.org/, last accessed 10/11/2021

[8] http://shexml.herminiogarcia.com/spec/, last accessed 10/11/2021

(S6). It does not use an intermediate representation of the input data (S7) and can handle nested data structures through its nested iterators (S8).

ShExML describes how subjects, predicates and objects (S10), graphs (S14), RDF's Literal language tags or data types (S12). ShExML supports IRIs (S11) and blank nodes (S13). ShExML describes the schema transformation (S1), but it cannot describe collections or containers (S4, S15). ShExML does not describe to where the generated RDF is exported to (S3, S4), or how to handle multi-paths (S9). ShExML does not describe data transformations to apply on the heterogeneous (semi-)structured data (S2, S4).

***Facade-X (2021)***

Facade-X [34] overrides SPARQL's `SERVICE` operator (S5) to generate RDF from heterogeneous (semi-)structured data sources. It supports JSON, CSV, HTML, XML, spreadsheets, binary data and embedded data such as EXIF in images. Facade-X uses the input data directly as it follows the approach of W3C's Direct Mapping recommendation [3] (S7). As SPARQL-Generate, it relies on SPARQL for joins (S6) and functions (S2), nested data structures (S8), RDF generation (S10, S11, S12, S13, S14), and describes the schema transformation (S1). However, Facade-X does not specify how the RDF should be exported (S3, S4), or how to handle multi-paths (S9).

### 4.5. Remarks

In this subsection, we highlight mapping languages which were excluded from this systematic literature review because they do not support heterogeneous (semi-)structured data, but they are worth mentioning since they influenced existing mapping languages and, thus, the research domain overall.

$R_2O$ [9] and D2RQ [33] are both mapping languages for generating RDF graphs but only from relational databases. They are worth to mention because they influenced the works of the RDB2RDF Working Group[9] when the R2RML [35] and Direct Mapping [3] specifications became W3C recommendations. A lot of the mapping languages discussed in this paper (RML, xR2RML, and D2RML) extend R2RML.

Ontop Mapping Language [112] is an alternative mapping language to R2RML. Ontop Mapping Language can describe how to generate RDF from relational databases. This mapping language is compatible with R2RML and mapping rules can be converted in both directions. The Ontop knowledge graph generation system does support RDF generation from heterogeneous (semi-)structured data but without extending its mapping language. On the contrary, Ontop relies on data virtualization frameworks, such as Denodo[10], Dremio[11], and Teiid[12] to access heterogeneous (semi-) structured data as if they reside in a relational database.

X3ML mapping definition language [100, 94] is an alternative mapping language written in XML. It is used by the X3ML engine to transform XML data into RDF. Other data formats are under developement.

When DBpedia was originally introduced, it had its own mapping language[13] based on the MediaWiki syntax to transform Wikipedia pages into the DBpedia RDF graph. However, not all schema and data transformations were defined using its mapping language but a few of these transformations were embedded in the DBpedia extraction framework [95]. While DBpedia is not generated from heterogeneous data and its mapping language does not refer to heterogeneous data, we thought that it is worth mentioning due to the impact of DBpedia's RDF graph to the broader Semantic Web community.

YARRRML [60] is a human friendly representation of RML based on YAML[11] to support users to define mappings. YARRRML is widely adopted in the RML community, but it was excluded because it is published as a demo.

## 5. Data transformations

In this Section, we discuss data transformation descriptions (Table 1) and how they are aligned with mapping languages. Besides schema transformations (Section 4), data transformations – which describe how to change data values into a new representation [110] – may also be needed for generating an RDF knowledge graph from heterogeneous (semi-)structured data. To date, no overview of existing data transformations is available. In this Section we evaluate a set of characteristics on existing data transformation approaches.

Section 5.1 discusses the characteristics we applied, and Section 5.2 discusses each data transformation. Afterwards, we mentions data transformations approaches which are excluded, but are worth mentioning in Section 5.3.

### 5.1. Characteristics

We derived a set of characteristics (Table 3) and divided them in 2 categories: *description* and *alignment*.

***Description***

The way in which a data transformation is described such as function's body or its parameters.

**F1: Declarativeness** If the data transformations are declaratively described, considering the function body, parameter values, and return values. Declarative descriptions are not tight to a single knowledge graph generation pipeline but can be reused for generating other knowledge graphs.

**F2: Shareability** If data transformations can be shared and reused through a common repository or library. Reusing data transformations through a common repository enables systems to avoid re-implementing each data transformation.

**F3: Custom data transformations** If custom data transformations can be described declaratively and be used with existing data transformations since generating a knowledge graph may require a custom transformation not built-in into the system.

[9] https://www.w3.org/2001/sw/rdb2rdf/, last accessed 23/03/2022

[10] https://www.denodo.com/, last accessed 07/04/2022

[11] https://www.dremio.com/, last accessed 07/04/2022

[12] https://teiid.io/, last accessed 07/04/2022

[13] https://mappings.dbpedia.org, last accessed 23/03/2022

***Alignment***

How the data transformations are aligned with mapping languages and if they can be used standalone or not.

**F4: Independence** If the data transformation description is independent, it can be used for data cleaning and processing without a mapping language.

**F5: Integration with mapping languages** How mapping languages integrate data transformations, interact with them, and describe them in the mapping rules. Data transformations may only be applicable before, during, or after the schema transformation.

## 5.2. Data transformations

In this Section, we analyzed 7 data transformation approaches on 5 different characteristics to provide an overview of these approaches.

***SPARQL Functions (2013)***

The W3C-recommended SPARQL [109] supports functions[14] which are executed during query evaluation by a SPARQL engine. SPARQL specifies a set of built-in functions, conditions, and custom functions (F3). However, supported functions heavily depend on the underlying SPARQL engine executing the query (F4). SPARQL functions can be reused among different engines through SPARQL Federated Queries [7] (F2). Function parameters and return values are not hardcoded but binded using SPARQL `BIND` expressions (F1). SPARQL also provides conditions through `FILTER` and `IF` expressions to evaluate parts of the SPARQL query only when a certain condition is met. However, SPARQL functions cannot be used standalone because of their tight integration with the SPARQL query and engine (F4). SPARQL functions are integrated in SPARQL queries as a SPARQL operator and can be applied on any part of the schema transformation (F5). SPARQL functions are leveraged by SPARQL-Generate[15] [86], XSPARQL[16] [15], and Facade-X [34] to provide data transformations.

***GeoTriples (2014)***

GeoTriples[17] [83, 81] extends RML [46] with support for geographical data sources and transformations. GeoTriples uses GeoSPARQL[18] [10] and stSPARQL [82, 12] functions which are aligned with RML mapping rules through two extensions: `rrx:Function` and `rrx:ArgumentMap`. The parameters order in `rrx:ArgumentMap` matches the order of arguments of the function (`rdf:List`). Each function parameter is an `rr:TermMap` which allows GeoTriples to reference values as function parameters. The return value is used directly in the mapping rules and are not declarative described (F1, F5). GeoTriples supports a fixed set of GeoSPARQL and stSPARQL functions (F3) as data transformations, but these functions do not depend on GeoTriples. Thus, GeoSPARQL and stSPARQL functions can be used with other GeoSPARQL and stSPARQL engines (F4). Currently, GeoSPARQL and stSPARQL functions are built-in into the GeoSPARQL query language [105], and not shared through a repository as FnO (F2).

***KR2RML (2015)***

KR2RML [122] is based on R2RML [35] with support for data transformations through custom functions and conditions written in Python (F3). The function's body and parameter values are integrated in KR2RML mapping rules as a string (F5). Functions can use built-in Python modules or from Python Package Index[19] (F2). Function parameters and return values are hardcoded in the KR2RML mapping rules without a declarative description (F1, F4). KR2RML functions are implemented in Karma[20].

***Function Ontology (2016)***

The Function Ontology[21] (FnO) [38, 39, 95] is a semantic description of functions without depending on their implementation (F4). FnO describes each function's parameters and return value to specify how the function should be used. Moreover, FnO also describes the problem the function solves to enable semantically reuse of functions. Existing functions and their implementations in various languages can be shared throughout the Function Hub [39] (F2). Values which are passed to a function are not hardcoded but referenced (F1). Custom functions can be added by providing an FnO description and optionally one or multiple implementations (F3). FnO can be used standalone (F4), but is aligned with RML to apply data transformations.

In RML, FnO descriptions can be added in the mapping rules on any part of the schema transformation through `fnml:FunctionMap`[22] since `fnml:FunctionMap` is compatible with an R2RML Term Map. Each Function Map refers to the function description to execute the function (`fno:executes`) and its parameters (example: `grel:inputString`) (F5). An FnO function can be used as a data transformation or as a condition in RML when performing the schema transformation. Several materialization implementations for generating knowledge graphs from heterogeneous (semi-)structured data using mapping languages incorporated support for FnO functions [64, 46, 70, 120].

***FunUL (2016)***

FunUL [73] extends RML [46] by incorporating functions and conditions inside the mapping rules. FunUL function's body is a Literal, specified with the `rrf:functionBody` property. Its function's name is described with the property `rrf:functionName`. Each function can be called through a `rrf:functionCall` which references to a function (`rrf:function`) and its parameters (`rrf:parameterBindings`) (F5).

[14] https://www.w3.org/TR/sparql11-query/#func-rdfTerms, last accessed 10/11/2021

[15] https://github.com/sparql-generate/sparql-generate, last accessed 10/11/2021

[16] https://github.com/semantalytics/xsparql, last accessed 10/11/2021

[17] https://github.com/LinkedEOData/GeoTriples, last accessed 10/11/2021

[18] https://www.opengis.net/doc/IS/geosparql/1.0, last accessed 10/11/2021

[19] https://pypi.org, last accessed 10/11/2021

[20] https://github.com/usc-isi-i2/Web-Karma, last accessed 10/11/2021

[21] https://fno.io/spec/, last accessed 10/11/2021

[22] http://semweb.mmlab.be/ns/fnml#, last accessed 10/11/2021

| Data Transformation | F1 | F2 | F3 | F4 | F5 |
|---|---|---|---|---|---|
| SPARQL Functions | declarative described, referenced function & parameters | Federated Queries | Yes | Depend on SPARQL | Any |
| GeoTriples | declarative described, referenced function & parameters | No | No | Standalone | R2RML or RML |
| KR2RML | function body and parameters as hardcoded strings | Python Package Index | Yes | Depend on KR2RML | KR2RML-only |
| Function Ontology | declarative described, referenced function & parameters | Function Hub | Yes | Standalone | Any |
| FunUL | function body as string, referenced function & parameters | Implementation dependent | Yes | Depend on RML | RML-only |
| D2RML | declarative described, referenced function & parameters | No | Partially | Depend on D2RML | D2RML-only, only pre-processing input data |
| D-REPR | function body and parameters as string, hardcoded parameters | Python Package Index | Yes | Depend on D-REPR | D-REPR-only, only pre-processing input data |

**Table 3**
Data transformation characteristics for mapping languages.
F1:Declarativeness, F2:Execution, F3:Shareable, F4:Independence, F5:Integration

FunUL is based on R2RML-F [40] and broadens R2RML-F's scope from relational databases to heterogeneous (semi-)structured data. FunUL is not stand-alone, as it must be used together with RML mapping rules (F4). FunUL's functions are written in JavaScript as a Literal in the RML mapping rules, but FunUL's approach does not depend on a specific programming language as with KR2RML's data transformations. FunUL's functions parameters and return values are not hardcoded in the mapping rules for reusability (F1). However, FunUL reuses `rr:column`, `rml:reference`, and `rr:constant` from RML [46] and R2RML [35] for referencing values for function parameters, therefore no other mapping language can be used. FunUL's function return values are directly used to generate RDF in a R2RML Predicate Object Map by the system executing the RML mapping rules. Built-in functions of a programming language can be used and custom functions can be defined as well inside the mapping rules (F3). Depending on the programming language and system, functions can be shared and reused from repositories, but there is no equivalent of FnO's Function Hub (F2). FunUL is demonstrated in a forked version of the RMLProcessor[23].

### *D2RML (2018)*

D2RML [30] extends R2RML [35] for mapping heterogeneous data into RDF (Section 4), but also with `dr:Function` to support data transformations and conditions. D2RML incorporates data transformations through `dr:Transformations` in its declarative description for generating RDF and are applied on the heterogeneous (semi-)structured data retrieved through D2RML's declarative data source description (Logical Source). D2RML's data transformations can only be described in a D2RML's Triples Map to apply on the retrieved data, they cannot be used on D2RML's abstracted intermediate format or on the generated RDF. D2RML conditions are located in R2RML's Term Maps to generate an RDF triple based upon a certain condition (F5). D2RML uses `dr:Function` to refer to a function and `dr:ParameterBinding` to refer to function's parameters, but the return value is not described (F1). D2RML data transformations and conditions cannot be used standalone because of the tight integration with D2RML (F4). D2RML provides a set of custom IRI generation functions and allows the use of web services to apply custom transformations (F3). D2RML can not leverage a function repository to share and reuse existing functions (F2). D2RML data transformations and conditions are available as a web service[24].

### *D-REPR (2019)*

D-REPR integrates data transformations for pre-processing data. Functions can be customized (F3), and are shareable as with KR2RML through the Python Package Index (F2). Functions are written in Python with hardcoded parameters of each function (F1, F4), following a similar principle as in the case of KR2RML. These functions are only declaratively described for input data in a `[preprocessing]` YAML block which contains a list of functions. Each function has a function type (`[type]`), its input parameters (`[input]`), expected return values (`[output]`) and its body (`[code]`). Since these functions are described in the `[preprocessing]`) block, they cannot be applied on the intermediate JSON tree format or

[23] `https://github.com/CNGL-repo/RMLProcessor`, last accessed 10/11/2021

[24] `https://apps.islab.ntua.gr/d2rml/`, last accessed 10/11/2021

on the generated RDF (F5). D-REPR's data transformations are implemented in D-REPR[25].

### 5.3. Remarks

D2RQ Mapping Language [33] is a mapping language which highly influenced R2RML [35] (Section 4) and supports conditions similar to FunUL, SPARQL Functions, or D2RML. However, D2RQ Mapping Language is not included in this systematic literature review as it is not a mapping language that supports heterogeneous data sources. It is worth mentioning though that while D2RQ Mapping Language influenced R2RML and while D2RQ Mapping Language does support conditions and not other data transformations, these conditions were not integrated in R2RML. While this is not a problem for R2RML, as such conditions may be addressed with a view, this is not the case for the mapping languages that extend R2RML for heterogeneous data. Currently, this might be one of the major drawbacks of these mapping languages and the W3C Community Group on Knowledge Graph Construction[26] considers bringing it back in its specification's revision as shortcut for conditional data transformations.

## 6. Systems

We discuss systems for generating RDF graphs from heterogeneous (semi-)structured data using mapping languages. We divided the systems into two categories, based on how they generate the RDF graphs:

**Materialization systems** execute the mapping rules and materialize the RDF graphs, like an Extract-Transform-Load (ETL) process [35].

**Virtualization systems** answer a query by virtualizing RDF graphs through mapping rule execution (often referred to as Ontology Based Data Access (OBDA)) [35].

Several systems exist for the same schema e.g. RML (section 4.3) or data transformation e.g. FnO (Section 5.2). More recent systems support more features e.g. RML's Logical Target [128] than older systems. Multiple systems for the same schema or data transformation improves the optimizations for executing these transformations [64, 54].

Section 6.1 discusses the characteristics we evaluate for each system, Section 6.2 describes the materialization systems and Section 6.3 the virtualization systems evaluated in this paper. Section 6.4 mentions systems which are worth mentioning but could not be included in this systematic literature review.

### 6.1. Characteristics

We compiled a list of common characteristics which differ between systems:

#### *Input/Output*

These characteristics are related to the input and output data and which schema and data transformations are used by each system:

**T1: Input Data** Which data sources are accessible and where data can be retrieved from.

**T2: Input Data formats** Which data formats are supported for the input data.

**T3: Output Data** How the generated knowledge graphs are exported to a specific location in a certain format.

**T4: Output Data formats** Which data formats are supported for the output data.

**T5: Schema transformation language** Which mapping language(s) are supported by a system.

#### *Data transformation*

Each system may have support for data transformations besides schema transformations. The following characteristics are used to validate how and which data transformations are supported by a given system:

**T6: Data transformation language** Which data transformation language(s) are supported by a system.

**T7: Applicability** Where during the execution process does the system apply data transformations to the data: (i) preprocessing, during the input data retrieval, (ii) during schema transformation into RDF, (iii) post-processing, after the schema transformation.

#### *Implementation*

Each system has specific characteristics regarding the implementation itself such as the programming language or how it integrates with other systems.

**T8: Programming language** The programming language of the system.

**T9: Integration** How an implementation can be integrated with other systems.

**T10: Interface** How a system can be used.

**T11: License** The license of a system.

**T12: Repository** The location of the system's code repository, if available.

Characteristics specific to materialization and virtualization systems are described in Sections 6.2 and 6.3

### 6.2. Materialization implementations

In this Section, we discuss materialization systems for transforming heterogeneous (semi-)structured data (Table 1) into knowledge graphs as RDF with mapping languages. We applied a similar categorization to materialization systems as in Section 4 *Schema transformations* and list them in chronological order.

We studied each materialization system and created a list of characteristics specific for materialization which greatly differ between these systems. We discuss the following characteristics for each materialization system (Tables 7, 8):

**T13a: Optimizations** Which optimizations are applied to the materialization process?

**T13b: Scaling** Which approaches are used to scale the materialization process?

[25] https://github.com/usc-isi-i2/d-repr, last accessed 10/11/2021

[26] https://www.w3.org/community/kg-construct/, last accessed 07/04/2022

***Materialization systems for dedicated mapping languages***

*Linked Stream Middleware (2012)* Linked Stream Middleware (LSM) [85] focuses on sensor data transformation from relational databases and other data sources in a streaming fashion (T13b). LSM leverages R2RML [35] and D2R [16] mapping languages (T5) to access relational databases while a wrapper is required to generate RDF from other data sources (T1, T2). Data sources can either push messages onto LSM's message queue for processing or LSM pulls data from data sources through an Apache Hadoop based cluster for generating RDF (T13a). RDF graphs can be stored in a triple store or queried through LSM's query interface with SPARQL [109] or CQELS [84] continuously or in a pulling fashion (T3, T4). LSM does not support data transformations, it only generates RDF from heterogeneous (semi-)structured data sources (T6, T7). LSM was publicly deployed, but this deployment is not available anymore. LSM's Java source code and license are not publicly available, therefore we cannot study all characteristics (T8, T9, T10, T11, T12).

*Morph-RDB v3.12.5 (2014)* Morph-RDB [107], previously known as ODEMapster, implements the R2RML mapping language [35] (T5) to generate RDF graphs from SQL query results of relational databases and CSV files (T1, T2). Morph-RDB applies optimization techniques, such as self-join and subquery elimination when querying the relational databases for generating RDF (T13a, T13b). The RDF graphs are exported to a file, either in N-Triples, Turtle, N3, or RDF/XML, specified in Morph-RDB's configuration file (T3, T4). Morph-RDB can also translate SPARQL queries into SQL queries (Section 6.3). Morph-RDB is written in Scala (T8) and does not support data transformations (T6, T7). Morph-RDB is available as a CLI implementation and provides a Docker image (T9, T10). Morph-RDB is released on GitHub[27] (T12) under *Apache License 2.0* (T11).

*RMLMapper v4.12.0 (2014)* RMLMapper [46] is a Java implementation of an RML processor with support for FnO functions and RML's Logical Target. RMLMapper can also generate provenance metadata during the schema and data transformation if enabled. RMLProcessor, RMLMapper's predecessor, was forked and extended with FunUL functions to provide data transformations [73]. The RMLMapper supports RML for schema transformations (T5), FnO for data transformations (T6), and generation of metadata during execution. It first retrieves all data, applies joins if needed, executes FnO functions and generates RDF (T13a, T13b). Thus, FnO functions are executed as part of the schema transformation (T7). The RDF graphs are exported to the specified RML's Logical Targets (T3). Currently, the RMLMapper supports N-Triples, N-Quads, Turtle, N3, RDF/XML, JSON-LD, HDT, Trix and TriG as RDF output formats (T4). It supports relational databases, SPARQL endpoints, files, and data on the Web (T1). The RMLMapper generates RDF graphs from SQL query results, W3C Web of Things [74] web APIs, SPARQL [109] query results, XLSX, ODS, CSV, TSV, JSON and XML files (T2). It exports the RDF graphs in N-Quads, Turtle, TriG, TriX, JSON-LD and HDT to files, VoID datasets, and with `SPARQL UPDATE` queries to SPARQL endpoints. RMLMapper is available as a CLI implementation, a Docker image, and a Java library (T9, T10). The RMLMapper is released under *MIT* license (T11) on GitHub[28] (T12).

*Karma v2.5 (2015)* Karma [122] implements KR2RML [122] (T5) for transforming heterogeneous data sources into RDF graphs through an intermediate representation (Nested Relational Model [91]). Karma transforms any heterogeneous (semi-)structured data into NRM format and applies joins if needed. Afterwards, the transformation to RDF with KR2RML mapping rules is applied (T13a, T13b). Karma supports KR2RML's data transformations written in Python (T6) which are excuted during the schema transformation (T7). Karma is written in Java (T8) and is integrated in Karma's CLI implementation (T9, T10). Karma can access relational databases and files (T1) and transform SQL tables, CSV, TSV, JSON and XML files to RDF (T2). The RDF is exported to a file specified as a CLI parameter (T3). Karma supports JSON-LD and N3 as RDF output formats (T4). Karma is publicly available[29] (T12) under *Apache License 2.0* (T11).

*Morph-xR2RML v1.3.1 (2015)* Morph-xR2RML [98] extends Morph-RDB (T13a, T13b) with support for xR2RML [98] mapping language (T5). Morph-xR2RML is written in Scala (T8) and supports relational and NoSQL databases, and files (T1). Morph-xR2RML can transform SQL query results, NoSQL query results, JSON, XML, CSV, and TSV files to RDF (T2). Morph-xR2RML retrieves the data, applies joins if needed and generates RDF, including RDFS collections and containers (T13b). Morph-xR2RML does not provide data transformations, such as FnO or FunUL (T6, T7). Morph-xR2RML is configured with a configuration file and provides a CLI implementation to export the RDF graphs to a file or a SPARQL endpoint (T3, T6, T9, T10). Morph-xR2RML supports N-Triples, Turtle, N3, RDF/XML, and JSON-LD as RDF output format. Morph-xR2RML is available on GitHub[30] (T12) under *Apache License 2.0* (T11).

*TripleWave v2.1.1 (2016)* TripleWave [96] generates RDF graphs in a streaming fashion (T13b) using R2RML mapping rules [35] (T5). TripleWave consists of wrappers to access data sources and currently supports JSON files (T1, T2). R2RML mapping rules are only used to generate RDF from the retrieved data. TripleWave scales vertically by the number of CPU cores (T13a). TripleWave does not support data transformation, but they can be hardcoded into the data access wrappers (T6). Because of that, any data transfor-

[27] `https://github.com/oeg-upm/morph-rdb`, last accessed 23/03/2022, date last commit on default branch 08/05/2021

[28] `https://github.com/RMLio/rmlmapper-java`, last accessed 23/03/2022, date last commit on default branch 11/03/2022

[29] `https://github.com/usc-isi-i2/Web-Karma`, last accessed 23/03/2022, date last commit on default branch 27/02/2022

[30] `https://github.com/frmichel/morph-xr2rml`, last accessed 23/03/2022, date last commit on default branch 13/01/2022

mations is executed as a pre-processing step when accessing the input data (T7). It is configured with a configuration file to export its RDF stream in JSON-LD format (T4) to a file or publish it as a WebSocket or MQTT stream (T3). TripleWave is written in NodeJS (T8) and it is available under *Apache License 2.0* (T11) on GitHub[31] (T12) as a CLI implementation (T9, T10).

*GeoTriples v1.2.1 (2018)* GeoTriples [83, 81] implements GeoTriples's GeoSPARQL extensions to RML [46] (T5). It is written in Java (T8) available as a CLI implementation and webapp (T9, T10). GeoTriples can access relational databases and files and generates RDF from SQL query results, CSV, XML, JSON, GeoJSON, KML, and shapefiles (T1, T2). GeoTriples implements geospatial transformations from GeoSPARQL [105] and stSPARQL [12, 82] to handle geospatial data (T6). These transformations are executed together with the schema transformation (T7). It first fetches all geospatial data, applies transformations and joins if needed and generates RDF graphs to a file (T13a, T13b), RML's Logical Target is not supported yet (T3), but it can export RDF in Turtle or RDF/XML format (T4). GeoTriples is available as a CLI implementation or webapp (T9, T10) and it is released[32] (T12) under *Apache License 2.0* (T11).

*D2RML processor (2018)* D2RML processor [29] implements D2RML [29] mapping language (T5) with support for data transformations (T6), conditions and RDF generation from heterogeneous (semi-)structured data. D2RML processor's source code is not publicly available, but can be used as a web service (T9, T10). Since the source code is not publicly available (T12), we cannot study or discuss characteristics T1, T2, T3, T4, T8, T13a, or T13b.

*RDF-Gen (2018)* RDF-Gen [114] generates RDF in a streaming fashion from multiple sources described by a custom syntax (T5). RDF-Gen can access relational databases and files in CSV, XML, and JSON format (T1, T2). RDF-Gen first transforms data of a data source into records, these records are later on transformed into RDF using a centralized cluster (T13a, T13b). The authors claim that RDF-Gen support functions for data transformation, but we have not been able to verify this as the source is not publicly available (T6, T7). RDF-Gen exports RDF in N-Triples or Turtle to a file specified in its configuration file (T3, T4). RDF-Gen is available as compiled Java Jar CLI implementation (T9, T10, T12)[33] under *Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License* (T11).

*RMLStreamer v2.1.1 (2019)* The RMLStreamer [54] is RMLMapper's counterpart (Section 6.2) for generating RDF in a streaming fashion (T13b) using RML mapping rules [46] (T5). It can access files in CSV, JSON, or XML format, and Kafka, MQTT, or TCP streams (T1, T2). The RMLStreamer uses Apache Flink to scale horizontally and vertically depending on the number of streams and input rate when generating RDF. Moreover, the RMLStreamer also leverages Apache Flink for optimizing memory usage, high-availability, and fault-tolerance (T13b). It leverages FnO [38, 39] to apply data transformations during the schema transformation (T6, T7), and RML's Logical Target [128] (T3). The RMLStreamer can export RDF graphs in JSON-LD, N-Triples, or N-Quads format (T4). It is written in Scala (T8) and available[34] (T12) under *MIT* license (T11) as a CLI implementation and Docker image (T9, T10).

*MapSDI v1.0 (2019)* MapSDI [72] pre-processes heterogeneous data to remove duplicates and unnecessary data by exploiting RML [46] mapping rules (T5), and applies relational algebra to improve the execution of the mapping rules. MapSDI leverages existing RML processors to execute the mapping rules, therefore it inherits their characteristics (T3, T4, T5, T6, T7, T13a, T13b). MapSDI can pre-process files in CSV format (T1, T2), is written in Python (T8) and available[35] (T12) as a CLI implementation (T9, T10) under *Apache License 2.0* (T11).

*D-REPR v2.9.3 (2019)* D-REPR [132] is a Python and Rust (T8) based mapping language processor which generates RDF using the D-REPR mapping language [132] (T5). D-REPR can access CSV, JSON, XML, spreadsheets, NetCDF files, and relational & non-relational databases (T1, T2). It applies data transformation with built-in or custom functions (T6). D-REPR retrieves the data, infers classes and necessary joins, applies pre-processing data transformations on the input data (T7), and generates RDF (T13a, T13b). D-REPR can export its RDF to a file as Turtle or a custom JSON format (T3, T4). D-REPR is available on GitHub[36] (T12) under *MIT* license as a CLI implementation and webapp (T9, T10, T11).

*RocketRML v1.11.3 (2019)* RocketRML [120] is a NodeJS implementation (T8) of an RML [46] processor (T5) with FnO [38, 39] support for data transformations (T6). It can use two different XML parsers for performance or XML specification compliance reasons (T13a) and supports files (T1) in JSON, XML, or CSV format (T2). RocketRML retrieves all data, applies FnO functions, joins if necessary and generates RDF (T13b). It executes the data transformation together with the schema transformation (T7). RDF is outputted to a file as RocketRML does not support RML's Logical Target yet to specify how the RDF graphs should be exported to one or multiple targets (T3). It is available as a NodeJS package on NPM, CLI implementation, Docker im-

[31] https://github.com/streamreasoning/TripleWave, last accessed 23/03/2022, date last commit on default branch 10/07/2019

[32] https://github.com/LinkedEOData/GeoTriples, last accessed 23/03/2022, date last commit on default branch 10/09/2021

[33] https://github.com/datAcron-project/RDF-Gen, last accessed 23/03/2022, date last commit on default branch 18/07/2019

[34] https://github.com/RMLio/RMLStreamer, last accessed 23/03/2022, date last commit on default branch 25/02/2022

[35] https://github.com/SDM-TIB/MapSDI, last accessed 23/02/2022, date last commit on default branch 25/10/2021

[36] https://github.com/usc-isi-i2/d-repr/, last accessed 23/03/2022, date last commit on default branch 14/06/2021

age, or through a web application (T9, T10). RocketRML is released[37] (T12) under *Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International* license (T11).

*SDM-RDFizer v4.0 (2020)* SDM-RDFizer [64] is a Python implementation (T8) of an RML [46] processor with a focus on efficient execution of RML mapping rules (T5). SDM-RDFizer uses optimized data structures such as indexes, relational algebra operators, and multi-threading to improve the execution. It also avoid generating duplicate RDF and executing duplicate joins between data (T13a). SDM-RDFizer can access files and relational databases (T1). SDM-RDFizer can transform SQL query results, CSV, TSV, JSON, and XML files to RDF (T2). It does not support data transformations with FnO [38, 39], but it relies on FunMap (Section 6.2) to achieve this (T6). This way, the SDM-RDFizer can focus only on the schema transformation and optimize it. SDM-RDFizer incrementally parses the input data, applies joins, and generates RDF graphs (T13b). RDF graphs are exported to a file (T3). SDM-RDFizer is available as a CLI implementation, Docker container or a Python module on the Python Package Index (T9, T10) and released[38] (T12) under *Apache License 2.0* (T11).

*FunMap v1.0 (2020)* FunMap [70] is a Python-based (T8) FnO [38, 39] function processor which pre-processes FnO functions in RML mapping rules [46] (T5, T7). This way, FunMap allows to use mapping rules containing RML and FnO to be executed on RML processors which does not support any FnO functions such as the SDM-RDFizer. Moreover, it optimizes the function execution by avoiding executing a function multiple times when the function yields the same result. (T6, T7, T13a). FunMap leverages existing RML processors to execute the mapping rules, therefore it inherits their characteristics (T1, T2, T3, T13b). FunMap is available as a CLI implementation (T9, T10) and released[39] (T12) under *Apache License 2.0* (T11).

*Chimera v2.2 (2020)* Chimera [116] generates RDF graphs (uplifting) through RML mapping rules (T5) and transforms RDF into various data formats (lowering) with Apache Velocity Templates (T5). Chimera uses uplifting for creating an RDF graph as intermediate format. Afterwards, lowering is applied to export the data in various non-RDF formats. This approach is common in public transportation use cases where it is desired to publish the same in multiple formats. Chimera leverages and extends the RMLMapper. It improves RMLMapper's memory utilization by not caching subjects and data records for mapping rules without joins, executing mapping rules by data source, and incremental batches. Chimera's optimizations differs from MapSDI (Section 6.2) and FunMap (Section 6.2) because MapSDI and FunMap optimize the mapping rules while Chimera optimizes the implementation to execute these mapping rules. Chimera also adds multi-thread execution of mapping rules to increase the RMLMapper's throughput and access to remote RDF stores for storing the generated RDF graphs (T13a, T13b). Since Chimera reuses the RMLMapper, it inherits its characteristics (T1, T2, T3, T4, T5, T6, T7, T8), but it uses an older version of the RMLMapper which did not implement RML's Logical Target and W3C Web of Things for web APIs yet. Chimera is released[40] (T12) as a CLI implementation under *Apache License 2.0* and the RMLMapper extension under *MIT* license (T9, T10, T11).

***Materialization systems for query-language-driven mapping languages***

*SPARQL-Generate v2.0.9 (2017)* SPARQL-Generate [86, 87] is the reference implementation of SPARQL-Generate mapping language [86, 87] (T5) on top of Apache Jena in Java (T8). SPARQL-Generate leverages Apache Jena and its SPARQL [109] implementation with SPARQL-Generate's extensions to process the SPARQL-Generate query. SPARQL-Generate supports files, HTTP web APIs, and streams (T1), as well as joins and SPARQL Functions to apply data transformations (T6). It can generate RDF from WebSocket streams, MQTT streams, HTTP web APIs, plain text with regular expressions, HTML, CSV, TSV, XML, JSON, GeoJSON, CBOR, and HDT files (T2). SPARQL-Generate fetches all data, applies SPARQL Functions and joins if applicable, generates RDF graphs (T13a, T13b) and exports them to a file (T3). SPARQL Functions are executed by the underlying SPARQL engine together with the schema transformation (T7). SPARQL-Generate is available as a CLI implementation, a Java library, a webapp, and inside the Sublime Text editor (T9, T10). It is released[41] (T12) under *Apache License 2.0* (T11).

*SPARQL-Anything v0.4.1 (2021)* SPARQL-Anything [34] implements Facade-X's SPARQL `SERVICE` operator overriding (T5) for generating RDF from heterogeneous (semi-) structured data. It can access files (T1) in CSV, JSON, HMTL, XML, RDF, and plain text formats. SPARQL-Anything can also access data in archives, spreadsheets, images, and encoded metadata data, e.g., EXIF data in images (T2). It is implemented on top of Apache Jena SPARQL engine in Java (T8). Since SPARQL-Anything reuses an existing SPARQL engine, support for data transformations depends on the underlying engine (T6). Thus, the SPARQL Functions are executed together with the schema transformation (T7). SPARQL-Anything retrieves all data in memory and transforms it to RDF (T13a, T13b). The generated RDF graphs are exported to a file or are available through a Fuseki SPARQL endpoint (T3). SPARQL-Anything is available as a CLI implementation or a SPARQL endpoint (T9, T10). It is released[42]

[37] https://github.com/semantifyit/RocketRML, last accessed 23/03/2022, date last commit on default branch 30/11/2021

[38] https://github.com/SDM-TIB/SDM-RDFizer, last accessed 23/03/2022, date last commit on default branch 18/03/2022

[39] https://github.com/SDM-TIB/FunMap, last accessed 23/03/2022, date last commit on default branch 09/01/2021

[40] https://github.com/cefriel/chimera, last accessed 23/03/2022, date last commit on default branch 01/10/2021

[41] https://github.com/sparql-generate/sparql-generate, last accessed 07/04/2022, date last commit on default branch 07/04/2022

[42] https://github.com/SPARQL-Anything/sparql.anything, last accessed 23/03/2022, date last commit on default branch 18/03/2022

(T12) under *Apache License 2.0* (T11).

***Materialization systems for constraint-driven mapping languages***

*ShExML v0.2.6 (2020)* ShExML [51] is the reference implementation of the ShExML mapping language [51] (T5) in Scala (T8). It generates RDF using ShExML mapping rules. ShExML can also generate the ShEx [108] validation shapes and translate ShExML mapping rules into RML [46] mapping rules (T6). ShExML can access HTTP web APIs, files in XML, JSON, CSV, or TSV format, and relational databases (T1, T2). ShExML retrieves the data, applies joins if needed and generates RDF graphs (T13a, T13b) which are exported to a file (T3). ShExML does not provide any data transformations such as FnO (T6, T7). ShExML is available as a CLI implementation, Java library, and a webapp (T9, T10) and released on GitHub[43] (T12) under the *MIT* license (T11).

## 6.3. Virtualization implementations

In this Section, we discuss virtualization systems for transforming (semi-)structured heterogeneous data into knowledge graphs as RDF with mapping languages (Table 1). We applied a similar categorization to virtualization systems as in Section 4 *Schema transformations* and list them in chronological order. All virtualization implementations discussed in this Section implement dedicated mapping languages such as R2RML or RML.

We studied each virtualization system and created a list of characteristics specific for virtualization which differ between the discussed systems. We discuss two characteristics for each virtualization system (Tables 9, 10):

**T13a: Features** Each system has its own set of features regarding virtual access to its heterogeneous (semi-)structured data sources. Which features are applied to the virtualization process?

**T13b: Federation** Virtualization systems access multiple data sources typically using federation. Does the system support federation or not?

*Morph-RDB extension (2010)* Morph-RDB extension [19] enables Ontology-Based Data Access to streaming data sources by extending $R_2O$ [19] mapping rules with streaming support ($S_2O$ mapping rules) (T5) and querying these data sources with $SPARQL_{Stream}$ (T13a). Morph-RDB's $SPARQL_{Stream}$ is inspired by C-SPARQL [8] and SNEEql [17] but with support for streaming windows and SPARQL 1.1 aggregates [109]. Morph-RDB transforms $SPARQL_{Stream}$ queries into SNEE queries (SNEEql), the SNEE engine executes the query on the data sources. Morph-RDB generates the triples to answer the $SPARQL_{Stream}$ query and returns them (T3, T4). The $S_2O$ mappings are used to transform the SNEE query results back into RDF (T4). Morph-RDB supports files in CSV format, SQL query results of relational databases and sensor streams (T1, T2). Morph-RDB does not support any data transformations (T6) or federation (T13b). It is written in Scala (T8). Morph-RDB is accessible as a CLI implementation and provides a Docker image (T9, T10). Morph-RDB's extension is released[44] (T12) under the *Apache License 2.0* (T11).

*Ontop v4.1.1 (2013)* Ontop [112] virtualizes access to relational SQL databases and OpenGIS by translating SPARQL queries into SQL queries. It leverages R2RML [35] mapping rules or the Ontop Mapping Language (T5) to perform this translation, and extensions for accessing OpenGIS data (T1, T2). Generated triples are returned as SPARQL query results (T3, T4). Ontop does not support data transformations (T6), advanced SPARQL features such as property paths, existential queries with `[NOT] EXIST`, or basic query federation with `SERVICE` (T13b). However, efforts such as Denodo[45], Dremio[46], or Teiid[47] extend Ontop with federation support (T13b). Ontop uses an Intermediate Query language (T13a) with well known SQL optimizations, such as (i) redundant join elimination and pushing down joins to the data-level, and (ii) virtualization-specific optimizations for SQL derived from the `OPTIONAL` and `MINUS` SPARQL constructs (T10e). Ontop translates SPARQL aggregates to SQL aggregates for the database engine to perform the aggregation (T13a). Ontop supports ontological entailment by relying on a mapping saturation approach: in an offline phase, the mapping rules are saturated with the ontology rules, meaning additional mapping rules are generated for entailed triples. In previous generations of Ontop, ontological entailment was supported with a query rewriting approach using the PerfectRef algorithm [23], but that was abandoned as mapping saturation proved more efficient. Ontop supports entailment for RDFS and OWL 2 QL, and has experimental support for entailment with SWRL rules (T13a). Ontop has been integrated in the Optique Platform (2013) [76, 52] to access data such as relational databases, triple stores, temporal databases, data streams, etc. Optique leverages User Defined Functions as data transformation for accessing external data sources, windowing, data mining. However, Optique is not publicly available, while Ontop is available as a CLI implementation and Docker image (T9, T10). Ontop is written in Java (T8) and released[48] (T12) under *Apache License 2.0* (T11).

*XGSN v2.0.1 (2014)* XGSN [21] leverages Global Sensor Networks (GSN) middleware to provide virtualization over Internet of Things sensors using the W3C-recommended SSN ontology [55] (T5). XGSN uses wrappers to interface with sensors such as UDP, serial, HTTP, MQTT, etc. (T1, T2, T6). Data transformations are applied in data wrappers when

[43] https://github.com/herminiogg/ShExML, last accessed 23/03/2022, date last commit on default branch 28/02/2022

[44] https://github.com/oeg-upm/morph-rdb, last accessed 23/03/2022, date last commit on default branch 08/05/2021

[45] https://ontop-vkg.org/tutorial/federation/denodo/, last accessed 10/11/2021

[46] https://ontop-vkg.org/tutorial/federation/dremio/, last accessed 10/11/2021

[47] https://ontop-vkg.org/tutorial/federation/teiid/, last accessed 10/11/2021

[48] https://github.com/ontop/ontop, last accessed 23/03/2022, date last commit on default branch 21/03/2022

accessing the input data (T7). XGSN configures each sensor using a custom XML document to specify the wrapper, sampling rate, storage size, and which sensor values to expose through the SSN ontology. While XGSN can access (semi-)structured heterogeneous data from various sensors, it is limited to sensor data only. XGSN allows to use different RDF stream processors (T13a) – such as CQELS – which provide a SPARQL interface to access the RDF graphs (T3, T4). XGSN does not support federation (T13b). XGSN is written in Java (T8) as a CLI implementation, and can be accessed through a webapp and an API (T9, T10). XGSN is available on GitHub[49] (T12) under the *GNU General Public License v3.0* (T11).

*Morph-RDB v3.12.5 (2014)* Morph-RDB [107] – also discussed in Section 6.2 as materialization system – supports virtualization for relational databases, such as MySQL, PostgresSQL, H2, and MonetDB or files in CSV format (T1, T2) through R2RML [35] mapping rules (T5). Morph-RDB translates unions, filters and aggregation from SPARQL to SQL to reduce the client-side processing (T13a). Morph-RDB applies two types of well-known optimizations (T13a) on SQL queries: (i) self-join elimination, and (ii) subquery elimination. Morph-RDB does not use an intermediate query language, nor does it supports data transformations (T6, T7), ontology entailment (T13a), and federation (T13b). Morph-RDB is accessible as a CLI implementation written in Scala (T8) and provides a Docker image (T9, T10). Morph-RDB is released[50] (T12) under *Apache License 2.0* (T11).

*Morph-xR2RML v1.3.1 (2015)* Morph-xR2RML [98] – also discussed in Section 6.2 as materialization implementation – is a fork of Morph-RDB [107] which translates SPARQL queries into SQL queries and MongoDB queries using xR2RML mapping rules (T1, T2, T5, T13a). Morph-xR2RML does not support data transformations (T6) and not all operators of SPARQL such as joins and some filters, therefore the query is translated in two steps: (i) an abstract query is generated from the SPARQL query using the xR2RML mapping rules. (ii) the abstract query is translated into MongoDB queries. Untranslated parts of the query are processed by xR2RML client-side. Morph-xR2RML[51] provides a CLI implementation written in Scala (T8) or as a SPARQL endpoint (T9, T10) and is available (T12) under *Apache License 2.0* (T11).

*SparqlMap-M v0.7.4 (2016)* SparqlMap-M [126] is an extension of SparqlMap [127] to provide virtualization over CSV files, relational databases, and non-relational databases using R2RML mapping rules [35] (T5). SparqlMap-M extends R2RML [35] to support data transformations and conditions (T6) which are executed during the schema transformation (T7). SparqlMap-M analyses SPARQL queries [109] to reduce the R2RML mappings to the minimum for answering the query and translates to SQL to execute the query on the the database engine (T13a). SparqlMap-M does not implement support for SPARQL Federated Queries (T13b). SparqlMap-M is written in Java (T8) and available as a CLI implementation or SPARQL endpoint on GitHub[52] without a license (T9, T10, T11, T12).

*Morph-streams++ v1.0.10 (2016)* Morph-streams++ [89] is the successor of the extended Morph-RDB and leverages $\text{SPARQL}_{\text{Stream}}$ [20] queries. Morph-streams++ uses existing Distributed Stream Management Systems (DSMS) to execute $\text{SPARQL}_{\text{Stream}}$ queries with R2RML mapping rules [35] (T5). The $\text{SPARQL}_{\text{Stream}}$ queries are first translated into queries supported by the underlying DSMS and its results are translated into RDF with R2RML mapping rules (T13a). Generated RDF graphs are returned as $\text{SPARQL}_{\text{Stream}}$ query results (T3, T4). Supported data sources and formats in Morph-streams++ depend on the underlying DSMS: currently the Esper[53] and Global Sensors Network (GSN) [1] (T1, T2) are supported by Morph-streams++. Morph-streams++ does not support any data transformations (T6, T7), or federation (T13b). Morph-streams++ is written in Scala (T8) and released[54] (T12) as a CLI implementation without a license (T9, T10, T11).

*Squerall v0.2 (2019)* Squerall [93] answers SPARQL queries [109] over heterogeneous (semi-)structured data sources that are mapped to RDF with RML [46] (T5) and FnO functions [38, 39] (T6). Squerall leverages the distributed data processing systems Spark or Presto: both systems use an internal tabular data format into which different data sources can be (virtually) integrated. Spark and Presto allow the manipulation of multiple heterogeneous (semi-)structured data sources in a uniform, SQL-like manner. Squerall translates SPARQL queries into Spark's or Presto's SQL queries for execution (T13a). Squerall applies FnO functions during its schema transformation with RML mapping rules (T7). Squerall only supports translation of SPARQL aggregations; other SPARQL operators are considered future work (T13a). Squerall can access Apache Parquet, CSV files, relational databases with JDBC connectors, and non-relational databases, such as MongoDB, Elasticsearch, or Couchbase (T1, T2). Squerall[55] is written in Java (T8) and available under *Apache License 2.0* (T12) as a CLI implementation and via a GUI (T9, T10).

*Ontario (2019)* Ontario [48] provides virtualization over a set of SPARQL endpoints [109] and non-SPARQL database interfaces using RML mapping rules [46] (T5). It translates queries into star shaped subqueries (T13a) and are com-

[49] https://github.com/LSIR/gsn, last accessed 23/03/2022, date last commit on default branch 12/03/2017

[50] https://github.com/oeg-upm/morph-rdb, last accessed 23/03/2022, date last commit on default branch 08/05/2021

[51] https://github.com/frmichel/morph-xr2rml, last accessed 23/03/2022, date last commit on default branch 13/01/2022

[52] https://github.com/tomatophantastico/sparqlmap, last accessed 23/03/2022, date last commit on default branch 31/08/2017

[53] http://www.espertech.com/esper, last accessed 10/11/2021

[54] https://github.com/jpcik/morph-streams, last accessed 23/03/2022, date last commit on default branch 28/09/2016

[55] https://github.com/EIS-Bonn/Squerall, last accessed 23/03/2022, date last commit on default branch 10/09/2021

bined with RML mapping rules to select the right SPARQL endpoint to answer the subquery. The results of each subquery are joined locally (T13a). Ontario supports SPARQL endpoints, relational databases, such as MySQL, files (locally or on Apache Hadoop) in CSV, TSV, and XML format, and non-relational databases such as MongoDB and graph databases, such as Neo4J (T1, T2). It implements SPARQL Federated Queries via SPARQL's `SERVICE` operator (T13b), but it does not support data transformations (T6). Ontario is written in Python (T8) available as CLI implementation[56] under the *General Public License 2.0* (T9, T10, T11 T12).

*Morph-GraphQL v1.0.0 (2020)* Morph-GraphQL [26] is a virtualization system written in NodeJS (T8) for translating heterogeneous (semi-)structured data into RDF and exposing it over GraphQL (T3, T4) with R2RML and RML mappings (T5). Morph-GraphQL supports relational databases e.g. MySQL and files in CSV format (T1, T2). Morph-GraphQL does not support data transformations (T6, T7). Morph-GraphQL is available as CLI implementation and Docker image on GitHub[57] under *Apache License 2.0* (T9, T10, T11, T12). However, the authors marked Morph-GraphQL as deprecated in its repository.

*Obi-Wan (2020)* Obi-Wan [18] is a Java-based (T8) virtualization system for data stored in relational and non-relational databases. Obi-Wan uses a custom Global-Local As View (GLAV) mapping language (T5) to link heterogeneous data to an RDF representation. Obi-Wan supports relational databases e.g. PostgreSQL, non-relational databases e.g. MongoDB, Redis, and triple stores e.g. Jena TDB (T1, T2). Obi-Wan supports multiple entailment strategies such as query rewriting, mapping saturation, and hybrid rewriting-saturation (T13a). Obi-Wan returns the generated triples as SPARQL query results (T3, T4). To the best of our knowledge, Obi-Wan does not support data transformations (T6, T7), or SPARQL Federated Queries (T13b). Obi-Wan is available as CLI implementation and webapp on GitLab[58] under *MIT* license (T9, T10, T11 T12).

### 6.4. Remarks

Our systematic literature review focuses on systems which generate RDF from heterogeneous (semi-)structured data. However, we discuss here a few systems which are excluded due to our exclusion criteria, but are important in this field.

Commercial systems for generating RDF also exist. We indicatively mention Virtuoso[59] which is an RDF triple store with support for R2RML, Oracle RDF[60] which allows Oracle databases to store semantic data as RDF, Stardog[61] which provides the Stardog Mappings to generate RDF from heterogeneous data, and CARML[62] which implements RML. However, these commercial systems are excluded following our inclusion and exclusion criteria which exclude commercial systems.

Different systems of the Morph family (Morph-RDB [107, 19], Morph-xR2RML [98], Morph-GraphQL [26], Morph-streams++ [89]) were included in the survey, but each of these systems grows independently. Only Morph-Skyline [53] and Morph-xR2RML build upon Morph-RDB. Besides these systems, the Morph family also contains Morph-CSV [28] which was excluded from our survey because it did not adhere to our inclusion and exclusion criteria (Step 4), as it only supports CSV data. Morph-Skyline was also excluded by our methodology in Step 4 since it only supports relational databases. However, we consider that it is worth mentioning because it implements RML, thus it has the potential to generate RDF from other data formats too.

X3ML engine [100, 94] transforms data into RDF with the X3ML mapping definition language. The exclusion criteria excluded the X3ML engine because it currently only supports XML as input data format. However, it may be extended to RDF as input format in the future.

Last, it is worth mentioning two systems: Morph-KGC [5] and EABlock [71], which were accepted while this journal was under review. Morph-KGC supports CSV, TSV, JSON and XML data, as well as relational databases. EABlock follows a similar approach to FunMap by transforming RML mapping rules with FnO functions into RML mapping rules without FnO functions with an efficient strategy to evaluate them.

## 7. Discussion

In this Section, we discuss the outcome of this systematic literature review of declarative RDF graph generation from (semi-)structured heterogeneous data. We discuss the highlights of this systematic review and its categories (*schema* and *data transformations*, as well as *materialization* and *virtualization systems*). Afterwards, we discuss the open issues we encountered, excluded approaches, automation, and the future of declarative RDF graph generation from heterogeneous (semi-)structured data.

*Overview* In this systematic literature review, we analyzed 52 articles focusing on declarative RDF graph generation from heterogeneous (semi-)structured data sources. We closely inspected the schema and data transformations proposed by different mapping languages, as well as systems for generating RDF graphs from heterogeneous (semi-)structured data. 18 articles introduce a schema transformation, 16 articles a data transformation, and 39 articles describe a system. 12 articles were submitted to workshops, 27 to conferences, and 13 to journals.

We observed that the study of mapping languages and their system implementations is getting more and more ma-

[56] https://github.com/SDM-TIB/Ontario, last accessed 23/03/2022, date last commit on default branch 09/03/2021

[57] https://github.com/oeg-upm/morph-graphql, last accessed 07/04/2022, date last commit on default branch 13/10/2021

[58] https://gitlab.inria.fr/cedar/obi-wan, last accessed 23/03/2022, date last commit on default branch 22/10/2021

[59] https://virtuoso.openlinksw.com, last accessed 07/04/2022

[60] https://docs.oracle.com/database/121/RDFRM/rdf-overview.htm, last accessed 07/04/2022

[61] https://docs.stardog.com/virtual-graphs/mapping-data-sources, last accessed 07/04/2022

[62] https://github.com/carml/carml, last accessed 07/04/2022

ture and we more often encounter solutions being proposed to conferences and journals the last few years. The number of systems is twice the number of mapping languages indicating that mapping languages are implemented and reused among systems to leverage the merits of declarative approaches.

*Schema transformations* We discussed 15 characteristics for each schema transformation (Section 4). 8 mapping languages were identified. We observed that 6 of them are either based on R2RML/RML [35, 46] (4 out of 6) or SPARQL [109] (2 out of 6). Some schema transformations provide their own custom syntax, such as D-REPR [132], or leverages other specifications, such as ShExML [51].

Only 2 mapping languages (D-REPR [132] and D2RML [30]) transform input data into an intermediate representation before applying their schema transformation. Almost all mapping languages describe how input data should be retrieved and accessed, but do not consider how the generated RDF should be exported (except RML [128]), thus, they do not declaratively describe the complete workflow.

Most mapping languages follow the same approach to access the input data and apply the schema transformations: they refer to the input data considering a language relevant to the input data. That does not hold though for the mapping languages where the input data is transformed to an intermediate representation early in the process. XSPARQL [15, 41] immediately translates a SPARQL query to an SQL or XPath query; Facade-X [34] directly maps to RDF and then applies custom schema transformations; D-REPR [132] to a JSON tree structure; and D2RML [30] to a tabular structure. An intermediate representation creates an additional processing step for the system, but reduces the complexity of the schema transformation in the system.

All mapping languages cover the RDF specification, besides D-REPR [132] that does not support named graphs and language tags. However, not all mapping languages support RDFS collections and containers. In fact, this is covered by all SPARQL-based languages, as SPARQL intuitively covers RDFS collections and containers, and only 1 mapping language based on R2RML/RML [35, 46]: xR2RML [98]. While SPARQL covers RDFS collections and containers, it was not designed to be used as a mapping language which requires additional extensions such as SPARQL-Generate [86], XSPARQL [15, 41], or Facade-X [34] to use it for declarative RDF generation from (semi-)structured heterogeneous data.

*Data transformations* 5 characteristics were discussed for data transformations (Section 5). We observed that most data transformations are dedicated to a certain schema transformation, but only in the case of RML, there are 3 alternative data transformations proposed (GeoTriples [83, 81], Function Ontology (FnO) [38, 39, 95], and FunUL [73]).

FnO [38, 39, 95] and SPARQL Functions [109] are the only data transformations not depending on a specific schema transformation. However, FnO is completely standalone from any schema transformation while SPARQL Functions are only available in a SPARQL query. Other data transformations such as D2RML [40] or KR2RML [122] are tightly integrated with their schema transformation.

There is a big diversity on how the different data transformations are defined (declaratively described, described in a string, or hard-coded in the system implementing the approach). All possible combinations of declaratively described data transformations' body and parameters are encountered. As opposed to schema transformations, data transformations do not converge on certain practices, indicating that there is still room for experimentation and improvement.

*Systems* We analyzed 14 characteristics for 19 materialization-based and 11 virtualization-based systems. We observed that, in most cases, a system is proposed for each introduced mapping language. However, only for one mapping language (R2RML/RML) and one data transformation (FnO), alternative systems are proposed. In total, 8 systems were proposed for schema transformation based on RML [46]: RMLMapper [46], GeoTriples [83, 81], RMLStreamer [54], MapSDI [72], RocketRML [120], SDM-RDFizer [64], FunMap [70], Chimera [116], and, 4 systems based on modified versions of R2RML [35]: Karma [122], Morph-RDB [107], Morph-xR2RML [98], and TripleWave [96]. 5 of the aforementioned systems also support data transformations based on FnO [38, 39, 95]: RMLMapper [46], RMLStreamer [54], RocketRML [120], FunMap [70], and Chimera [116]. To be complete, there is yet another materialization system based on RML, CARML[63], but it is not mentioned in our survey because there are no publications about it and it is used in a commercial setting.

Each alternative system provides its own set of features or optimizations for the mapping rules execution. We observed two main directions: data cleaning and vertical scaling, but each system achieves it in a different way. On the one hand, SDM-RDFizer [64] and MapSDI [72] rely their optimizations on duplicate removal, while FunMap [70] on functions pre-processing and their execution optimization. We noticed that all materialization systems apply data transformations either as a pre-processing step during the input data retrieval or during the schema transformation, but not after the schema transformation. On the other hand, TripleWave [96], RMLStreamer [54], SDM-RDFizer [64], and Chimera [116] opt for (horizontal and) vertical scaling.

While materialization systems follow similar directions to optimize the systems, the virtualization system offer a greater variety and experiment more with alternative solutions. SQL and subquerying optimizations, as well as ontology entailment are among the most common directions taken by different systems, but there is a broad range of alternatives being investigated. As opposed to materialization systems, virtualization systems seem to be less mature. During our analysis, we observed an evolution in maturity and optimizations for systems of the same schema or data transformation, but there is still room for improvement. Some systems e.g., Morph-RDB [107] support both materialization or virtual-

[63] https://github.com/carml/carml, last accessed 29/11/2021

ization, but they are not optimized for this. There are no systems that optimize both of them, let alone their combination.

*Open Issues* During this systematic literature review, we encountered several open issues for schema and data transformations and their corresponding systems:

Current data transformations do not provide support for conditional generation or post-processing of the generated RDF. Conditions allow mapping languages to specify if certain RDF graphs must be generated or not based upon the result of a data transformation. We observed that applying conditions are only vaguely described among data transformations. Conditions appeared in D2RQ but they were discarded in R2RML which is nowadays the reference mapping language. However, conditions might be useful in a mapping language beyond data transformations. Data transformations can only be applied before or during the RDF generation, but not after the RDF was generated. Currently, conditions and post-processing are an active topic of discussion in the W3C's Knowledge Graph Construction Community Group[64].

Web APIs as a data source are currently supported by different schema transformations, such as RML [128] and SPARQL-Generate [86]. However, data derived from a web API cannot be re-used to request data from other web APIs in any mapping language discussed in this systematic literature review. This limits access to web APIs since some web APIs need multiple requests to retrieve their data. A solution is currently being investigated in the W3C's Knowledge Graph Construction Community Group[65].

All systems evaluated in this systematic literature review support either materialization or virtualization, but not a combination of both. Some systems support both, but not at the same time and not optimized.

*Worth mentioning* This systematic literature review follows the survey methodology (Section 3) which excluded several schema transformations and systems. Several mapping languages which significantly influenced the evolution of mapping languages are not included in the survey methodology. We would like to mention D2RQ [33] mapping language which is the predecessor of R2RML [35], the W3C-recommended mapping language to generate RDF graphs from data in relational databases. These two languages are not included in our systematic literature review because they do not refer to heterogeneous data. Similarly, the Ontop Mapping Language [22] was also not included in the criteria for the same reasons as D2RQ, but we consider worth mentioning because of its active use for over a decade and its broad adoption. Last, other serializations of mapping languages, such as YARRRML [60], which was proposed as an alternative for a human-friendly representation of RML was also excluded since it was only presented as a demo. We consider it worth mentioning as it is broadly adopted by the community that works with RML.

*Automation* Given that the special issue is related to "automating" the knowledge graph generation, it is worth mentioning that the automation of the mapping rules' definition and knowledge graph generation is not broadly investigated so far. A few of the most recent systems look into automating the planning for optimal mapping rules execution, such as RMLStreamer [54], FunMap [70], and SDM-RDFizer [64], but there is still room for further investigation.

Despite the declarative nature of schema and data transformations, that decouples the mapping rules from their system, only few solutions aim to (semi-)automate the definition of mapping rules, or keep in some cases the *human-in-the-loop* too. MIRROR [97], AutoMap4OBDA [119], and BootOX [69] are a few of the most well-known approaches which automate the generation of mapping rules. Most solutions nowadays aim to automate the knowledge graph generation without considering the declarative description with mapping rules, see, for instance, the solutions proposed for the SemTab challenge [67, 66, 68]. The lack of automated solutions for mapping rules definition and the adjustment of existing automated solutions to generate first the mapping rules and then the knowlede graph is discussed in a position paper [44] where some preliminary thoughts on the extension of existing automation approaches to align with declarative mapping languages are discussed.

*Future* The schema and data transformations of different mapping languages, as well as the corresponding systems discussed in this systematic literature review are subject of the W3C's Knowledge Graph Construction Community Group[66]. The goal of the community group is to introduce a new recommendation, as a successor of R2RML, but for heterogeneous data sources. Our aim is to support the community group's activities with this systematic literature review and provide an online version [67] that will be actively maintained by the community group in the long term.

*Completion time* We started with this systematic literature review in 2019 and finished it in November 2021. A systematic literature review helps other researchers to establish a baseline for their future research on schema and data transformations for heterogeneous (semi-)structured data.

## 8. Conclusion

Previous studies and surveys focused on generating RDF graphs from relational databases [57, 59, 50], a certain type of systems, i.e., only on virtualization [133] approaches, but no materialization approaches, or how the integration is performed between data, but not the generation of RDF graphs [102]. So far, there is no systematic literature review that provides an overview of RDF graph generation from heterogeneous

[64] https://www.w3.org/community/kg-construct/, last accessed 10/11/2021

[65] https://www.w3.org/community/kg-construct/, last accessed 10/11/2021

[66] https://www.w3.org/community/kg-construct/, last accessed 10/11/2021

[67] https://w3id.org/kg-construct/survey, last accessed 22/05/2022

data. The significant number of mapping languages that were listed and the different approaches for performing schema and data transformations as well as the high number of systems prove that the systematic study of the domain is required. Our systematic literature review provides an overview of the last 20 years of research around mapping languages, providing a closer look on their schema and data transformations, as well as their systems. We discuss for each of them a set of characteristics to provide an overview for other researchers. We observed an evolution in maturity of schema and data transformations and their corresponding systems. However, there is still room for improvement and experimentation.

Since this is an active research domain, new approaches are frequently discovered which would not be discussed in this systematic literature review. To address this, we provide an online version[68] where new approaches can be easily added to keep this overview up-to-date.

## Acknowledgements

This research was funded by the Special Research Fund of Ghent University[69] under grant BOF20/DOC/132.

## CRediT authorship contribution statement

**Dylan Van Assche:** Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – Original draft, Writing – Review & editing, Visualization, Funding acquisition. **Thomas Delva:** Validation, Investigation, Resources, Writing – Original draft. **Gerald Haesendonck:** Validation, Investigation, Resources, Writing – Original draft. **Pieter Heyvaert:** Validation, Investigation, Resources, Writing – Original draft. **Ben De Meester:** Validation, Investigation, Resources, Writing – Original draft, Writing – Review & editing. **Anastasia Dimou:** Conceptualization, Methodology, Validation, Formal analysis, Investigation, Resources, Writing – Original draft, Writing – Review & editing, Visualization, Supervision, Project administration, Funding acquisition.

## References

[1] Aberer, K., Hauswirth, M., Salehi, A., 2006. Global sensor networks. Technical Report.

[2] A.M., R., E., A., S., A., M., M., O.H., A., M., Y.G., S., H.A., M., H., 2021. Artificial intelligence approaches and mechanisms for big data analytics: a systematic study. Computer Science .

[3] Arenas, M., Bertails, A., Prud'hommeaux, E., Sequeda, J., 2012. A Direct Mapping of Relational Data to RDF. Recommendation. World Wide Web Consortium (W3C). URL: http://www.w3.org/TR/rdb-direct-mapping/.

[4] Arenas, M., Pérez, J., Riveros, C., 2009. The recovery of a schema mapping: Bringing exchanged data back.

[5] Arenas-Guerrero, J., Chaves-Fraga, D., Toledo, J., S. Pérez, M., Corcho, O., 2022. Morph-kgc: Scalable knowledge graph materialization with mapping partitions. Semantic Web Journal .

[6] Arenas-Guerrero, J., Scrocca, M., Iglesias-Molina, A., Toledo, J., Gilo, L.P., Dona, D., Corcho, O., Chaves-Fraga, D., 2021. Knowledge graph construction with r2rml and rml: An etl system-based overview, in: Proceedings of the 2nd International Workshop on Knowledge Graph Construction.

[7] Atzori, M., 2014. Toward the Web of Functions: Interoperable Higher-Order Functions in SPARQL, Springer, Cham, Switzerland. pp. 406–421. doi:10.1007/978-3-319-11915-1_26.

[8] Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M., 2009. C-SPARQL: SPARQL for continuous querying, in: Proceedings of the 18th international conference on World Wide Web, ACM, New York, USA. pp. 1061–1062. URL: http://dx.doi.org/10.1145/1526709.1526856, doi:10.1145/1526709.1526856.

[9] Barrasa, J., Corcho, O., Gomez-Perez, A., 2004. R2o, an extensible and semantically based database-to-ontology mapping language .

[10] Battle, R., Kolas, D., 2011. GeoSPARQL: enabling a geospatial Semantic Web. Semantic Web Journal 3, 355–370.

[11] Ben-Kiki, O., Evans, C., döt Net, I., 2009. YAML Ain't Markup Language (YAML™) Version 1.2. techreport. URL: https://yaml.org/spec/1.2/spec.html.

[12] Bereta, K., Smeros, P., Koubarakis, M., 2013. Representation and querying of valid time of triples in linked geospatial data, in: Extended Semantic Web Conference, Springer. pp. 259–274.

[13] Bereta, K., Xiao, G., Koubarakis, M., 2019. Ontop-spatial: Ontop of geospatial databases. Journal of Web Semantics 58, 100514. URL: https://www.sciencedirect.com/science/article/pii/S1570826819300447, doi:https://doi.org/10.1016/j.websem.2019.100514.

[14] Bikakis, N., Tsinaraki, C., Gioldasis, N., Stavrakantonakis, I., Christodoulakis, S., 2016. The XML and Semantic Web Worlds: Technologies, Interoperability and Integration. A Survey of the State of the Art. arXiv e-prints , arXiv:1608.03556arXiv:1608.03556.

[15] Bischof, S., Decker, S., Krennwallner, T., Lopes, N., Polleres, A., 2012. Mapping between RDF and XML with XSPARQL. Journal on Data Semantics 1, 147–185. URL: https://doi.org/10.1007/s13740-012-0008-7, doi:10.1007/s13740-012-0008-7.

[16] Bizer, C., 2003. D2r map – a database to rdf mapping language .

[17] Brenninkmeijer, C.Y.A., Galpin, I., Fernandes, A.A.A., Paton, N.W., 2008. A semantics for a query language over sensors, streams and relations, in: Gray, A., Jeffery, K., Shao, J. (Eds.), Sharing Data, Information and Knowledge, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 87–99.

[18] Buron, M., Goasdoué, F., Manolescu, I., Mugnier, M.L., 2020. Obi-wan: Ontology-based rdf integration of heterogeneous data. Proc. VLDB Endow. 13, 2933–2936. URL: https://doi.org/10.14778/3415478.3415512, doi:10.14778/3415478.3415512.

[19] Calbimonte, J.P., Corcho, O., Gray, A.J.G., 2010. Enabling ontology-based access to streaming data sources, in: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (Eds.), The Semantic Web – ISWC 2010, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 96–111.

[20] Calbimonte, J.P., Jeung, H., Corcho, O., Aberer, K., 2012. Enabling query technologies for the semantic sensor web. International Journal On Semantic Web and Information Systems (IJSWIS) 8, 43–63.

[21] Calbimonte, J.P., Sarni, S., Eberle, J., Aberer, K., 2014. Xgsn: An open-source semantic sensing middleware for the web of things, in: TC/SSN@ISWC.

[22] Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G., 2017. Ontop: Answering SPARQL Queries over Relational Databases. Semantic Web Journal 8, 471–487. doi:10.3233/SW-160217.

[23] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R., 2007. Tractable reasoning and efficient query answering in description logics: The dl-lite family. Journal of Automated reasoning 39, 385–429.

[24] Chang, S., 2018. Scaling knowledge access and retrieval at airbnb. URL: https://medium.com/airbnb-engineering/scaling-knowledge-access-and-retrieval-at-airbnb-665b6ba21e95.

[68] https://w3id.org/kg-construct/survey, last accessed 22/05/2022

[69] https://www.ugent.be/en/research/funding/bof/overview.htm, last accessed 10/11/2021

[25] Chaves-Fraga, D., Endris, K.M., Iglesias, E., Corcho, O., Vidal, M.E., 2019. What are the parameters that affect the construction of a knowledge graph?, in: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer. pp. 695–713.

[26] Chaves-Fraga, D., Priyatna, F., Alobaid, A., Corcho, O., 2020a. Exploiting declarative mapping rules for generating graphql servers with morph-graphql. International Journal of Software Engineering and Knowledge Engineering 30, 785–803. URL: https://doi.org/10.1142/S0218194020400070, doi:10.1142/S0218194020400070.

[27] Chaves-Fraga, D., Priyatna, F., Cimmino, A., Toledo, J., Ruckhaus, E., Corcho, O., 2020b. Gtfs-madrid-bench: A benchmark for virtual knowledge graph access in the transport domain. Journal of Web Semantics 65, 100596. URL: https://www.sciencedirect.com/science/article/pii/S1570826820300354, doi:https://doi.org/10.1016/j.websem.2020.100596.

[28] Chaves-Fraga, D., Ruckhaus, E., Priyatna, F., Vidal, M.E., Corcho, O., 2021. Enhancing virtual ontology based access over tabular data with morph-csv. Semantic Web , 1–34doi:10.3233/SW-210432.

[29] Chortaras, A., Stamou, G., 2018a. D2rml: Integrating heterogeneous data and web services into custom rdf graphs, in: LDOW@WWW.

[30] Chortaras, A., Stamou, G., 2018b. Mapping Diverse Data to RDF in Practice, in: Vrandečić, D., Bontcheva, K., Suárez-Figueroa, M.C., Presutti, V., Celino, I., Sabou, M., Kaffee, L.A., Simperl, E. (Eds.), The Semantic Web – ISWC 2018, Springer, Cham. pp. 441–457. doi:10.1007/978-3-030-00671-6.

[31] Corby, O., Faron-Zucker, C., 2015. Sttl: A sparql-based transformation language for rdf, in: Proceedings of the 11th International Conference on Web Information Systems and Technologies - WEBIST. doi:10.5220/0005450604660476.

[32] Corby, O., Faron-Zucker, C., Gandon, F., 2017. Ldscript: A linked data script language, in: d'Amato, C., Fernandez, M., Tamma, V., Lecue, F., Cudré-Mauroux, P., Sequeda, J., Lange, C., Heflin, J. (Eds.), The Semantic Web – ISWC 2017, Springer International Publishing, Cham. pp. 208–224.

[33] Cyganiak, R., Bizer, C., Garbers, J., Maresch, O., Becker, C., 2012. The D2RQ Mapping Language. Technical Report. FU Berlin, DERI, UCB, JP Morgan Chase, AGFA Healthcare, HP Labs, Johannes Kepler Universität Linz. URL: http://d2rq.org/d2rq-language.

[34] Daga, E., Asprino, L., Mulholland, P., Gangemi, A., 2021. Facade-x: an opinionated approach to sparql anything. arXiv preprint arXiv:2106.02361 .

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

[36] De Meester, B., Dimou, A., Verborgh, R., Mannens, E., 2017a. Detailed Provenance Capture of Data Processing, in: Garijo, D., van Hage, W.R., Kauppinen, T., Kuhn, T., Zhao, J. (Eds.), Proceedings of the First Workshop on Enabling Open Semantic Science (SemSci), CEUR. pp. 31–38. URL: http://ceur-ws.org/Vol-1931/#paper-05.

[37] De Meester, B., Heyvaert, P., Verborgh, R., Dimou, A., 2019. Mapping languages: analysis of comparative characteristics, in: Proceedings of First Knowledge Graph Building Workshop. URL: https://openreview.net/forum?id=HklWL4erv4.

[38] De Meester, B., Maroy, W., Dimou, A., Verborgh, R., Mannens, E., 2017b. Declarative Data Transformations for Linked Data Generation: the case of DBpedia, Springer, Cham. pp. 33–48. URL: https://link.springer.com/chapter/10.1007/978-3-319-58451-5_3, doi:10.1007/978-3-319-58451-5_3.

[39] De Meester, B., Seymoens, T., Dimou, A., Verborgh, R., 2020. Implementation-independent Function Reuse. Future Generation Computer Systems 110, 946–959. URL: https://ben.de-meester.org/research/preprints/DeMeester2019Implementation.pdf, doi:10.1016/j.future.2019.10.006.

[40] Debruyne, C., O'Sullivan, D., 2016. R2RML-F: Towards Sharing and Executing Domain Logic in R2RML Mappings, in: Workshop on Linked Data on the Web, CEUR.

[41] Dell'Aglio, D., Polleres, A., Lopes, N., Bischof, S., 2014. Querying the web of data with xsparql 1.1, in: SEMWEB.

[42] Delva, T., Van Assche, D., Heyvaert, P., De Meester, B., Dimou, A., 2021. Integrating nested data into knowledge graphs with RML fields, in: Chaves-Fraga, D., Dimou, A., Heyvaert, P., Priyatna, F., Sequeda, J. (Eds.), Proceedings of the 2nd International Workshop on Knowledge Graph Construction co-located with 18th Extended Semantic Web Conference (ESWC 2021), CEUR. URL: http://ceur-ws.org/Vol-2873/paper9.pdf.

[43] Devarajan, D., 2017. Happy birthday watson discovery. URL: https://www.ibm.com/blogs/bluemix/2017/12/happy-birthday-watson-discovery.

[44] Dimou, A., Chaves-Fraga, D., 2022. Declarative description of knowledge graphs construction automation: Status & challenges .

[45] Dimou, A., Sande, M.V., Slepicka, J., Szekely, P., Mannens, E., Knoblock, C., Walle, R.V.d., 2014a. Mapping hierarchical sources into rdf using the rml mapping language, in: 2014 IEEE International Conference on Semantic Computing, pp. 151–158. doi:10.1109/ICSC.2014.25.

[46] Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R., 2014b. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: Bizer, C., Heath, T., Auer, S., Berners-Lee, T. (Eds.), Proceedings of the 7th Workshop on Linked Data on the Web, CEUR-WS.org. URL: http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

[47] Dimou, A., Verborgh, R., Sande, M.V., Mannens, E., de Walle, R.V., 2015. Machine-interpretable dataset and service descriptions for heterogeneous data access and retrieval, in: Proceedings of the 11th International Conference on Semantic Systems - SEMANTICS '15, ACM Press. doi:10.1145/2814864.2814873.

[48] Endris, K.M., Rohde, P.D., Vidal, M.E., Auer, S., 2019. Ontario: Federated Query Processing Against a Semantic Data Lake, in: Hartmann, S., Küng, J., Chakravarthy, S., Anderst-Kotsis, G., Tjoa, A.M., Khalil, I. (Eds.), Database and Expert Systems Applications: 30th International Conference, DEXA, Part I, Springer, Cham. pp. 379–395. doi:10.1007/978-3-030-27615-7_29.

[49] Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C., 2005. Composing schema mappings: Second-order dependencies to the rescue. ACM Trans. Database Syst. 30, 994–1055. URL: https://doi.org/10.1145/1114244.1114249, doi:10.1145/1114244.1114249.

[50] Fiorelli, M., Stellato, A., 2021. Lifting tabular data to rdf: A survey, in: Garoufallou, E., Ovalle-Perandones, M.A. (Eds.), Metadata and Semantic Research, Springer International Publishing, Cham. pp. 85–96.

[51] García-González, H., Boneva, I., Staworko, S., Labra-Gayo, J.E., Lovelle, J.M.C., 2020. ShExML: improving the usability of heterogeneous data mapping languages for first-time users. PeerJ Computer Science 6, e318.

[52] Giese, M., Soylu, A., Vega-Gorgojo, G., Waaler, A., Haase, P., Jimenez-Ruiz, E., Lanti, D., Rezk, M., Xiao, G., Ozcep, O., Rosati, R., 2015. Optique: Zooming in on big data. Computer 48, 60–67. doi:10.1109/MC.2015.82.

[53] Goncalves, M., Chaves-Fraga, D., Corcho, O., 2020. Morph-skyline: Skyline queries for virtual knowledge graph access.

[54] Haesendonck, G., Maroy, W., Heyvaert, P., Verborgh, R., Dimou, A., 2019. Parallel RDF generation from heterogeneous big data, in: Groppe, S., Gruenwald, L. (Eds.), Proceedings of the International Workshop on Semantic Big Data - SBD '19, ACM Press, Amsterdam, Netherlands. URL: https://biblio.ugent.be/publication/8619808/file/8659668.pdf, doi:10.1145/3323878.3325802.

[55] Haller, A., Janowicz, K., Cox, S., Le Phuoc, D., Taylor, K., Lefrançois, M., 2017. Semantic Sensor Network Ontology. Working Group Recommendation. World Wide Web Consortium (W3C). URL: https://www.w3.org/TR/vocab-ssn/.

[56] Hamad, F., Liu, I., , Zhang, X.X., 2018. Food discovery with uber eats: Building a query understanding engine. URL: https://eng.uber.com/uber-eats-query-understanding.

[57] Hazber, M., Li, R., Li, B., Zhao, Y., Alalayah, K., 2019. A sur-

vey: Transformation for integrating relational database with semantic web, in: Proceedings of ICMSS 2019, pp. 66–73. doi:10.1145/3312662.3312692.

[58] He, Q., Chen, B.C., Agarwal, D., 2016. Building the linkedin knowledge graph. URL: https://engineering.linkedin.com/blog/2016/10/building-the-linkedin-knowledge-graph.

[59] Hert, M., Reif, G., Gall, H.C., 2011. A Comparison of RDB-to-RDF Mapping Languages, in: Proc. of the 7th International Conference on Semantic Systems, ACM. pp. 25–32. URL: http://doi.acm.org/10.1145/2063518.2063522, doi:10.1145/2063518.2063522.

[60] Heyvaert, P., De Meester, B., Dimou, A., Verborgh, R., 2018. Declarative Rules for Linked Data Generation at your Fingertips!, 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, Springer, Cham, Heraklion, Crete, Greece. URL: https://2018.eswc-conferences.org/files/posters-demos/paper_297.pdf, doi:https://doi.org/10.1007/978-3-319-98192-5.

[61] Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutiérrez, C., Gayo, J.E.L., Kirrane, S., Neumaier, S., Polleres, A., Navigli, R., Ngomo, A.N., Rashid, S.M., Rula, A., Schmelzeisen, L., Sequeda, J.F., Staab, S., Zimmermann, A., 2020. Knowledge graphs. CoRR abs/2003.02320. URL: https://arxiv.org/abs/2003.02320, arXiv:2003.02320.

[62] Holm, J., Thomas, G., Hendler, J., Musialek, C., 2012. Us government linked open data: Semantic.data.gov. IEEE Intelligent Systems 27, 25–31. doi:10.1109/MIS.2012.27.

[63] Hyland, B., Atemezing, G.A., Villazón-Terrazas, B., 2014. Best Practices for Publishing Linked Data. Working Group Note. World Wide Web Consortium (W3C). URL: https://www.w3.org/TR/ld-bp/.

[64] Iglesias, E., Jozashoori, S., Chaves-Fraga, D., Collarana, D., Vidal, M.E., 2020. SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs, in: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, ACM. doi:10.1145/3340531.3412881.

[65] Issa, S., Adekunle, O., Hamdi, F., Cherfi, S.S.S., Dumontier, M., Zaveri, A., 2021. Knowledge graph completeness: A systematic literature review. IEEE Access 9, 31322–31339. doi:10.1109/ACCESS.2021.3056622.

[66] Jiménez-Ruiz, E., Hassanzadeh, O., Efthymiou, V., Chen, J., Srinivas, K., Cutrona, V., 2020. Semtab 2020: Semantic web challenge on tabular data to knowledge graph matching 2020 URL: http://ceur-ws.org/Vol-2775/.

[67] Jiménez-Ruiz, E., Hassanzadeh, O., Srinivas, K., Efthymiou, V., Chen, J., 2019. Semtab 2019: Semantic web challenge on tabular data to knowledge graph matching URL: http://ceur-ws.org/Vol-2553/.

[68] Jiménez-Ruiz, E., Hassanzadeh, O., Srinivas, K., Efthymiou, V., Chen, J., 2021. Semtab 2021: Semantic web challenge on tabular data to knowledge graph matching URL: http://ceur-ws.org/Vol-2553/.

[69] Jiménez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I., Pinkel, C., Skjæveland, M.G., Thorstensen, E., Mora, J., 2015. Bootox: Practical mapping of rdbs to owl 2, in: Arenas, M., Corcho, O., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P., Dumontier, M., Heflin, J., Thirunarayan, K., Staab, S. (Eds.), The Semantic Web - ISWC 2015, Springer International Publishing, Cham. pp. 113–132.

[70] Jozashoori, S., Chaves-Fraga, D., Iglesias, E., Vidal, M.E., Corcho, O., 2020. Funmap: Efficient execution of functional mappings for knowledge graph creation, in: International Semantic Web Conference, Springer. pp. 276–293.

[71] Jozashoori, S., Sakor, A., Iglesias, E., Vidal, M.E., 2022. Eablock: A declarative entity alignment block for knowledge graph creation pipelines, in: Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, Association for Computing Machinery, New York, NY, USA. p. 1908–1916. URL: https://doi.org/10.1145/3477314.3507132, doi:10.1145/3477314.3507132.

[72] Jozashoori, S., Vidal, M.E., 2019. Mapsdi: A scaled-up semantic data integration framework for knowledge graph creation, in: Panetto, H., Debruyne, C., Hepp, M., Lewis, D., Ardagna, C.A., Meersman, R. (Eds.), On the Move to Meaningful Internet Systems: OTM 2019 Conferences, Springer International Publishing, Cham. pp. 58–75.

[73] Junior, A.C., Debruyne, C., Brennan, R., O'Sullivan, D., 2016. Funul: A method to incorporate functions into uplift mapping languages, in: Proceedings of the 18th International Conference on Information Integration and Web-Based Applications and Services, Association for Computing Machinery, New York, NY, USA. p. 267–275. URL: https://doi.org/10.1145/3011141.3011152, doi:10.1145/3011141.3011152.

[74] Kaebisch, S., Kamiya, T., McCool, M., Charpenay, V., Kovatsch, M., 2020. Web of Things (WoT) Thing Description. Working Group Recommendation. World Wide Web Consortium (W3C). URL: http://www.w3.org/TR/wot-thing-description/.

[75] Kalayci, E.G., Brandt, S., Calvanese, D., Ryzhikov, V., Xiao, G., Zakharyaschev, M., 2019. Ontology–based access to temporal data with ontop: A framework proposal. International Journal of Applied Mathematics and Computer Science 29, 17–30. URL: https://doi.org/10.2478/amcs-2019-0002, doi:doi:10.2478/amcs-2019-0002.

[76] Kharlamov, E., Jiménez-Ruiz, E., Zheleznyakov, D., Bilidas, D., Giese, M., Haase, P., Horrocks, I., Kllapi, H., Koubarakis, M., Özçep, Ö., Rodríguez-Muro, M., Rosati, R., Schmidt, M., Schlatte, R., Soylu, A., Waaler, A., 2013. Optique: Towards obda systems for industry, in: Cimiano, P., Fernández, M., Lopez, V., Schlobach, S., Völker, J. (Eds.), The Semantic Web: ESWC 2013 Satellite Events, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 125–140.

[77] Kitchenham, B., 2004a. Procedures for performing systematic reviews. Keele, UK, Keele University 33, 1–26.

[78] Kitchenham, B., 2004b. Procedures for performing systematic reviews. Keele, UK, Keele Univ. 33.

[79] Klyne, G., Carroll, J.J., 2004. Resource Description Framework (RDF): Concepts and Abstract Syntax. Recommendation. World Wide Web Consortium (W3C). URL: http://www.w3.org/TR/rdf-concepts/.

[80] Krishnan, A., 2018. Making search easier: How amazon's product graph is helping customers find products more easily. URL: https://blog.aboutamazon.com/innovation/making-search-easier.

[81] Kyzirakos, K., . GeoTriples: a Tool for Publishing Geospatial Data as RDF Graphs Using R2RML Mappings , 12.

[82] Kyzirakos, K., Karpathiotakis, M., Koubarakis, M., 2012. Strabon: A semantic geospatial dbms, in: International Semantic Web Conference, Springer. pp. 295–311.

[83] Kyzirakos, K., Savva, D., Vlachopoulos, I., Vasileiou, A., Karalis, N., Koubarakis, M., Manegold, S., 2018. GeoTriples: Transforming geospatial data into RDF graphs using R2RML and RML mappings. Journal of Web Semantics 52-53, 16–32. URL: https://www.sciencedirect.com/science/article/pii/S1570826818300428, doi:https://doi.org/10.1016/j.websem.2018.08.003.

[84] Le Phuoc, D., Dao-Tran, M., Le Tuan, A., Duc, M.N., Hauswirth, M., 2015. Rdf stream processing with cqels framework for real-time analysis, in: Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, Association for Computing Machinery, New York, NY, USA. p. 285–292. URL: https://doi.org/10.1145/2675743.2772586, doi:10.1145/2675743.2772586.

[85] Le-Phuoc, D., Nguyen-Mau, H.Q., Parreira, J.X., Hauswirth, M., 2012. A middleware framework for scalable management of linked streams. Journal of Web Semantics 16, 42–51. URL: https://www.sciencedirect.com/science/article/pii/S1570826812000728, doi:https://doi.org/10.1016/j.websem.2012.06.003. the Semantic Web Challenge 2011.

[86] Lefrançois, M., Zimmermann, A., Bakerally, N., 2017. A SPARQL extension for generating RDF from heterogeneous formats, in: The Semantic Web 14th International Conference,

ESWC 2017, Portorož, Slovenia, May 28 – June 1, 2017, Proceedings, Springer International Publishing, Portoroz, Slovenia. pp. 35–50. URL: http://www.maxime-lefrancois.info/docs/LefrancoisZimmermannBakerally-ESWC2017-Generate.pdf.

[87] Lefrançois, M., Zimmermann, A., Bakerally, N., 2017. Flexible rdf generation from rdf and heterogeneous data sources with sparql-generate, in: Ciancarini, P., Poggi, F., Horridge, M., Zhao, J., Groza, T., Suarez-Figueroa, M.C., d'Aquin, M., Presutti, V. (Eds.), Knowledge Engineering and Knowledge Management, Springer International Publishing, Cham. pp. 131–135.

[88] Lenzerini, M., 2002. Data integration: a theoretical perspective, in: PODS '02.

[89] Llaves, A., Corcho, Ó., Taylor, P., Taylor, K.L., 2016. Enabling rdf stream processing for sensor data management in the environmental domain. Int. J. Semantic Web Inf. Syst. 12, 1–21.

[90] Lopes, N., Bischof, S., Decker, S., Polleres, A., 2011. On the semantics of heterogeneous querying of relational, xml and rdf data with xsparql, in: Proceedings of the 15th Portuguese Conference on Artificial Intelligence (EPIA 2011), Lisbon, Portugal, Citeseer. pp. 10–13.

[91] Makinouchi, A., 1977. A consideration on normal form of not-necessarily-normalized relation in the relational data model., in: VLDB, Citeseer. pp. 447–453.

[92] Malhotra, A., Melton, J., Walsh, N., Kay, M., 2015. XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition). Recommendation. World Wide Web Consortium (W3C). URL: https://www.w3.org/TR/2010/REC-xpath-functions-20101214/.

[93] Mami, M.N., Graux, D., Scerri, S., Jabeen, H., Auer, S., Lehmann, J., 2019. Squerall: Virtual ontology-based access to heterogeneous and large data sources, Springer, Cham. pp. 229–245. doi:10.1007/978-3-030-30796-7_15.

[94] Marketakis, Y., Minadakis, N., Kondylakis, H., Konsolaki, K., Samaritakis, G., Theodoridou, M., Flouris, G., Doerr, M., 2017. X3ml mapping framework for information integration in cultural heritage and beyond. International Journal on Digital Libraries 18, 301–319.

[95] Maroy, W., Dimou, A., Kontokostas, D., De Meester, B., Verborgh, R., Lehmann, J., Mannens, E., Hellmann, S., 2017. Sustainable linked data generation: The case of DBpedia, Springer, Cham, Vienna, Austria. pp. 297–313. URL: http://jens-lehmann.org/files/2017/iswc_dbpedia_rml.pdf, doi:10.1007/978-3-319-68204-4_28.

[96] Mauri, A., Calbimonte, J.P., Dell'Aglio, D., Balduini, M., Brambilla, M., Della Valle, E., Aberer, K., 2016. TripleWave: Spreading RDF streams on the web, in: Groth, P., Simperl, E., Gray, A., Sabou, M., Krötzsch, M., Lecue, F., Flöck, F., Gil, Y. (Eds.), The Semantic Web – ISWC 2016, Springer International Publishing, Cham. pp. 140–149. doi:10.1007/978-3-319-46547-0_15.

[97] de Medeiros, L.F., Priyatna, F., Corcho, O., 2015. Mirror: Automatic r2rml mapping generation from relational databases, in: Cimiano, P., Frasincar, F., Houben, G.J., Schwabe, D. (Eds.), Engineering the Web in the Big Data Era, Springer International Publishing, Cham. pp. 326–343.

[98] Michel, F., Djimenou, L., Faron-Zucker, C., Montagnat, J., 2015. Translation of relational and non-relational databases into rdf with xr2rml. doi:10.5220/0005448304430454.

[99] Michel., F., Faron-Zucker., C., Montagnat., J., 2016. A generic mapping-based query translation from sparql to various target database query languages, in: Proceedings of the 12th International Conference on Web Information Systems and Technologies - Volume 2: WEBIST,, INSTICC. SciTePress. pp. 147–158. doi:10.5220/0005905401470158.

[100] Minadakis, N., Marketakis, Y., Kondylakis, H., Flouris, G., Theodoridou, M., Doerr, M., Jong, G., 2015. X3ml framework: An effective suite for supporting data mappings.

[101] Moher, D., Liberati, A., Tetzlaff, J., Altman, D., 2009. Preferred reporting items for systematic reviews and meta-analyses: the prisma statement. Br Med J 8, 336–341. doi:10.1371/joumal.pmedl000097.

[102] Mountantonakis, M., Tzitzikas, Y., 2019. Large-scale semantic integration of linked data: A survey. ACM Comput. Surv. 52. URL: https://doi.org/10.1145/3345551, doi:10.1145/3345551.

[103] Noy, N., Gao, Y., Jain, A., Narayanan, A., Patterson, A., Taylor, J., 2019. Industry-scale knowledge graphs: Lessons and challenges. Commun. ACM 62, 36–43. URL: https://doi.org/10.1145/3331166, doi:10.1145/3331166.

[104] Pankowski, T., Bąk, J., 2019. Dafo: An ontological database system with faceted queries, in: Hitzler, P., Kirrane, S., Hartig, O., de Boer, V., Vidal, M.E., Maleshkova, M., Schlobach, S., Hammar, K., Lasierra, N., Stadtmüller, S., Hose, K., Verborgh, R. (Eds.), The Semantic Web: ESWC 2019 Satellite Events, Springer International Publishing, Cham. pp. 152–155.

[105] Perry, M., Herring, J., 2012. OGC GeoSPARQL - A geographic query language for RDF data. Technical Report. URL: https://opengeospatial.github.io/ogc-geosparql/geosparql10/11-052r4_OGC_GeoSPARQL.pdf.

[106] Pittman, R.J., Srivastava, A., Hewavitharana, S., Kale, A., Mansour, S., 2017. Cracking the code on conversational commerce. URL: https://www.ebayinc.com/stories/news/cracking-the-code-on-conversational-commerce.

[107] Priyatna, F., Corcho, O., Sequeda, J., 2014. Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph, in: Proceedings of the 23rd international conference on World wide web, Association for Computing Machinery, Seoul, Korea. pp. 479–490.

[108] Prud'hommeaux, E., Boneva, I., Labra Gayo, J.E., Kellogg, G., 2018. Shape Expressions Language 2.1. Draft Community Group Report. World Wide Web Consortium (W3C). URL: http://shex.io/shex-semantics/.

[109] Prud'hommeaux, E., Seaborne, A., 2008. SPARQL Query Language for RDF. Recommendation. World Wide Web Consortium (W3C). URL: http://www.w3.org/TR/rdf-sparql-query/.

[110] Rahm, E., Do, H.H., 2000. Data cleaning: Problems and current approaches. IEEE Data Engineering Bulletin 23, 3–13.

[111] Raimond, Y., Ferne, T., Smethurst, M., Adams, G., 2014. The bbc world service archive prototype. Journal of Web Semantics 27-28, 2–9. URL: https://www.sciencedirect.com/science/article/pii/S1570826814000535, doi:https://doi.org/10.1016/j.websem.2014.07.005. semantic Web Challenge 2013.

[112] Rodríguez-Muro, M., Kontchakov, R., Zakharyaschev, M., 2013. Ontology-based data access: Ontop of databases, in: Alani, H., Kagal, L., Fokoue, A., Groth, P., Biemann, C., Parreira, J.X., Aroyo, L., Noy, N., Welty, C., Janowicz, K. (Eds.), The Semantic Web – ISWC 2013, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 558–573.

[113] Ryen, V., Soylu, A., Roman, D., 2022. Building semantic knowledge graphs from (semi-)structured data: A review. Future Internet 14. URL: https://www.mdpi.com/1999-5903/14/5/129.

[114] Santipantakis, G.M., Kotis, K.I., Vouros, G.A., Doulkeridis, C., 2018. RDF-Gen: Generating RDF from Streaming and Archival Data, in: Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics, pp. 1–10.

[115] Schultz, A., Matteini, A., Isele, R., Bizer, C., Becker, C., . Linked Data Integration Framework , 6.

[116] Scrocca, M., Comerio, M., Carenini, A., Celino, I., 2020. Turning Transport Data to Comply with EU Standards While Enabling a Multimodal Transport Knowledge Graph, in: International Semantic Web Conference, Springer. pp. 411–429.

[117] Shadbolt, N., O'Hara, K., 2013. Linked data in government. Internet Computing, IEEE 17, 72–77. doi:10.1109/MIC.2013.72.

[118] Shrivastava, S., 2017. Bring rich knowledge of people, places, things and local businesses to your apps. URL: https://blogs.bing.com/search-quality-insights/2017-07/bring-rich-knowledge-of-people-places-things-and-local-businesses-to-your-app

[119] Sicilia, A., Nemirovski, G., 2016. AutoMap4OBDA: Automated Generation of R2RML Mappings for OBDA, in: Knowledge Engineering and Knowledge Management, Springer International Publishing, Cham. pp. 577–592.

[120] Simsek, U., Kärle, E., Fensel, D., 2019. Rocketrml - a nodejs imple-

mentation of a use case specific rml mapper. ArXiv abs/1903.04969.

[121] Singhal, A., 2012. Introducing the knowledge graph: things, not strings. URL: https://www.blog.google/products/search/introducing-knowledge-graph-things-not.

[122] Slepicka, J., Yin, C., Szekely, P.A., Knoblock, C.A., 2015. Kr2rml: An alternative interpretation of r2rml for heterogenous sources., in: Proceedings of the 6th International Workshop on Consuming Linked Data (COLD 2015). URL: http://usc-isi-i2.github.io/papers/slepicka15-cold.pdf.

[123] Spanos, D.E., Stavrou, P., Mitrou, N., 2012. Bringing relational databases into the semantic web: A survey. Semantic Web 3, 169–209. doi:10.3233/SW-2011-0055.

[124] Sporny, M., Kellogg, G., Lanthaler, M., 2014. JSON-LD 1.0 – A JSON-based Serialization for Linked Data. Recommendation. World Wide Web Consortium (W3C). URL: http://www.w3.org/TR/json-ld/.

[125] Tamašauskaitė, G., Groth, P., 2022. Defining a knowledge graph development process through a systematic review. ACM Trans. Softw. Eng. Methodol. URL: https://doi.org/10.1145/3522586, doi:10.1145/3522586. just Accepted.

[126] Unbehauen, J., Martin, M., 2016. Executing sparql queries over mapped document store with sparqlmap-m, in: Proceedings of the 12th International Conference on Semantic Systems, Association for Computing Machinery, New York, NY, USA. p. 137–144. URL: https://doi.org/10.1145/2993318.2993326, doi:10.1145/2993318.2993326.

[127] Unbehauen, J., Stadler, C., Auer, S., 2013. Accessing relational data on the web with sparqlmap, in: Takeda, H., Qu, Y., Mizoguchi, R., Kitamura, Y. (Eds.), Semantic Technology, Springer Berlin Heidelberg, Berlin, Heidelberg. pp. 65–80.

[128] Van Assche, D., Haesendonck, G., De Mulder, G., Delva, T., Heyvaert, P., De Meester, B., Dimou, A., 2021a. Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation, in: Brambilla, M., Chbeir, R., Frasincar, F., Manolescu, I. (Eds.), Web Engineering, Springer International Publishing, Cham. pp. 337–352.

[129] Van Assche, D., Haesendonck, G., De Mulder, G., Delva, T., Heyvaert, P., De Meester, B., Dimou, A., 2021b. Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation, in: Brambilla, M., Chbeir, R., Frasincar, F., Manolescu, I. (Eds.), Web Engineering, Springer, Cham. pp. 337–352. doi:10.1007/978-3-030-74296-6_26.

[130] Vassalos, V., 2009. Answering Queries Using Views. Springer US, Boston, MA. pp. 92–98. URL: https://doi.org/10.1007/978-0-387-39940-9_849, doi:10.1007/978-0-387-39940-9_849.

[131] Verreydt, S., Yskout, K., Joosen, W., 2021. Security and privacy requirements for electronic consent: A systematic literature review. ACM Trans. Comput. Healthcare 2. URL: https://doi.org/10.1145/3433995, doi:10.1145/3433995.

[132] Vu, B., Pujara, J., Knoblock, C.A., 2019. D-repr: A language for describing and mapping diversely-structured data sources to rdf, in: Proceedings of the 10th International Conference on Knowledge Capture, Association for Computing Machinery, New York, NY, USA. p. 189–196. URL: https://doi.org/10.1145/3360901.3364449, doi:10.1145/3360901.3364449.

[133] Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyaschev, M., 2018. Ontology-based data access: A survey, IJCAI Organization.

[134] Xiao, G., Ding, L., Cogrel, B., Calvanese, D., 2019. Virtual Knowledge Graphs: An Overview of Systems and Use Cases. Data Intelligence 1, 201–223. URL: https://doi.org/10.1162/dint_a_00011, doi:10.1162/dint_a_00011, arXiv:https://direct.mit.edu/dint/article-pdf/1/3/201/683759/dint_a_00011.pdf.

[135] Xiao, G., Lanti, D., Kontchakov, R., Komla-Ebri, S., Güzel-Kalaycı, E., Ding, L., Corman, J., Cogrel, B., Calvanese, D., Botoeva, E., 2020. The virtual knowledge graph system ontop, in: Pan, J.Z., Tamma, V., d'Amato, C., Janowicz, K., Fu, B., Polleres, A., Seneviratne, O., Kagal, L. (Eds.), The Semantic Web – ISWC 2020, Springer International Publishing, Cham. pp. 259–277.

[136] Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., Auer, S., 2015. Quality assessment for linked data: A survey. Semantic Web 7, 63–93. doi:10.3233/SW-150175.

Dylan Van Assche is a PhD student at IDLab – Ghent University – imec in Belgium working on trade-offs of knowledge graph generation.
https://dylanvanassche.be

| Characteristic | Karma | Morph-RDB | Morph-xR2RML | TripleWave | GeoTriples | RMLMapper | RMLStreamer | RDF-Gen | MapSDI | RocketRML | SDM-RDFizer | D-REPR | FunMap | Chimera | SPARQL-Generate | SPARQL-Anything | ShExML |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **T1: Datasource** | | | | | | | | | | | | | | | | | |
| File | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| SQL RDB | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | | | ✓ | ✓ | ✓ | ✓ | | | |
| NoSQL DB | | | ✓ | | | | | | | | | ✓ | | | | | |
| Kafka stream | | | | | | | ✓ | | | | | | | | | | |
| WebSocket stream | | | | | | | | | | | | | | | ✓ | | |
| MQTT stream | | | | | | | ✓ | | | | | | | | ✓ | | |
| TCP stream | | | | | | | ✓ | | | | | | | | | | |
| SPARQL endpoint | | | | | | ✓ | | | | | | | | ✓ | | | |
| Web API | | | | | | ✓ | | | | | | | | | ✓ | | ✓ |
| Apache Hadoop | | | | | | | | | | | | | | | | | |
| **T2: Data format** | | | | | | | | | | | | | | | | | |
| CSV format | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| JSON format | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ |
| XML format | ✓ | | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ |
| TSV format | ✓ | | ✓ | | | ✓ | | | | | ✓ | | | ✓ | ✓ | | ✓ |
| ODS format | | | | | | ✓ | | | | | | ✓ | | | | | |
| XLSX format | | | | | | ✓ | | | | | | ✓ | | | | | |
| GeoJSON format | | | | | ✓ | | | | | | | | | | ✓ | | |
| KML format | | | | | ✓ | | | | | | | | | | | | |
| Shapefiles | | | | | ✓ | | | | | | | | | | | | |
| SQL query | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | | | ✓ | ✓ | ✓ | ✓ | | | |
| NoSQL query | | | ✓ | | | | | | | | | ✓ | | | | | |
| SPARQL query | | | | | | ✓ | | | | | | | | ✓ | | | |
| CBOR format | | | | | | | | | | | | | | | ✓ | | |
| HDT format | | | | | | | | | | | | | | | ✓ | | |
| RDF format | | | | | | | | | | | | | | | | ✓ | |
| plain text | | | | | | | | | | | | | | | ✓ | ✓ | |
| HTML format | | | ✓ | | | | | | | | | | | | | ✓ | |
| Archives | | | | | | | | | | | | | | | | ✓ | |
| EXIF encoded | | | | | | | | | | | | | | | | ✓ | |
| NetCDF | | | | | | | | | | | | ✓ | | | | ✓ | |
| **T3: Output Data** | | | | | | | | | | | | | | | | | |
| File | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Kafka stream | | | | | | | ✓ | | | | | | | | | | |
| WebSocket stream | | | | ✓ | | | ✓ | | | | | | | | | | |
| MQTT stream | | | | ✓ | | | ✓ | | | | | | | | | | |
| TCP stream | | | | | | | ✓ | | | | | | | | | | |
| SPARQL endpoint | | | | | | ✓ | ✓ | | | | | | | ✓ | | | |
| Web API | | | | | | ✓ | | | | | | | | | ✓ | | |
| **T4: Output Data formats** | | | | | | | | | | | | | | | | | |
| N-Triples | | ✓ | ✓ | | | ✓ | ✓ | ✓ | ⋆ | | ✓ | | ⋆ | ✓ | ✓ | ✓ | ✓ |
| N-Quads | | | | | | ✓ | ✓ | | ⋆ | ✓ | | | ⋆ | ✓ | ✓ | ✓ | ✓ |
| Turtle | | ✓ | ✓ | | ✓ | ✓ | | ✓ | ⋆ | | ✓ | ✓ | ⋆ | ✓ | ✓ | ✓ | ✓ |
| N3 | ✓ | ✓ | ✓ | | | ✓ | | | ⋆ | | | | ⋆ | ✓ | ✓ | ✓ | ✓ |
| RDF/XML | | ✓ | ✓ | | ✓ | ✓ | | | ⋆ | | | | ⋆ | ✓ | ✓ | ✓ | ✓ |
| RDF/JSON | | | | | | | | | ⋆ | | | | ⋆ | | ✓ | ✓ | ✓ |
| JSON-LD | ✓ | | ✓ | ✓ | | ✓ | ✓ | | ⋆ | ✓ | | | ⋆ | ✓ | ✓ | ✓ | ✓ |
| HDT | | | | | | ✓ | | | ⋆ | | | | ⋆ | ✓ | ✓ | | |
| TriX | | | | | | ✓ | | | ⋆ | | | | ⋆ | ✓ | ✓ | ✓ | ✓ |
| TriG | | | | | | ✓ | | | ⋆ | | | | ⋆ | ✓ | ✓ | ✓ | ✓ |
| D-REPR JSON | | | | | | | | | ⋆ | | | ✓ | ⋆ | | | | |

**Table 4**
System characteristics applied on the discussed materialization systems. Systems which do not provide their source code are excluded. Some systems (⋆) can be combined with others, thus support depends on its combination.

| Materialization implementation | T5 | T6 | T7 | T8 | T9 and T10 | T11 | T13a | T13b |
|---|---|---|---|---|---|---|---|---|
| Karma | KR2RML | Python code snippets | together, during schema transformation | Python | CLI | Apache-2.0 | None | ETL |
| Morph-RDB | R2RML | No | NA | Scala | CLI | Apache-2.0 | Self-join and subquery elimination | ETL |
| Morph-xR2RML | xR2RML | No | NA | Scala | CLI, SPARQL endpoint | Apache-2.0 | Inherited from Morph-RDB | ETL |
| TripleWave | R2RML | Hardcoded | together, pre-processing of input data | NodeJS | CLI | Apache-2.0 | Vertical scaling | Streaming |
| GeoTriples | RML with GeoTriples extensions | GeoSPARQL & stSPARQL functions | together, during schema transformation | Java | CLI, webapp | Apache-2.0 | None | ETL |
| RMLMapper | RML | FnO | together, during schema transformation | Java | CLI, library, Docker | MIT | None | ETL |
| RMLStreamer | RML | FnO | together, during schema transformation | Scala | CLI, Docker | MIT | Horizontal and vertical scaling | Streaming |
| MapSDI | RML | NA | NA | Python | CLI | Apache-2.0 | Removal of unused and duplicate data, relational algebra | Inherited from mapping rule processor |
| RocketRML | RML | FnO | together, during schema transformation | NodeJS | CLI, library, Docker, webapp | CC-BY-SA-4.0 | Multiple XML parsers | ETL |
| SDM-RDFizer | RML | No | NA | Python | CLI, library, Docker | Apache-2.0 | Removal of unused and duplicate data, relational algebra, memory optimizations | Batch processing |
| D-REPR | D-REPR | Python code snippets | together, pre-processing of input data | Python & Rust | CLI, webapp, Docker | MIT | Execution plans, core processing in native programming language | ETL |
| FunMap | RML | FnO | separately, before schema transformation | Python | CLI | Apache-2.0 | Function preprocessing, vertical scaling | Inherited from mapping rule processor |
| Chimera | RML | FnO | together, during schema transformation | Java | CLI | Apache-2.0 | Mapping rules without joins optimizations, vertical scaling | Batch processing |
| SPARQL-Generate | SPARQL-Generate | SPARQL Functions | together, during schema transformation | Java | CLI, library, webapp, Sublime Text integration | Apache-2.0 | None | ETL |
| SPARQL-Anything | Facade-X | SPARQL Functions | together, during schema transformation | Java | CLI, SPARQL endpoint | Apache-2.0 | None | ETL |
| ShExML | ShExML | No | NA | Scala | CLI, library, webapp | MIT | None | ETL |

**Table 5**
System characteristics applied on materialization implementations. Systems without available source code are excluded. Some systems have characteristics which does not apply to them, these characteristics are marked as 'Not Applicable' (NA).
T5: Schema transformation language, T6: Data transformation language, T7: Applicability, T8: Programming language, T9: Integration, T10: Interface, T11: License, T13a: Optimizations, T13b: Scaling

| Virtualization implementation | T5 | T6 | T7 | T8 | T9 and T10 | T11 | T13a | T13b |
|---|---|---|---|---|---|---|---|---|
| morph-streams++ | R2RML | No | NA | Scala | CLI | No license | Processing on DSMS engine | No |
| Ontop | R2RML or Ontop Mapping Language | No | NA | Java | CLI | Apache-2.0 | SQL & virtualization optimizations, aggregation on databases, ontological entailement | with Denodo, Dremio or Teiid |
| Morph-RDB | R2RML | No | NA | Scala | CLI, Docker | Apache-2.0 | SQL optimizations | No |
| Morph-xR2RML | xR2RML | No | NA | Scala | CLI, SPARQL end-point | Apache-2.0 | Inherited from Morph-RDB | No |
| SparqlMap-M | R2RML | R2RML custom extension | together, during schema transformation | Java | CLI, SPARQL end-point | No license | Query normalization & analysis, mapping binding, execute on database engine | No |
| Squerall | RML | FnO | together, during schema transformation | Java | CLI, GUI | Apache-2.0 | Spark or Presto processing, aggregation on data sources | with Spark or Presto |
| Ontario | RML | No | NA | Python | CLI | GPL-2.0 | Star shape subqueries on data sources, locally joined | SPARQL federation |
| Morph-RDB extension | $S_2O$ | No | NA | Scala | CLI, Docker | Apache-2.0 | Inherited from Morph-RDB | with SNEEql |
| XGSN | SSN ontology | Hardcoded in wrapper | together, pre-processing of input data | Java | CLI, we-bapp, API | GPL-3.0 | Access delegated to RDF stream processor | No |
| Obi-Wan | custom GLAV mapping | No | NA | Java | CLI, webapp | MIT | Ontology entailment | with Tatooine |

**Table 6**
System characteristics applied on the discussed virtualization implementations. Systems which do not provide their source code are excluded.
T5: Schema transformation language, T6: Data transformation language, T7: Applicability, T8: Programming language, T9: Integration, T10: Interface, T11: License, T13a: Features, T13b: Federation

| Characteristic | Morph-streams++ | Ontop | Morph-RDB | Morph-xR2RML | SparqlMap-M | Squerall | Ontario | Morph-RDB extension | Morph-GraphQL | Obi-Wan | XGSN |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **T1: Data source** | | | | | | | | | | | |
| File | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| SQL RDB | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| NoSQL DB | | | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | |
| OpenGIS | | ✓ | | | | | | | | | |
| Graph DB | | | | | | | ✓ | | | | |
| Triple Store | | | | | | | | | | ✓ | |
| Kafka stream | | | | | | | | | | | |
| WebSocket stream | | | | | | | | | | | |
| MQTT stream | | | ✓ | | | | | | | | |
| TCP stream | | | | | | | | | | | |
| UDP stream | | | | | | | | | | | ✓ |
| SPARQL endpoint | | | | | | | | | | | |
| Web API | | | ✓ | | | | | | | | |
| Apache Hadoop | | | | | | | ✓ | | | | |
| Esper | ✓ | | | | | | | | | | |
| GSN | ✓ | | | | | | | | | | ✓ |
| Apache Parquet | | | | | | ✓ | | | | | |
| Serial | | | | | | | | | | | ✓ |
| CoAP | | | | | | | | | | | ✓ |
| USB | | | | | | | | | | | ✓ |
| **T2: Data format** | | | | | | | | | | | |
| CSV format | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ |
| JSON format | | | | ✓ | | | | | | | |
| XML format | | | | ✓ | | | ✓ | | | | |
| TSV format | | | | | | | ✓ | | | | |
| ODS format | | | | | | | | | | | |
| XLSX format | | | | | | | | | | | |
| GeoJSON format | | | | | | | | | | | |
| KML format | | | | | | | | | | | |
| Shapefiles | | | | | | | | | | | |
| SQL query | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NoSQL query | | | | ✓ | ✓ | ✓ | ✓ | | | ✓ | |
| SPARQL query | | | | | | | ✓ | | | ✓ | |
| CBOR format | | | | | | | | | | | |
| HDT format | | | | | | | | | | | |
| RDF format | | | | | | | | | | | |
| plain text | | | | | | | | | | | |
| HTML format | | | | ✓ | | | | | | | |
| Archives | | | | | | | | | | | |
| Neo4J graph | | | | | | | ✓ | | | | |
| EXIF encoded | | | | | | | | | | | |
| RSS | | | | | | | | | | | ✓ |
| ESRI Grid ASCII | | | | | | | | | | | ✓ |
| Images | | | | | | | | | | | ✓ |
| **T3: Output Data & T4: Output Data formats** | | | | | | | | | | | |
| $SPARQL_{Stream}$ query result | ✓ | | | | | | | ✓ | | | |
| SPARQL query result | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ |
| CQELS query result | | | | | | | | | | | ✓ |
| GraphQL query result | | | | | | | | | ✓ | | |

**Table 7**
Data sources supported by the discussed virtualization systems. Systems which do not provide their source code are excluded.