typhon-rml: Modularised Declarative Knowledge Graph Construction for Flexible Integrations and Performance Optimisation

Marco Grassi*,†, Mario Scrocca*,†, Alessio Carenini and Irene Celino

Cefriel – Politecnico di Milano, Milan, Italy

Abstract

Adopting declarative approaches for constructing knowledge graphs enhances the maintainability and reusability of schemas and data transformations from diverse data sources. However, a fully declarative description requires the user to encode specific details of the data integration process within the mapping rules, including how to extract the input data from specific data sources and how to load the result into the target ones. This aspect significantly burdens the developers of mapping processors, who must adhere to the mapping language features to transform heterogeneous data formats to RDF, while also facilitating efficient access to various input and output data sources. Additionally, considering the user's point of view, a tightly coupled approach for the declaration and execution of the entire construction process can affect the flexibility of reusing mapping rules for different data integration scenarios. In this paper, we address these challenges and propose a method to modularise the declarative construction of knowledge graphs, allowing for a decoupled processing of input/output data sources and mapping rules. We introduce the `typhon-rml` library to demonstrate this approach. Focusing on RML mapping rules as input, we showcase how the tool facilitates their reuse and customisation for various integration requirements. A preliminary qualitative evaluation is conducted in the context of a relevant scenario for smart traffic management.

Keywords

Declarative Mappings, Knowledge Graph Construction, Mapping Languages

1. Introduction

A declarative approach for knowledge graph construction is based on a mapping language to specify the required schema and data transformations from various heterogeneous data sources [1] and a mapping processor capable of their execution. This approach, in contrast with ad-hoc data integration pipelines, enhances the maintainability and reusability of mapping rules. However, relying on a fully declarative approach for describing the entire ETL process (*extraction* from input data sources, *transformation* to RDF, *loading* to target data sources) can impose a growing challenge for developers of mapping processors who aim to adhere to a specific mapping language. As also emerged from the results of the latest implementation challenges organised within the Knowledge Graph Construction Workshop [2, 3] considering RML [4], there is great heterogeneity in mapping processors regarding not only the coverage of language features (i.e., modules of the RML specification covered) but also the types of data sources (e.g., local files, HTTPS services, relational databases, etc.) and formats (CSV, JSON, XML, etc.) supported. Indeed, a great challenge in implementing mapping processors is that they must not only support transformation capabilities defined by the mapping language specifications but also efficient access, parsing and querying of diverse input data sources and formats. Similarly, a declarative specification of the target data source possibly requires manipulation of the generated RDF (e.g., to obtain a different serialisation) and interaction with heterogeneous, and possibly remote, data sources

KGCW'25: 6th International Workshop on Knowledge Graph Construction, June 1st, 2025, Portorož, SLO

*Corresponding author.

†These authors contributed equally.

marco.grassi@cefriel.com (M. Grassi); mario.scrocca@cefriel.com (M. Scrocca); alessio.carenini@cefriel.com (A. Carenini); irene.celino@cefriel.com (I. Celino)

000-0003-3139-3049 (M. Grassi); 0000-0002-8235-7331 (M. Scrocca); 0000-0003-1948-807X (A. Carenini); 0000-0001-9962-7193 (I. Celino)

to store the data. These aspects pose a relevant challenge for developers of mapping processors since they involve selecting, integrating, and maintaining external libraries.

Additionally, from a user's perspective, there are potential drawbacks in a tightly coupled approach for knowledge graph construction that expects a mapping processor to support the complete execution of the ETL process defined by the declarative mapping language. The lack of modularisation makes it more difficult to efficiently apply the same declarative mappings across various data integration contexts if specific features are not supported by the selected mapping language or mapping processor.

This paper addresses these issues and investigates how a modularised approach for declarative knowledge graph construction can enable a more flexible development of data integration pipelines and their performance optimisation. The `typhon-rml` tool is introduced to propose and demonstrate an approach enabling the compilation of the input RML mapping rules to an intermediate representation that decouples extract-load operations from the rules specifying the data and schema transformations. We discuss how this method could support the reuse and customisation of a declarative mapping document to meet heterogeneous data integration requirements.

The remainder of the paper is organised as follows. Section 2 introduces relevant concepts and discusses the challenges considering a concrete scenario in the context of smart traffic management. Section 3 describes the designed approach and how this is implemented by the `typhon-rml` tool. Section 4 provides a preliminary evaluation of the tool, discussing its application to the scenario previously introduced. Finally, Section 5 draws the conclusions and presents the future work.

2. Preliminaries and Challenges Addressed

Alongside the declarative mapping rules to generate RDF triples, an RML [4] document specifies the heterogeneous data sources to be considered as input. In an RML mapping, a `rml:Source` describes the data sources to be accessed and a `rml:LogicalSource` how to extract data from them. From these, the RML document specifies the declarative mapping rules to be executed on a given `rml:LogicalSource` via a set of `rml:TriplesMaps`. These two aspects are tightly coupled, but these steps are logically independent and do not necessarily need to be intertwined.

The W3C Knowledge Graph Construction Community Group is currently working on the RML specification to improve the decoupling between RML Core and the dedicated RML IO module[1], e.g., by limiting the test cases associated with RML Core only to JSON inputs read from local files. Moreover, the introduction of an RML IO-Registry[2] guarantees that mapping processors can implement support only for specific data sources and data formats. However, despite the additional modularisation, an RML mapping processor aiming to fully comply with the specification should support the complete data integration process. The rmlmapper-java[3] implementation presented by [5] showcases how the need for supporting different input and output data sources required the addition of many external dependencies that increase the size of the library and, over time, may affect the technical debt associated with the mapping processor. Moreover, incorporating the input and output data access within the mapping processor limits users' ability to adapt and select technologies that may be better suited for a specific mapping scenario. As shown in [6] for RML processors but also for other KG construction engines [7], it is extremely important to consider that the libraries used to access, parse and query the input data sources may heavily impact the performances of the overall knowledge graph construction process.

Concerning the reusability of mapping rules, a certain degree of flexibility is needed for deploying the same mappings supporting different data integration pipelines [8, 9]. The need for encoding within the mapping rules how to access data from a specific data source may complicate its reuse for other data sources that expose data in the same format and may apply the same rules for KG construction. For example, if a RML mapping processor only processes inputs from files, and the input data is pro-

[1] https://w3id.org/rml/io/spec
[2] https://w3id.org/rml/rml-io-registry/
[3] https://github.com/RMLio/rmlmapper-java

vided via a message broker, it may be necessary at runtime to consume the data, save it to a file with a specific temporary name and refer to the specific file within the mapping rules. Additionally, the integration within the mapping processor of input/output operations hinders the opportunity to optimise the execution of mapping rules for specific mapping scenarios. While it is true that different mapping processors may provide a better solution in terms of performance in different cases, users often stick with selecting a single mapping processor due, for example, to the choice of a programming language of preference. As a result, the user is limited to the optimisations and features implemented by the selected mapping processor. The decoupled approach proposed in this work aims to enable the potential customisation of the data integration process to be executed given a set of RML mapping rules. By adopting `typhon-rml`, the user can support any customisation allowed by Java libraries. However, we foresee the potential implementation of a similar approach in other programming languages.

To explain better the challenges addressed, we introduce a motivating example from a use case within the SmartEdge European project[4] for real-time traffic data applications in Helsinki [10]. The use case aims to leverage a common interoperable representation of mobility data collected from a swarm of nodes (e.g., sensors/vehicles) that should collaborate at a given road intersection. To enable the conversion of incoming data to a shared RDF representation, a set of RML mapping rules has been developed to support a declarative mapping approach. In this paper, we focus on converting data about radar observations of the incoming traffic at a given intersection. Each observation provides a categorization of the type of vehicle detected by the radar and related metrics such as the speed measured and the estimated position. While the development of the mapping rules happened by using local samples of radar data in JSON format, bringing the defined mapping rules into a production deployment poses the following challenges:

- ***C1*** *Applying the same mapping rules to an input data source adopting the same data format but requiring different access mechanisms, e.g., in terms of protocol (HTTPS, WebSocket, etc.) or interaction paradigm (pull/push, frequency, pagination, etc.).* The *Logical Source* defined to access radar samples from a file should be modified to support the specification of data access from a WebSocket, but the mapping processor does not support this type of input source;

- ***C2*** *Deploying the same mapping rules for multiple data sources.* The same mapping rules should be deployed in different *Road Side Units* (RSU) at each intersection and the *Logical Source* description should be changed for each deployment according to the parameters of the relevant radar;

- ***C3*** *Forwarding the mapping output to specific target data sources.* The processed radar data should be sent to different NATS[5] queues for downstream processing, but the mapping processor does not support this type of target source;

- ***C4*** *Enable custom performance optimisation dependent on the input data being considered.* Adding a *TriplesMap* to describe the sensor would cause the generation of the same triple multiple times (i.e., one time for each observation received by the same radar in a single message). This would affect performance even in the case of automatic duplication removal by the mapping processor. Moreover, checks for non-null fields and IRI validity are performed for every triple generated in the generic case because no assumptions can be made about the incoming data by the mapping processor.

- ***C5*** *Customising the KG construction process to support specific requirements for data integration.* The mapping process should be instrumented to collect metrics and traces from each RSU on the number of messages processed and the conversion latency to RDF. Moreover, the processed data should be stored in a CSV format for historical data collection.

In our previous work, we introduced Chimera[6] [11] to configure composable semantic transformation pipelines for KG construction. Chimera provides a modular and configurable set of building

[4] https://www.smart-edge.eu/2023/07/25/use-case-preventing-rear-end-collisions-by-enhancing-road-intersection-safety/
[5] https://nats.io/
[6] https://github.com/cefriel/chimera

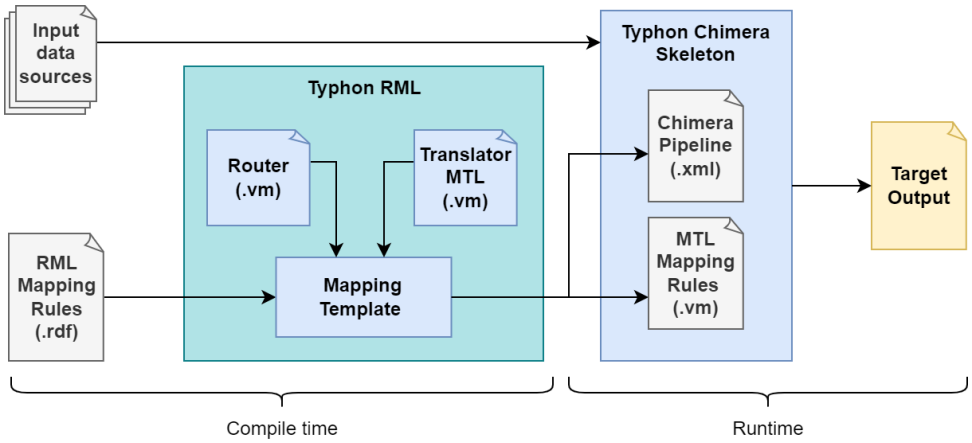

Figure 1: Overview of the `typhon-rml` approach for modularising the declarative mapping rules.

blocks to construct and manipulate RDF graphs within a data integration pipeline defined through the Apache Camel[7] integration framework. One of Chimera's objectives is to decouple the execution of mapping rules for KG construction from access to input/output data sources. The underlying idea of the `typhon-rml` tool presented in this paper is to automatize the generation of Chimera pipelines from existing declarative mapping rules expressed via RML. Such an approach facilitates leveraging already existing adapters provided by the Apache Camel ecosystem while enabling the execution of RML mapping rules in Chimera by a more thin mapping processor focusing on applying transformations.

Additionally, in the paper [12], we proposed an approach for generic knowledge conversions between data formats derived from declarative approaches for KG construction like RML. The `mapping-template` tool[8] implements this approach and leverages a template engine to describe mapping rules according to the Mapping Template Language (MTL) syntax. The `mapping-template` can process RML mapping rules by converting them to a template-based representation using MTL, thus offering the possibility of introducing specific performance optimisations to the mapping rules before executing them (e.g., reducing duplicated triples generation considering specific assumptions on the input data). The `typhon-rml` tool explicitly generates an MTL version of the input RML mapping rules to support their customisation by the user before deployment.

As a related work, the RMLWeaver-JS [13] mapping processor also leverages an intermediate representation obtained from RML at compile time. In this case, a mapping plan consisting of algebraic mapping operators is used for optimising the execution of the mapping rules.

3. Design and Implementation

The designed approach aims at modularizing the knowledge graph construction process, and a first implementation is made available via the `typhon-rml` tool leveraging the Chimera framework for implementing it. The `typhon-rml` tool is available at https://github.com/cefriel/typhon-rml.

The underlying idea of the approach is that starting from an RML mapping, we can separate the data access and the execution of mapping rules on the accessed data into distinct parts. A dedicated component can be executed at *compile time* to transform the RML mapping rules into artefacts that can be customised, if needed, to address different integration scenarios and to introduce performance optimisations. At *runtime*, the generated artefacts are executed to perform the correct KG construction process described declaratively in the RML file.

The `typhon-rml` component is a Java-based solution that implements the process shown in Figure 1. The `typhon-rml` tool gets as input an RML mapping, and produces: (i) a Chimera data pipeline

[7] https://camel.apache.org/
[8] https://github.com/cefriel/mapping-template

(route.xml) to read the needed input data sources, execute the mapping rules on them and finally store or send off this mapping results to a given output data source, and (ii) the mapping rules needed to perform this mapping operation.

Chimera is used to implement the data pipeline that uses the appropriate Camel component needed to access the data source specified in the RML and write the results to the right target(s). The mapping execution part is also executed via Chimera through the camel-chimera-mapping-template component, which integrates the mapping-template library into Chimera. The pipeline is generated from RML as an Apache Camel route specified declaratively via the Camel DSL in XML. Additionally, a set of mapping rules using MTL (template.vm) are generated to execute the same transformations defined by the input RML. At *runtime*, the typhon-chimera-skeleton component is provided to execute the data pipeline and the mappings generated, producing an output that is equivalent to the one that would have been produced by the RML mapping.

Notably, we decided to apply the mapping-template, our own solution for generic knowledge conversion between different data formats, to transform the input RML mapping rules at *compile time*. Both generated files are produced by applying declarative MTL mappings on the input RML file. In the former case, a router.vm[9] file is used to produce an XML serialized Chimera pipeline while, in the latter, a translator.vm[10] MTL file is used to produce the RML equivalent MTL file. The router.vm MTL mapping generates a Chimera pipeline composed of multiple Apache Camel routes. Firstly, through a SPARQL query the distinct rml:Source in the RML file are extracted alongside their corresponding rml:LogicalSource. This information is used to configure the Chimera pipeline that accesses data by using the relevant Apache Camel components[11]. For instance, if the RML mapping declares two distinct sources, two different routes are generated by typhon-rml to access both data sources. In the generate Chimera pipeline, all accessed data are then forwarded to the camel-chimera-mapping-template component, which applies the template.vm generated mapping. In this first version, we only support the default RML behaviour of saving the generated KG to a local file.

The benefit of this approach is that the generated Chimera route.xml and template.vm MTL mappings are not executed automatically. Instead, there is an additional stage allowing the user to modify both files. These modifications might be necessary to adapt the KG construction as described in the motivating example and the identified related challenges (Section 2). The Chimera route.xml pipeline can be modified by changing from where data is read, as long as this is done by using an available Apache Camel component. For example, this might mean reading data relying on a specific binary protocol instead of reading data from a local file. Similarly, the MTL mapping can be manually changed to introduce performance optimisations not explicitly definable in RML. An example is an optimised join condition that relies on external knowledge of the input data sources that can not be expressed in RML.

To be executed, the generated Chimera pipeline and accompanying MTL mappings are processed by the typhon-chimera-skeleton component. This component is a Java template project with the needed dependencies to run Chimera pipelines. The typhon-chimera-skeleton project can manually be modified to include additional Java dependencies needed for Chimera route customizations.

4. Preliminary Evaluation

We discuss a qualitative evaluation of typhon-rml by applying it to the motivating example introduced in Section 2. All the artefacts mentioned in this section are made available in the tool repository.

Each radar in the considered environment records the position of detected vehicles and their estimated speed alongside a timestamp for when the observation was made. In the RML mapping shown in Listing 1, the SOSA ontology [14] is used to define each radar as a sosa:Sensor, which generates sosa:Observations related to the vehicle that the radar is detecting. The detected vehicle is classified according to the type of vehicle detected and as a sosa:FeatureOfInterest to which the measured

[9] https://github.com/cefriel/typhon-rml/blob/main/typhon-rml/src/main/resources/router.vm
[10] https://github.com/cefriel/typhon-rml/blob/main/typhon-rml/src/main/resources/typhon-rml-compiler.vm
[11] https://camel.apache.org/components/4.10.x/index.html

properties of position and speed are attached. The RML mapping assumes that the observations are available as a local JSON file called `data.json`.

This RML mapping is used as input for `typhon-rml`, which produces the XML Chimera pipeline shown in Listing 2 and the MTL mapping equivalent to the input RML mapping. The mapping illustrates the typical iterative process of mapping development, where representative data from a data source is stored locally and used for mapping development on that sample. However, to deploy the mappings in a production environment like the one of our example, the data should be fetched from the data source itself and the same mapping rules should be applied to different radars in each RSU. For RML, this means having a copy of the development mapping and changing the *LogicalSource* to reflect the actual data source, while checking that the selected mapping processor supports it. Following the `typhon-rml` approach, the pipeline can also be configured to rely on external libraries not directly supported by the mapping processor, and the MTL mapping remains unchanged.

Listing 1: Declaration of an RML *LogicalSource* in an RML mapping

```
@prefix rml: <http://w3id.org/rml/> .
@prefix sosa: <http://www.w3.org/ns/sosa/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.org/ns#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .

<http://example.com/base/LogicalSource> a rml:LogicalSource;
  rml:iterator "$[*]";
  rml:referenceFormulation rml:JSONPath;
  rml:source [ a rml:RelativePathSource;
    rml:root rml:MappingDirectory;
    rml:path "data.json"
  ] .

<http://example.com/base/TriplesMap1> a rml:TriplesMap;
  rml:logicalSource <http://example.com/base/LogicalSource>;

  rml:subjectMap [
    rml:template "http://example.org/observation/
       {message_id}/{timestamp}" ;
    rml:class sosa:Observation
  ];

  rml:predicateObjectMap [
    rml:predicate sosa:hasFeatureOfInterest ;
    rml:objectMap [
      rml:template "http://example.org/location/{lat}/{lon}" ;
      rml:class sosa:FeatureOfInterest
    ]
  ];

[...]
```

Listing 2: Corresponding `typhon-rml` generated Chimera pipeline

```
<?xml version="1.0" encoding="utf-8"?>
<beans>
  <bean name="template" type="com.cefriel.util.ChimeraResourceBean">
    <properties>
      <property key="url" value="file://./data/template.vm"/>
      <property key="serializationFormat" value="vtl"/>
    </properties>
  </bean>

  <route id="zbiffgddfja">
    <from uri="file:./?fileName=data.json&amp;noop=true"/>
    <convertBodyTo type="java.lang.String"/>
    <setVariable name="readerFormat">json</setVariable>
    <setVariable name="readerName">zbiffgddfja</setVariable>
    <to uri="direct:aggregate"/>
  </route>
  <route id="readersAggregation">
    <from uri="direct:aggregate"/>
    <aggregate
      aggregationStrategy="#class:com.cefriel.ReadersAggregation"
      completionSize="1">
      <correlationExpression>
        <constant>true</constant>
      </correlationExpression>
      <to uri="mapt://readers?template=#bean:template"/>
      <to uri="graph://get?rdfFormat=nquads&amp;defaultGraph=true"/>
      <to uri="graph://dump?dumpFormat=nquads"/>
      <to uri="file://?fileName=output.nq"/>
      <process ref="#class:com.cefriel.ShutDownProcessor"/>
    </aggregate>
  </route>
</beans>
```

The identified challenge ***C1*** is thus addressed by the `typhon-rml` because the user can edit the generated Chimera pipeline to read data from a WebSocket connection instead of reading from a local file. To achieve this, the route where the data is being read can be modified as shown in the modified route example[12]. Additionally, this manual intervention also solves challenge ***C2*** because the radar to be considered by the Chimera route is specified in the WebSocket connection string and can be possibly read via environment variables. The challenge ***C3*** is also similarly addressed as the NATS target can be specified as a step of the Chimera pipeline where the mapped data will be sent. This can be done by adding the Camel NATS component as the last step of pipeline as shown in the modified route example[13]. These examples show how the adoption of the Apache Camel framework simplifies the customisation of the pipelines for additional requirements by leveraging already existing components. Nevertheless, also custom components can be developed and integrated within a pipeline.

Additionally, the generated MTL file may be leveraged to customise the transformations to be executed and improve the overall performance. In the considered example, we can remove checks for non-null values and IRI validity since we know the specific format of the messages received. Moreover,

[12] https://github.com/cefriel/typhon-rml/blob/bf4c513fde22c4b26c4218c21ce279caa95e755d/typhon-rml/example/route-modified.xml#L11

[13] https://github.com/cefriel/typhon-rml/blob/bf4c513fde22c4b26c4218c21ce279caa95e755d/typhon-rml/example/route-modified.xml#L33

we may generate RDF triples describing a radar as a *sosa:Sensor* performing a set of observations only for distinct values in the same message (***C4***). The considered examples have minor performance impact, however, similar optimisations may have a greater effect, considering, for instance, cases that involve joins between different data sources and that can be optimised in the MTL file (cf. the *shapes.txt* case in the GTFS-Madrid Benchmark [15, 16]).

Finally, considering the challenge ***C5***, the user may adapt the pipeline to insert a monitoring component in the Chimera pipeline which can be used to observe and manage the KG construction process of each RSU. Moreover, the generated pipeline can be modified according to more specific requirements for data integration. To obtain historical data for CSV collection, the MTL file generated from the RML specification can be modified to generate a CSV output[14] and obtain a ready-to-be-deployed artefact for this requirement. Similarly, the mappings can be customised to cover features RML mapping processor of choice (e.g., access to specific input data sources).

The current design of `typhon-rml` aims to support also the processing within Chimera pipeline of RML mapping rules without requiring a conversion to MTL and by leveraging directly an RML mapping processor. However, this approach requires modifications to the mapping engine to support the execution of mapping rules on *Logical Sources* provided dynamically. A related discussion is happening within the W3C Community Group for Knowledge Graph Construction[15] and could enable support for `typhon-rml` considering existing RML mapping processors.

Currently, the `mapping-template` provides coverage for the RML Core specification as demonstrated by its execution against the test cases for the KGCW Challenge 2024. The `typhon-rml` supports decoupled execution of the same set of RML Core test cases, thus including reading from CSV, JSON, XML local files and from MySQL and Postgres databases. The configuration to run the tool and the generated Chimera route and MTL file for each test case are made available in the online repository[16].

5. Conclusions

The declarative knowledge graph construction process can benefit from a decoupled approach separating input and output operations from the execution of the mapping rules. In this paper, we highlighted the challenges of a fully declarative and integrated approach, considering a motivating example for smart traffic management. The `typhon-rml` tool is proposed to implement a decoupled approach leveraging Chimera and the `mapping-template` tool to enable flexible development and performance optimisation of RML mapping rules. The advantages of such an approach have been discussed in the context of the proposed motivating example, showcasing how the same mapping rules can be easily modified to address different integration requirements and adapted to optimise data access according to the specific mapping scenario considered.

As future work, we plan to extend the compatibility of `typhon-rml` according to the test cases defined by the RML-IO specification and considering the RML-IO Registry. Moreover, we foresee a similar solution to support the definition of Chimera pipelines not only considering RML mapping rules but also from metadata describing a data source within a data catalogue (e.g., using DCAT). In this case, RML may simply refer to a data source, and `typhon-rml` may fetch its metadata to compose the correct pipeline to access the data source. Finally, we believe that the challenges highlighted and discussed in this paper may be relevant to the Knowledge Graph Construction community, opening the door to additional research contributions to enable a more decoupled management of input/output operations for existing mapping processors.

[14] https://github.com/cefriel/typhon-rml/blob/651f7609d04f36bafb9a2b94cbb4777663438b1a/example/template-modified.vm#L14

[15] cf. https://github.com/kg-construct/rml-io-registry/issues/10

[16] cf. https://github.com/cefriel/typhon-rml/tree/main/evaluation

Acknowledgments

The presented research was partially supported by the SmartEdge project, funded under the Horizon Europe RIA Research and Innovation Programme (Grant Agreement 101092908).

Declaration on Generative AI

During the preparation of this work, the author(s) used Grammarly to improve grammar, check spelling, and reword. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester, A. Dimou, Declarative RDF graph generation from heterogeneous (semi-)structured data: A systematic literature review, Web Semant. 75 (2023). doi:10.1016/j.websem.2022.100753.

[2] D. Chaves-Fraga, A. Dimou, A. Iglesias-Molina, U. Serles, D. V. Assche, Preface, in: Proceedings of the 4th International Workshop on Knowledge Graph Construction co-located with 20th Extended Semantic Web Conference, volume 3471 of *CEUR Workshop Proceedings*, CEUR, Hersonissos, Greece, 2023. URL: https://ceur-ws.org/Vol-3471/#preface, iSSN: 1613-0073.

[3] D. Chaves-Fraga, A. Dimou, A. Iglesias-Molina, U. Serles, D. V. Assche, Preface, in: Proceedings of the 5th International Workshop on Knowledge Graph Construction, volume 3718 of *CEUR Workshop Proceedings*, CEUR, Hersonissos, Greece, 2024. URL: https://ceur-ws.org/Vol-3718/#preface, iSSN: 1613-0073.

[4] A. Iglesias-Molina, D. Van Assche, J. Arenas-Guerrero, B. De Meester, C. Debruyne, S. Jozashoori, P. Maria, F. Michel, D. Chaves-Fraga, A. Dimou, The RML Ontology: A Community-Driven Modular Redesign After a Decade of Experience in Mapping Heterogeneous Data to RDF, in: T. R. Payne, V. Presutti, G. Qi, M. Poveda-Villalón, G. Stoilos, L. Hollink, Z. Kaoudi, G. Cheng, J. Li (Eds.), The Semantic Web – ISWC 2023, Lecture Notes in Computer Science, Springer Nature Switzerland, Cham, 2023, pp. 152–175. doi:10.1007/978-3-031-47243-5_9.

[5] D. Van Assche, G. Haesendonck, G. De Mulder, T. Delva, P. Heyvaert, B. De Meester, A. Dimou, Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation, in: Web Engineering, Springer International Publishing, Cham, 2021, pp. 337–352.

[6] M. Scrocca, A. Carenini, M. Comerio, I. Celino, Semantic Conversion of Transport Data Adopting Declarative Mappings: An Evaluation of Performance and Scalability, in: D. Chaves-Fraga, P. Colpaert, M. Sadeghi, M. Scrocca, M. Comerio (Eds.), Proceedings of the 3rd International Workshop Semantics And The Web For Transport, volume 2939 of *CEUR Workshop Proceedings*, CEUR, Online, September, 2021. URL: https://ceur-ws.org/Vol-2939/#paper2, iSSN: 1613-0073.

[7] H. García-González, Optimising the ShExML engine through code profiling: From turtle's pace to state-of-the-art performance, Semantic Web 16 (2025) SW–243736. URL: https://journals.sagepub.com/doi/abs/10.3233/SW-243736. doi:10.3233/SW-243736.

[8] L. Tailhardat, Y. Chabot, R. Troncy, Designing NORIA: a Knowledge Graph-based Platform for Anomaly Detection and Incident Management in ICT Systems, in: D. Chaves-Fraga, A. Dimou, A. Iglesias-Molina, U. Serles, D. V. Assche (Eds.), Proceedings of the 4th International Workshop on Knowledge Graph Construction co-located with 20th Extended Semantic Web Conference, volume 3471 of *CEUR Workshop Proceedings*, CEUR, Hersonissos, Greece, 2023. URL: https://ceur-ws.org/Vol-3471/paper3.pdf, iSSN: 1613-0073.

[9] D. Anicic, et al., SmartEdge project Deliverable D3.1 – Design of Tools for Continuous Semantic Integration, 2023. URL: https://www.smart-edge.eu/deliverables/.

[10] I. Kosonen, K. Koskinen, J. Kostiainen, Real-time traffic data applications in the mobility lab of helsinki-case smart junction, in: ITS World Congress, 2021.

[11] M. Grassi, M. Scrocca, A. Carenini, M. Comerio, I. Celino, Composable Semantic Data Transformation Pipelines with Chimera, in: D. Chaves-Fraga, A. Dimou, A. Iglesias-Molina, U. Serles, D. V. Assche (Eds.), Proceedings of the 4th International Workshop on Knowledge Graph Construction co-located with 20th Extended Semantic Web Conference, volume 3471 of *CEUR Workshop Proceedings*, CEUR, Hersonissos, Greece, 2023. URL: https://ceur-ws.org/Vol-3471/paper9.pdf, iSSN: 1613-0073.

[12] M. Scrocca, A. Carenini, M. Grassi, M. Comerio, I. Celino, Not everybody speaks RDF: Knowledge conversion between different data representations, in: Fifth International Workshop on Knowledge Graph Construction@ ESWC2024, 2024.

[13] S. M. Oo, T. Verbeken, B. D. Meester, RMLWeaver-JS: An algebraic mapping engine in the KGCW Challenge 2024, in: Proceedings of the 5th International Workshop on Knowledge Graph Construction, volume 3718 of *CEUR Workshop Proceedings*, CEUR, Hersonissos, Greece, 2024. URL: https://ceur-ws.org/Vol-3718/paper8.pdf, iSSN: 1613-0073.

[14] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, M. Lefrançois, SOSA: A lightweight ontology for sensors, observations, samples, and actuators, Web Semant. 56 (2019) 1–10. URL: https://doi.org/10.1016/j.websem.2018.06.003. doi:10.1016/j.websem.2018.06.003.

[15] D. Chaves-Fraga, F. Priyatna, A. Cimmino, J. Toledo, E. Ruckhaus, O. Corcho, GTFS-Madrid-Bench: A benchmark for virtual knowledge graph access in the transport domain, Journal of Web Semantics 65 (2020) 100596. doi:10.1016/j.websem.2020.100596.

[16] J. Arenas-Guerrero, M. Scrocca, A. Iglesias-Molina, J. Toledo, L. Pozo-Gilo, D. Doña, Ó. Corcho, D. Chaves-Fraga, Knowledge graph construction with R2RML and RML: an ETL system-based overview, in: D. Chaves-Fraga, A. Dimou, P. Heyvaert, F. Priyatna, J. F. Sequeda (Eds.), Proceedings of the 2nd International Workshop on Knowledge Graph Construction co-located with 18th Extended Semantic Web Conference (ESWC 2021), Online, June 6, 2021, volume 2873 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021. URL: http://ceur-ws.org/Vol-2873/paper11.pdf.