A Window into the Multiple Views of Linked Data

Sitt Min Oo[1][0000−0001−9157−7507]

IDLab, Department of Electronics and Information Systems,
Ghent University – imec, Technologiepark-Zwijnaarde 122, 9052 Ghent, Belgium
x.sittminoo@ugent.be

Abstract. RDF mapping engines enable access to existing heterogeneous data sources as RDF Knowledge Graph (KG). However, these mapping engines have two challenges: i) processing streaming data sources with changing velocity efficiently, ii) and providing a rich variety in the format of the generated KG output. To tackle these challenges, I carry out my research in 3 steps. I will first design a highly scalable data stream mapping solution to handle dynamic velocity of streaming data sources. Preliminary results indicate that our stream mapping solution outperforms state of the art engines with lower latency, constant memory usage, and higher throughput. I will then refine this architecture in a task-based fashion, aiming to be a common architecture for any kind of mapping. Finally, I will utilize the common modular mapping architecture and extend it with a component to derive an intermediate representation of the data mapping process, enabling heterogeneous to heterogeneous data mapping. The combined solution will provide a highly scalable heterogeneous to heterogeneous data stream mapping engine, enabling us to have multiple views of the underlying KG.

Keywords: RML · Knowledge Graph Generation · Mapping Engine.

1 Introduction

Knowledge Graph (KG) adoption is further intensified by technologies such as Solid [20], which decouples data from applications. This decoupling requires integrating heterogeneous data from different organizations and applications, which is seamlessly achieved with KGs [11].

KG data is represented in various formats such as those compliant with the Resource Definition Framework (RDF). These formats are required to facilitate data sharing in the form of a decentralized database. To use data from multiple heterogeneous data sources, mapping engines could be used to convert existing data to RDF data in a shape as required by the different applications.

However, current mapping engines are incapable of dealing with the following two characteristics of data: i) the **velocity** of streaming data sources, ii) and the **variety** in the KG output of these engines. The mapping engines cannot handle the dynamic velocity (i.e., variable data velocity with respect to time)

of streaming data sources efficiently. Furthermore, the mapping engines are designed to generate RDF data adhering to a single shape from heterogeneous data sources; there is no variation in the output. Finally, research on mapping engines is slowed down as they have redundant implementations due to the lack of a common, reusable, and modular components.

In this PhD thesis, I aim to provide a scalable heterogeneous to heterogeneous mapping engine. This is done in 3 steps: i) an RDF mapping solution to handle dynamic velocity, ii) a common modular architecture for mapping engines, and iii) an intermediate data mapping representation that can be used to achieve full heterogeneous to heterogeneous mapping.

2 State of the Art

After discussing mapping languages (Section 2.1) and mapping engines (Section 2.2) for mapping heterogeneous data to RDF data, I will elaborate and discuss optimization techniques employed for batch processing (Section 2.3).

2.1 Languages for mapping heterogeneous data

Several mapping languages exist for mapping heterogeneous data into RDF data [18]. Depending on the extensibility of the mapping language, data processing operations such as joins and data transformations are supported.

Languages such as R2RML [5], and its extensions such as RML [7] use Turtle syntax to write mapping rules. Extensions such as alignment to FnO [6] and Logical Target [19] are applied to RML to enable data transformations on the input, and describe how and where the output should be generated respectively.

Languages such as SPARQL-Generate [15] and Facade-X [4] extend SPARQL, while others such as ShExML [9] are based on ShEx. SPARQL-Generate and Facade-X rely on the underlying SPARQL engine for joins and data transformations. ShExML uses custom definitions to describe how to iterate over the data sources, process them, and join the iterated data items.

2.2 Mapping engines

Mapping engines generate RDF data and store it in a specific document or (triple) store [18]. The existing generation approaches could be further categorized into two groups based on its processing type: i) *batch processing* and ii) *stream processing*.

Batch processing engines work with inputs based on the assumption that the data is finite and bounded. RML's reference implementation RMLMapper[1] is one example of a batch processing mapping engine.

Stream processing engines have to work with data which are potentially *unbounded* and *infinite in size* [14]. For example, a temperature sensor monitoring a building's ambient temperature will keep generating data as long as

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

they are powered. Mapping engines handling streaming data sources include SPARQL-Generate [15], Chimera [2], and RMLStreamer [10].

2.3 Optimization techniques

In recent years, optimization of the mapping processes with batch processing has been the focus of research in KG generation [18], to either speed up the execution time, lower the memory footprint, or remove duplicates from the output.

Morph-KGC is a batch processing implementation that focuses on the *optimization of the mapping rules* in the mapping document through *partitioning* [1]. SDM-RDFizer employs *specialized data structures* to eliminate duplicates and empty values, and optimize joins [12].

The application of arbitrary functions during the mapping process – e.g. via FnO [6] – also presents opportunities for optimization as they bring processing overhead. FunMap [13] reduces the aforementioned processing overhead through eager execution of function rules, storing the function-applied results in an intermediate dataset before proceeding with the mapping rules.

The aforementioned optimization approaches are implemented in monolithic engines without the capability to integrate different optimizations from each other due to the **lack of a modular architecture**.

To the best of our knowledge, stream processing optimizations techniques are not actively researched unlike batch processing optimization techniques for KG generation. Approaches such as SPARQL-Generate [15] process streaming input data, without detailing generalizable optimizations.

3 Problem Statements and Contributions

I will focus on the two major drawbacks in the current state of the art mapping engines: i) inefficient handling of streaming data with dynamic **velocity**, and ii) lack of **variety** in the output data serialization by only serializing in RDF.

On the one hand, the velocity of streaming data can vary over time. For example, a temperature sensor might have an emission rate of 1 Hz under default conditions which can increase to 100 Hz in an event of a fire disaster to provide more accurate measurements. If there are multiple sensors feeding the data to the mapping engine, this can become the bottleneck of the mapping process if the engines are not scalable, leading to significantly increased latency. Thus, mapping engines handling streaming data sources must be able to deal with such a burst in data velocity.

On the other hand, there are numerous mapping solutions which output only *RDF* serializations, leading to a lack of variety in data being generated. In the context of web applications, current web applications utilizing a KG must communicate in RDF compliant formats with the servers hosting the KG. A mapping solution from heterogeneous to *heterogeneous* data would increase KG adoption, for example, by acting as a data translation layer between the server and the clients.

3.1 Inefficient handling of streaming heterogeneous data

The first problem I identified is the mapping engines' inability to handle streaming heterogeneous data efficiently. Mapping engines, working with data streams, must take into account that the data stream can change in velocity over time. This might have a performance impact if these engines cannot handle a sudden and large change in velocity. Complexity increases if the mapping engines have to join multiple data streams of different velocity. This leads us to our first Research Question (RQ).

RQ 1: What is an efficient architecture for mapping heterogeneous data streams which change in velocity over time, especially when joining data streams of different velocity in a distributed and parallel environment?

H: A task-based granular architecture with the ability to join two data streams of different velocity using a dynamic windowing algorithm would enable efficient mapping of heterogeneous data streams to RDF in terms of latency, memory, and CPU usage.

Contribution: A parallel and scalable stream processing architecture for mapping heterogeneous data to RDF data, which is able to efficiently process multiple data streams, with changing velocity, and also bounded datasets.

3.2 Lack of variety in output

Current state of the art stream mapping engines only map from heterogeneous data to RDF data. This leads to the problem that the underlying KG can only be viewed in a specific RDF shape: supporting multiple views is currently only possible by setting up multiple parallel mapping processes, inhibiting potential optimization opportunities. It can be serialized in multiple ways (JSON-LD, Turtle, etc.), but the shape remains the same. We do not have the ability to construct multiple views of the underlying KG for the client. Combined with Problem 3.1, this leads us to the following RQ:

RQ 2: How can we scale heterogeneous to heterogeneous stream mapping engines to support the complete decoupling of data and applications?

H: We can extend existing heterogeneous to RDF data mapping engines by modularizing them into granular operators, and devising an intermediate representation for the mapping process. The intermediate representation makes the mapping process independent from any source or target formats and languages.

Contribution: A modular heterogeneous to heterogeneous streaming data mapping engine. Using this engine, we can support generating multiple views simultaneously on top of the same source data. Not only would we be able to generate multiple RDF views, but also non-RDF views, achieving full heterogeneous to heterogeneous mapping.

I broke down RQ 2 into following subproblems with respective sub RQs. The combined solution from the subproblems provides the answer to heterogeneous to heterogeneous mapping.

Subproblem 1: Modular architecture There is no modular mapping architecture from which existing research is built upon. Hence, existing mapping engines have redundant implementations and their optimizations can not be easily integrated with each other. Furthermore, it is difficult to integrate state of the art mapping approaches and optimizations in existing mapping engines due to the complex and engine-specific architecture they are presented in.

RQ 2.1: Which components do we need to modularize the materialization process of mapping heterogeneous data to RDF data sufficiently to support existing optimization approaches?

H: Through generalizing the optimization approaches for the mapping engines, we find the set of components which enable mapping engines to be configured with granular optimizations and thus modularizing the materialization process of mapping heterogeneous data to RDF data.

Contribution: A modular heterogeneous to RDF mapping architecture that efficiently processes data streams by integrating existing optimizations.

Subproblem 2: Heterogeneous to heterogeneous mapping. Mapping from heterogeneous to heterogeneous data requires metadata on the target data structure and data format.

Current mapping engines derive the target RDF data structure from the mapping document defined in the mapping language of their choice. However, it is currently unknown how to incorporate the desired output *heterogeneous* data format and structure.

RQ 2.2: How do we incorporate the desired heterogeneous data structure in the mapping process to enable heterogeneous to heterogeneous data mapping?

H: The modular architecture, from Sub Problem 1, extended with a component to derive the intermediate representation of the data mapping process, has all necessary information on the output data structure to enable mapping from heterogeneous data to heterogeneous data.

Contribution: An intermediate representation to describe the mapping process from heterogeneous to heterogeneous data.

4 Research Methodology and Approach

Stream mapping engine I conduct an in depth study of state of the art mapping engines and their mapping processes. The mapping processes – broken down into clearly defined tasks – are used in my architecture design as granular task-based components. I review data processing paradigms and best practices to design the streaming architecture, and put additional focus on the component that joins multiple data sources with dynamic velocity. Existing stream joining techniques utilizing windows (a temporary buffer of an incoming data stream) are studied to choose the window type on which the dynamic window is built upon. I study congestion control techniques from networking as possible references to implement the dynamically adjusting window. The dynamically adjusting

window enables us to deal with sudden changes in data velocity, and mitigate the resulting *congestion* effects such as high latency and low throughput.

Modular mapping architecture I conduct a systematic survey of optimization techniques in mapping engines to analyze the impact of individual optimization techniques. For individual analysis of optimizations, I extend the aforementioned modular architecture to enable isolation of every optimization techniques, and investigate their impact on the different stages of the mapping workflow. Through the comparison of the different combinations of optimization techniques, I want to investigate if having a modular architecture for mapping engine results in a similar or better performance than state of the art mapping engines, with significant improvements in flexibility.

Heterogeneous to heterogeneous mapping I extend the previously developed modular architecture to map heterogeneous to heterogeneous data. I investigate the execution graph of the modular architecture and state of the art techniques for Intermediate Representation (IR) generation. From the execution graph of the modular architecture, I will apply IR generation techniques for data mapping. The generated IR will contain all necessary metadata on the data structures and formats for mapping from heterogeneous to heterogeneous data.

5 Evaluation

Stream mapping engine To evaluate my stream mapping engine, I need to consider the context in which the stream mapping engine is expected to be deployed. The stream mapping engine is located at the boundary of two domains: traditional stream processing for the data stream input that it consumes, and RDF stream processing for the output RDF stream that it produces. Thus, benchmark approaches from the domain of traditional stream processing and RDF stream processing are combined: i) the benchmark architecture from RSPLab [17], ii) the workload design to emulate dynamic streaming characteristics from Open Stream Processing Benchmark [8], and iii) the measurement strategies from Karimov et al [14]. As input for the evaluation, I will use real-world (logged) sensor data. This ensures that the data characteristics reflect with real-world data expected to be processed by a stream mapping engine.

The following metrics are measured to evaluate the performance of the stream mapping engine:

a) **Event-time latency (ms)** is measured to avoid the effect of coordinated occlusion [14], where queuing time is ignored, by also taking into account the queuing time of the records inside the engine. The output generation latency should be kept low if the stream processing engines are to process real-time data streams and match the velocity of the data stream.
b) **Throughput (records/s)** with increasing data velocity. I aim to find the *sustainable throughput* [14] which is the highest throughput an engine could sustain without increasing latency due to back-pressure.

c) **CPU and the peak memory usage** to evaluate the efficiency of the engine resource usage while dealing with varying input data stream velocity.

These are the core metrics I will use throughout my PhD to evaluate the subsequent implementations, since they are common metrics to evaluate stream processing engine.

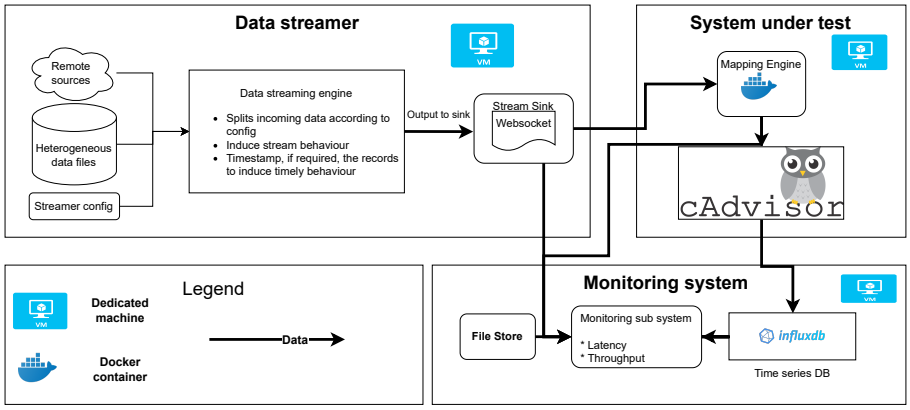

Fig. 1. Benchmark architecture to evaluate the different engines, inspired by RSPLab.

The benchmark architecture is inspired by RSPLab [17] with a modification to include a custom data streaming component (Figure 1). I isolate the different benchmark components (e.g. the measurement component from the System Under Test (SUT)) to reduce the influence of the benchmark components on the engines during the evaluation process.

The dataset for the evaluation is the time annotated traffic sensor data from the Netherlands provided by NDW (Nationale Databank Wegverkeersgegevens) [2], which is also used in Open Stream Processing Benchmark [8]. The traffic sensor data set has high duplicates, time characteristics and relationship between multiple sensors. This enables us to evaluate the mapping engine derived from the modular architecture for duplicate removal, ability to deal with data velocity through replaying the time series dataset, and joins or even functions to transform relationships between the different sensors.

Modular mapping architecture A combination of current state of the art optimization techniques, implemented using the modular architecture, will be evaluated using the same evaluation benchmark architecture as above. This allows me to find the best combination based on the aforementioned evaluation metrics, and also to analyze the impact of individual optimization techniques.

The implementation will be evaluated for performance against the state of the art monolithic engines with the same feature set. The performance of the

[2] NDW: http://www.ndw.nu

modular architecture should be similar to the target engine but very flexible to be extended with extra optimization components.

Heterogeneous to heterogeneous mapping For the heterogeneous to heterogeneous mapping contribution, I will evaluate the reference implementation of intermediate representation for correctness.

For the dataset, I will use heterogeneous datasets based on real-life data. Heterogeneous datasets of different formats (such as CSV, JSON, and XML) will be generated by GTFS-Madrid Benchmark [3], which derives the dataset from GTFS data files of Madrid's subway network. To my knowledge, it is currently the only generator that is capable of generating heterogeneous data for the evaluation of mapping engines.

For the correctness evaluation, the generated heterogeneous data will be used as input for our reference implementation. The output data format of the mapping engine will be different from that of the input data format (e.g. mapping from XML to JSON). The output data will be compared to check if it has the same data structure and shape as the original shape and structure of the heterogeneous data generated by GTFS-Madrid Benchmark, with the same data format as the output data.

6 Intermediate Results

I developed RMLStreamer-SISO: a streaming mapping engine published at ISWC 2022 [3]. The paper, "RMLStreamer-SISO: an RDF stream generator from heterogeneous data" [16], introduces RMLStreamer-SISO as a scalable stream mapping engine that is able to efficiently process data streams with varying velocity.

To develop RMLStreamer-SISO's architecture, I considered the dataflow architecture paradigm of existing stream processing engines, such as Apache Flink [4], with a task-based architecture and incorporated it in our stream mapping engine. To enable joining of multiple streaming data sources with dynamic velocity, I took inspiration from congestion control algorithms such as additive-increase, multiplicative decrease algorithm of TCP congestion control. This enabled RMLStremaer-SISO to be capable of joining multiple heterogeneous data streams with dynamic velocity by relying on a dynamic windowing algorithm, where the window changes its size based on the incoming data velocities.

I showed that the approach of a task-based architecture for mapping streaming data sources is scalable in terms of the volume, increasing velocity, and dynamic velocity of the input data streams. It outperforms state of the art streaming mapping engines by achieving millisecond latency, constant memory usage for all workloads, and sustainable throughput of around 70,000 records/s [16]. This answers our RQ 1 on efficient mapping of streaming heterogeneous data to RDF data, and confirmed our hypothesis.

The task-based architecture from RMLStreamer-SISO is the starting point for developing a modular mapping architecture, in which multiple optimizations

[3] RMLStreamer-SISO: https://github.com/RMLio/RMLStreamer/releases/tag/v2.3.0
[4] Apache Flink:https://flink.apache.org/

can be independently evaluated. I identified following most common components of the mapping process through studying the architecture of the state of the art mapping engines: a) Mapping language interpretation, b) Source iterator, c) Windowing, d) Transformation, e) RDF Mapping, and f) Serialization.

I also identified different optimization techniques employed by state of the art mapping engines. These techniques have been categorized according to the different mapping stages they are applied on: a) Language interpretation stage, b) Pre-mapping stage right before the data records are mapped, and c) Mapping stage while the data records are being mapped.

7 Conclusion and Lessons Learned

This thesis contributes to the KG generation community by enabling highly scalable and efficient heterogeneous to heterogeneous streaming data mapping. Preliminary results already show that having a task-based architecture enables an efficient and scalable stream mapping engine. This architecture has the potential to be extended to be modular where users could integrate the only specific stages of the mapping process to create their custom mapping engine.

Current optimization research is slowed down due to overlapping feature implementation even though the focus of the research is on just a specific stage of the mapping process. Researchers would greatly benefit from a modular architecture where they could build their own mapping engine by utilizing the common components of the modular architecture. The benefits are: a) a faster research speed on mapping engines, b) higher quality research by allowing researchers to focus only on the components that matter, and c) fairer comparisons during the evaluation of optimization approaches based on a modular common architecture.

On the industry side, integrating this engine into KG ecosystems such as Solid can increase its uptake, as KG application developers can keep using existing data format standards when communicating with KG servers, without needing to dedicate time to learn RDF graph technologies.

References

1. Arenas-Guerrero, J., Chaves-Fraga, D., Toledo, J., Pérez, M.S., Corcho, O.: Morph-KGC: Scalable knowledge graph materialization with mapping partitions. Semantic Web pp. 1–20 (Aug 2022). https://doi.org/10.3233/sw-223135
2. Belcao, M., Falzone, E., Bionda, E., Valle, E.D.: Chimera: A Bridge Between Big Data Analytics and Semantic Technologies. In: The Semantic Web – ISWC 2021. pp. 463–479 (2021)
3. Chaves-Fraga, D., Priyatna, F., Cimmino, A., Toledo, J., Ruckhaus, E., Corcho, O.: Gtfs-madrid-bench: A benchmark for virtual knowledge graph access in the transport domain. Journal of Web Semantics **65**, 100596 (2020)
4. Daga, E., Asprino, L., Mulholland, P., Gangemi, A.: Facade-X: An Opinionated Approach to SPARQL Anything. In: Further with Knowledge Graphs – 17th International Conference on Semantic Systems. vol. 53, pp. 58–73 (2021)

5. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. Working group recommendation, World Wide Web Consortium (W3C) (2012), http://www.w3.org/TR/r2rml/
6. De Meester, B., Seymoens, T., Dimou, A., Verborgh, R.: Implementation-independent Function Reuse. Future Generation Computer Systems **110**, 946–959 (2020). https://doi.org/10.1016/j.future.2019.10.006
7. Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In: 7th Workshop on Linked Data on the Web. vol. 1184 (2014)
8. van Dongen, G., Van den Poel, D.: Evaluation of stream processing frameworks. IEEE Transactions on Parallel and Distributed Systems **31**(8), 1845–1858 (2020). https://doi.org/10.1109/TPDS.2020.2978480
9. García-González, H., Boneva, I., Staworko, S., Labra-Gayo, J.E., Lovelle, J.M.C.: ShExML: improving the usability of heterogeneous data mapping languages for first-time users. PeerJ Computer Science **6** (2020)
10. Haesendonck, G., Maroy, W., Heyvaert, P., Verborgh, R., Dimou, A.: Parallel RDF generation from heterogeneous big data. In: International Workshop on Semantic Big Data. No. 1 (2019). https://doi.org/10.1145/3323878.3325802
11. Hogan, A., et al.: Knowledge Graphs. ACM Computing Surveys **54**(4), 1–37 (2021). https://doi.org/10.1145/3447772
12. Iglesias, E., Vidal, M., Jozashoori, S., Collarana, D., Chaves-Fraga, D.: Empowering the SDM-RDFizer Tool for Scaling Up to Complex Knowledge Graph Creation Pipelines. Semantic Web Journal (2022)
13. Jozashoori, S., Chaves-Fraga, D., Iglesias, E., Vidal, M.E., Corcho, O.: Funmap: Efficient execution of functional mappings for knowledge graph creation. In: International Semantic Web Conference. pp. 276–293 (2020)
14. Karimov, J., Rabl, T., Katsifodimos, A., Samarev, R., Heiskanen, H., Markl, V.: Benchmarking distributed stream data processing systems. IEEE 34th International Conference on Data Engineering (ICDE) (2018). https://doi.org/10.1109/icde.2018.00169
15. Lefrançois, M., Zimmermann, A., Bakerally, N.: A SPARQL extension for generating RDF from heterogeneous formats. In: The Semantic Web 14th International Conference, ESWC. pp. 35–50 (2017). https://doi.org/10.1007/978-3-319-58068-5_3
16. Sitt Min Oo, Haesendonck, G., De Meester, B., Dimou, A.: RMLStreamer-SISO: An RDF Stream Generator from Streaming Heterogeneous Data. In: The Semantic Web – ISWC 2022. pp. 697–713 (2022). https://doi.org/10.1007/978-3-031-19433-7_40
17. Tommasini, R., Della Valle, E., Mauri, A., Brambilla, M.: Rsplab: Rdf stream processing benchmarking made easy. In: The Semantic Web – ISWC 2017. pp. 202–209 (2017). https://doi.org/10.1007/978-3-319-68204-4_21
18. Van Assche, D., Delva, T., Haesendonck, G., Heyvaert, P., De Meester, B., Dimou, A.: Declarative RDF graph generation from heterogeneous (semi-)structured data: A systematic literature review. Journal of Web Semantics (2022). https://doi.org/10.1016/j.websem.2022.100753
19. Van Assche, D., Haesendonck, G., De Mulder, G., Delva, T., Heyvaert, P., De Meester, B., Dimou, A.: Leveraging Web of Things W3C Recommendations for Knowledge Graphs Generation. pp. 337–352 (2021). https://doi.org/10.1007/978-3-030-74296-6_26
20. Verstraete, M., Verbrugge, S., Colle, D.: Solid: Enabler of decentralized, digital platforms ecosystems (2022)