

Implementing the EHRI-KG: From Declarative Mapping Rules to a Three-Layered KG Construction Architecture

Herminio García-González^{1,*}, Mike Bryant²

¹Kazerne Dossin, Goswin de Stassartstraat 153, 2800 Mechelen, Belgium

²NIOD Institute for War, Holocaust and Genocide Studies, Herengracht 380, 1016 Cj Amsterdam, The Netherlands

Abstract

Conducting transnational research with primary source material relating to the Holocaust can be a complex endeavour, mainly due to the fragmentation and dispersal of much important material across different holding institutions and jurisdictions. The EHRI Portal acts as an aggregator of archival descriptions with the aim of lowering the barriers to the discovery of and access to source material. Building on earlier work, we set out to offer this data in a format more amenable to semantic technologies, as well as aligned with the ICA's new Records in Contexts standard for archival description. In this paper, we explore the EHRI Knowledge Graph three-layered architecture (batch conversion, validation and continuous update) and reflect on the challenges encountered and the solutions involved, to finally establish a discussion on further needs and future points of development within the Knowledge Graph Construction community and its toolset. We believe that this work can be helpful for other practitioners seeking to implement similar architectures, while serving as a use case for future work which will allow users to further streamline their developments and reduce the overall implementation time.

Keywords

Knowledge Graph construction, architecture, declarative mapping rules, EHRI, ShExML

1. Introduction

Archival material and the primary sources it represents are a key part of the historical research process and its preservation, classification, and cataloguing is fundamental to ensuring access to our collective memory. In the field of Holocaust studies and Holocaust historiography, however, the intentional destruction of evidence and the massive dislocation of people during and after the Second World War introduced particular challenges for researchers, due to the fragmentation and scattering of many important sources throughout widely dispersed collection-holding institutions. To meet this challenge, the European Holocaust Research Infrastructure (EHRI) undertook the creation of the EHRI Portal¹, an aggregator of Holocaust-relevant archival descriptions which seeks to serve as an entrypoint for Holocaust researchers [1]. In a recent project, we explored the possibility of aligning the data in the EHRI Portal to a format more amenable to the Semantic Web standards [2], adopting the (then draft) version of the Records in Contexts Ontology (RiC-O) [3] and seeking to improve the general FAIRness of the data [4]. Since then, the International Council on Archives (ICA) Expert Group on Archival Description (EGAD)² has released the first stable version of RiC-O³ and EHRI has launched the EHRI-KG project⁴ with the aim of consolidating and improving the Linked Open Data (LOD) transformation infrastructure into a production-ready service.

In this paper we present the data mapping architecture and workflow alongside some interesting challenges and the solutions we have derived. While the process heavily relies on declarative mapping rules [5] (due to their generally favourable maintainability), many orchestration components and *ad-hoc*

KGCW'26: 7th International Workshop on Knowledge Graph Construction, May 10th, 2026, Dubrovnik, Croatia

*Corresponding author.

✉ herminio.garciagonzalez@kazernedossin.eu (H. García-González); m.bryant@niod.knaw.nl (M. Bryant)

🌐 <https://herminio.garcia.com/> (H. García-González); <https://www.niod.nl/en/staff/mikebryant> (M. Bryant)

🆔 0000-0001-5590-4857 (H. García-González); 0000-0003-0765-7390 (M. Bryant)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

¹<https://portal.ehri-project.eu/>

²<https://www.ica.org/ica-network/expert-groups/egad/>

³<https://github.com/ICA-EGAD/RiC-O/releases>

⁴<https://ehri-knowledge-graph.ehri-project.eu/>

solutions are still required. We believe that this architecture can inform other practitioners facing similar situations while, at the same time, prompting discussions around new required features in the Knowledge Graph Construction (KGC) community.

The rest of this paper is structured as follows: Section 2 describes the related work, in Section 3 we introduce the EHRI Portal and its technical stack, followed by a description of the conversion architecture in Section 4; Section 5 discusses the additions required to the declarative mapping technology as a motivation for their adoption and implementation within the state-of-the-art solutions; finally, in Sections 6 and 7 respectively we outline our future plans and draw some conclusions from this work.

2. Related Work

Knowledge graphs are being increasingly adopted in fields such as cultural heritage and historical research [6], life sciences [7] or industrial applications [8], and their semantics, enhanced interoperability, and the capacity to represent complex relations in a graph structure have been demonstrated to be particularly effective in solving a variety of challenges.

The creation and maintenance of KGs requires the combination and harmonisation of various different techniques, leading to the generation of methodologies formalising some of these approaches, along with tools to tackle laborious aspects. Two prominent examples in the field are ontology and KG engineering. However, some studies have recently claimed that a more comprehensive and extensive methodology is necessary in order to ensure a more harmonised and homogeneous KG lifecycle [9, 10].

While many works describe the creation of KGs and their architectures, only a few provide sufficient detail for establishing a comparison with the current approach. In [11] the authors report on the creation of a KG for the discovery of interactions between drugs used for COVID-19 treatment, created using a combination of RDF Mapping Language (RML) [12] rules and Natural Processing Language (NLP) techniques. RML is also used by [13] to construct a KG for a public procurement data space whose execution is orchestrated by Apache Airflow⁵. Similarly, in [14] Apache Nifi⁶ is utilised to manage the workflow for creating a commercial KG based on RML mappings. In order to offer a KG for the European Railway, in [15] YARRRML [16] rules (converted to their RML counterparts) were employed though no further details are given around the followed workflow. Also on the transport domain, the authors of [17] produced a multimodal KG using Chimera [18], a framework to implement data transformation pipelines covering the graph construction (lifting), transformation, validation and exploitation (lowering) phases. In [19] the construction of KGs for research-performing organisations is discussed for which YARRRML was employed through a workflow that ensured the creation of the ontology and the declarative mapping rules, the validity of the latter against the ontology and the database model, and the final KG materialisation. The authors elaborate further on some of the lessons learnt like the need for simple tools (like YARRRML with its compact syntax) or the need for separating the mapping rules by ontology classes (similar to the process that we describe in Section 4).

As a result, we can extrapolate that the creation of such KGs relies on some well-established tools and techniques (like RML and its myriad of conformant engines), yet some parts of the process are still developed in an *ad-hoc* manner. It is worth noting though that the authors of [10] have recently published the results of a survey⁷ seeking to validate the proposed KG lifecycle methodology through the coverage of each of the defined activities by 31 ongoing projects dealing with the creation of KGs [20], helping to shed some light on commonalities and best practices.

In the Holocaust domain, several initiatives are seeking to provide access to Holocaust data as LOD through different thematic, regional, and institutional scopes. For example, Netwerk Oorlogsbronnen⁸ covers WWII Dutch sources whereas Archiefpunt⁹ enables access to archival descriptions in Flanders and Brussels but without any further topical filtering. Similarly, some institutions dedicated to the

⁵<https://airflow.apache.org/>

⁶<https://nifi.apache.org/>

⁷<https://lot.linkedata.es/LOT4KG/survey.html>

⁸<https://www.oorlogsbronnen.nl/>

⁹<https://archiefpunt.be/>

Holocaust also offer their data – or part thereof – in Semantic Web formats, like CDEC [21] which offers a KG of Italian Holocaust Victims or the Resistance in Belgium¹⁰ developed by CegeSoma/State Archives of Belgium on top of a Wikibase instance [22, 23]. Nevertheless, a broader implementation of semantic technologies in this field is still a work in progress with a very unbalanced distribution across regions and topics. As such, the EHRI-KG is, to the best of our knowledge, the first transnational effort to bring Holocaust archival descriptions together in a unified KG.

3. Preliminaries

The EHRI Portal started its development in 2012 with an architecture based on a Labelled Property Graph (LPG). Reasons for this included its ability to simplify the representation and querying of hierarchically-structured data, along with ease of evolving the data model over time [1]. The dearth of semantic ontologies well-suited to the archival problem space, and a lack of expertise in the consortium, factored into this decision as well. Recent advancements in KGC technologies, however, and their use in the new ICA’s archival standard, motivated us to explore the publication of EHRI’s archival metadata as LOD.

From this standpoint, and taking into account that a wholesale reworking of the EHRI Portal’s persistence layer would neither be practical nor affordable, we searched for a method that would allow the project to maintain its existing functionality with the current architecture, while providing a new method for accessing, querying, and linking its data. The solution was realised in the EHRI-KG¹¹, a live replica of the EHRI Portal based on semantic standards and fully queryable by third-party users via a SPARQL endpoint¹². This solution not only solves our requirement of offering the data as LOD – and aligning with the new ICA’s RiC-O standard – but also enables users to perform arbitrary queries without impacting our main platform.

While this solution may seem simple enough, the difficulties lay, as many times in this field, in the volume and variety of the data which, as described later, determined many of the technical decisions. The EHRI Portal holds, at the time of writing, more than 380,000 archival descriptions held by 2,318 institutions located in 61 countries which are enriched by a set of additional controlled vocabularies and links, all of them following existing archival standards (e.g., ISAD(G), ISAAR, ISDIAH, etc.). Despite it not being large enough to be considered Big Data [24], this amount of data, and in many cases its lack of normalisation, already imposed many organisational and technical challenges which are further described in Section 4 alongside the final architecture.

4. General architecture

At a high level, the architecture is divided into three main processes: a batch conversion workflow, a validation phase, and a module for handling update operations. Figure 1 provides a graphical representation of this three-layered architecture.

4.1. Batch conversion

The batch conversion layer is at the core of the conversion process. In this layer, the data is downloaded from the EHRI Portal, converted to RDF and merged into a single big Turtle file which will be finally delivered to a triple store for stable persistence and serving other applications (through a SPARQL endpoint). Presently, Apache Jena Fuseki¹³ is used, offering a good coverage of the SPARQL 1.1 specification and enabling us to build a text-search index¹⁴. Nevertheless, the process is designed to be agnostic to the triple store technology, giving us the freedom to switch platforms depending on our specific requirements or even support different query methods (e.g., Linked Data Fragments [25]).

¹⁰<https://data.arch.be/?lang=en>

¹¹<https://lod.ehri-project-test.eu/>

¹²<https://lod.ehri-project-test.eu/query/>

¹³<https://jena.apache.org/documentation/fuseki2/index.html>

¹⁴<https://jena.apache.org/documentation/query/text-query.html>

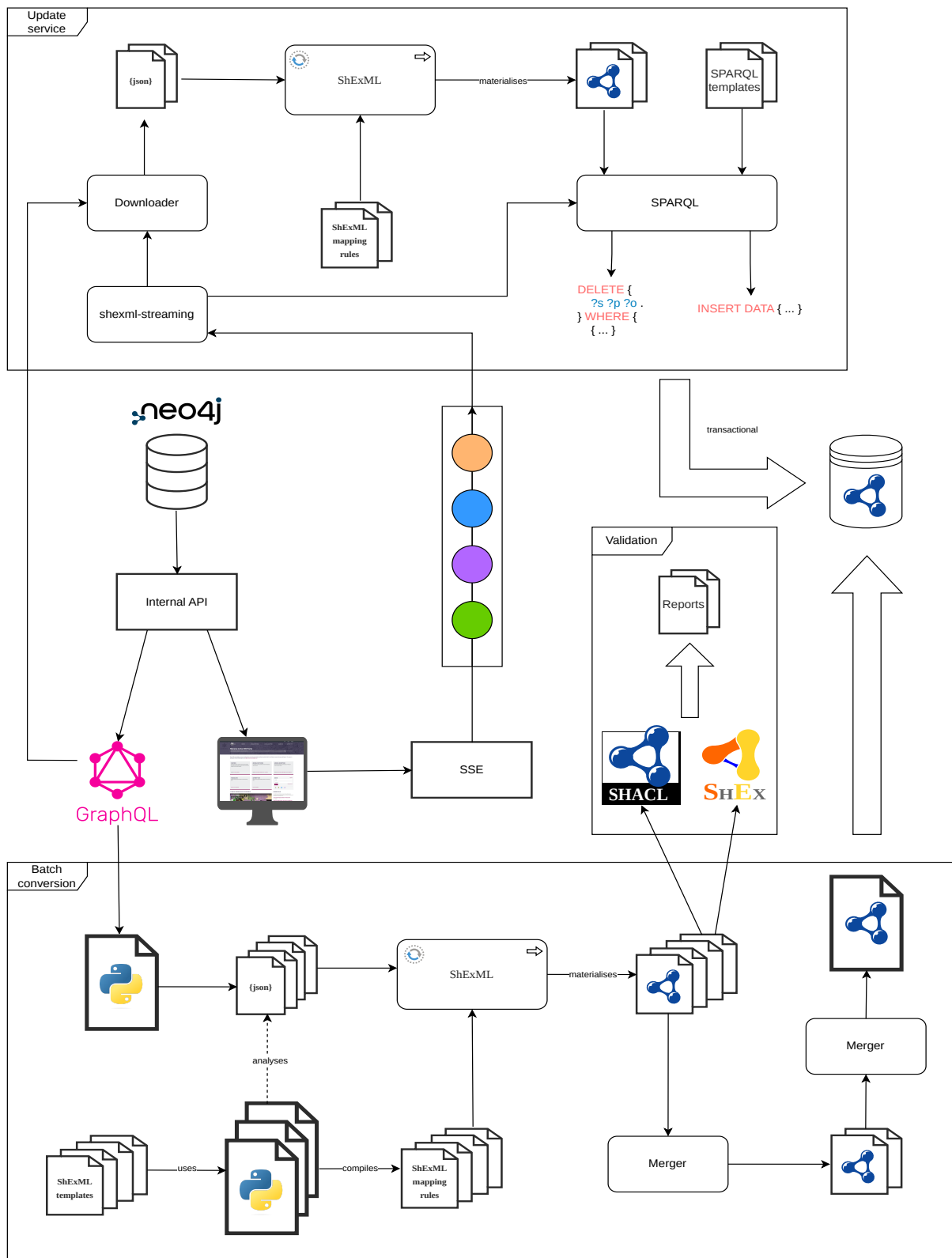


Figure 1: Diagram representing the implemented architecture. The central part represents the existing EHRI Portal stack, where a Neo4j graph database is accessed via an internal API which is leveraged by the EHRI Portal and the GraphQL endpoint. On the bottom, the batch system uses the GraphQL endpoint to perform conversion for targeted entities and materialises a comprehensive RDF graph which is later deposited in a triple store. On the top of the diagram, the update service connects to the SSE endpoint which transmits a series of events which are processed and trigger the different update transactions against the triple store. The validation layer uses the materialisation outputs before any merge. The arrows represent the sense in which the data flows within the architecture.

4.1.1. Downloader

This module downloads data for all the entity types concerned in the conversion process, that is, countries, institutions, archival descriptions, people, corporate bodies and links (copy-original, creation and access points links). For each of these types, a GraphQL query template is provided containing the attributes that we wish to convert. This template is then processed by the module which iterates over it for each result page delivered by the API. As a result, the process will create an output file in the JSON format for each page of an entity type. Given that this process can take a bit of time, we introduced affordances for avoiding flooding the endpoint (in terms of idle time between downloads) and avoiding re-downloading the data that has already been fetched. The latter also allows for a resumption of the download process at any time and tolerates potential network issues.

4.1.2. Data mapping

In order to make this process as reusable, flexible, and understandable as possible we decided to use declarative mapping rules. As described later in this section, however, they cannot yet fulfil all requirements so additional orchestration code was needed.

We decided to use ShExML for the conversion based on a two-fold decision: 1) ease of use as described in [26] (in case non-experts needed to contribute) and 2) as one of the authors is the main developer of the ShExML engine¹⁵, it allowed us to rapidly adapt the language and the engine with new features if required (some of the introduced additions will be explained later).

Thus, the conversion is based around a set of ShExML templates developed for each domain entity type involved. This per-type data mapping allows for a separation of concerns (not necessitating the re-conversion of data for other types whenever a change is introduced) and for updating entities in an isolated fashion (as further explained in Section 4.3). Furthermore, the templates are modularised to increase reusability and readability, which required the addition of a new **IMPORT** directive¹⁶ in ShExML. This directive is able to import other files including arbitrary ShExML contents to compose a single ShExML input prior to parse time.

Additionally, following the downloader's per-page breakdown, a set of mapping rules is created for each results page based on the previously created templates. This is delegated to a Python script which analyses the downloaded JSON files for a certain type and creates the consequent mapping rules, embedding the correct source path in ShExML's **SOURCE** directive (see Listing 1 for an example).

Listing 1: Generated mapping rules for the first institutions' results page where the **SOURCE** is defined by the mapping rules creation script. The **IMPORT** clause allows for centralising the shared code (between the entity type mapping rules) and performing modifications after creation.

```
IMPORT <ShExMLTemplates/partial/RepositoriesHeader.shexml>
IMPORT <ShExMLTemplates/partial/MatcherLanguageCode2Digit.shexml>
SOURCE repositories <institutions/institutions_1.json>
IMPORT <ShExMLTemplates/partial/RepositoriesIteratorsAndShapes.shexml>
```

Right after the creation of all the mapping rules for an entity type, the script calls the ShExML engine with the mapping rules as inputs and checks whether a specific page has already been materialised, as to avoid a re-materialisation and therefore support the resumption of the process at any point in time. As a performance affordance, the Python script allows running the mapping rules in parallel using half of the available CPU cores, for example, if the CPU has 8 cores – either physical or virtual – the script will divide the queued conversions in four chunks and launch four conversion threads simultaneously. The generated RDF files are then persisted in a dedicated folder for each entity type.

4.1.3. Merging results per entity type

Once all pages for an entity type are materialised we need to merge them in order to provide a single RDF file per type. Based on the RDF compositional property one can just merge several RDF graphs

¹⁵<https://github.com/herminiogg/ShExML>

¹⁶<https://shexml.herminiogarcia.com/spec/#import>

without requiring any other major operation. However, in order to favour performance (and avoid re-serialising the data) we opted for a syntactic merge at the level of the file. As we materialise the data in the Turtle serialisation format (due to data size concerns) we need to remove and homogenise the prefix declarations to only preserve one set of declarations at the top of the merged file. This process is performed using the **sed** command line utility which processes the final file in a line-per-line streaming fashion, making the process very efficient. Listing 2 shows an example of this process for data relating to collection-holding institutions.

Listing 2: Excerpt of the Bash script used to merge all the institutions' Turtle files. . . . indicates additional content which has been omitted for brevity reasons.

```
cat shexmlOutputInstitutions/*.ttl > institutions.ttl
sed -i 's/@prefix.*//g' institutions.ttl
sed -i "1i\@prefix wd: <http://www.wikidata.org/entity/> ." institutions.ttl
sed -i "1i\@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> ." institutions.ttl
sed -i "1i\@prefix ehri: <http://lod.ehri-project-test.eu/ontology#> ." institutions.ttl
sed -i "1i\@prefix ehri_countries: <http://lod.ehri-project-test.eu/countries/> ." institutions.ttl
...
```

4.1.4. Merging all entities

Similar to the previous merge, this process combines the output files for different entity types (generated by the previous module) into a single RDF Turtle file, again normalising prefix declarations. Additionally, it downloads the data for the EHRI vocabularies (i.e., controlled terms for subject headings, camps and ghettos), and given that they are already offered in RDF following the SKOS vocabulary, they are simply added into the merging process as an additional entity type file. Finally, a set of links coming from entity linking activities is also incorporated at this stage, allowing us to keep this concern separated and making it independently updatable.

4.1.5. Further data mapping challenges

Even though this architecture addresses many of the issues arising from a lack of certain features in declarative mapping languages, and in ShExML specifically, there are still some challenges that are best addressed within ShExML in order to achieve a more sustainable solution. In this section, we highlight some of them and reason why they are more suited to be solved within the mapping language and/or engine, rather than external tooling.

Matchers: Albeit not being a new feature in ShExML, this functionality took a new dimension during this project. In essence, it allows to replace one pre-defined text occurrence for another, and chain these declarations into a single block for automatic processing in the generation of a particular object.¹⁷ The main use case would be to match naming conventions between different KGs letting users link to other graphs in a seamless manner. However, in this specific project they were used to normalise language tags as the ones coming from the EHRI Portal follow the ISO 639-2¹⁸ codes and we needed to align them with BCP 47¹⁹, as prescribed by the RDF 1.1 specification²⁰, which favours the use of the shortest code from within the ISO 639 family. Listing 3 shows the language code matcher and Listing 4 exemplifies its use within the institutions' mapping rules.

Listing 3: Excerpt of the **MATCHER** used to convert ISO 639-2 language codes into BCP 47 conformant ones. . . . indicates additional content which has been omitted for brevity reasons.

```
MATCHER langcode_two_digit_equivalent <aar AS aa &
abk AS ab &
afr AS af &
aka AS ak &
...>
```

¹⁷<https://shexml.herminiogarcia.com/spec/#matcher>

¹⁸<https://www.loc.gov/standards/iso639-2/>

¹⁹<https://www.rfc-editor.org/info/bcp47>

²⁰<https://www.w3.org/TR/rdf11-concepts/>

Listing 4: Excerpt of the institutions' mapping rules which shows how the **MATCHER** in Listing 3 is leveraged to produce correct language tags. . . . indicates additional content which has been omitted for brevity reasons.

```
ehri:Institution ehri_institutions:[organization.id] {
  a ehri:Institution ;
  rico:name [organization.descriptions.name] @[organization.descriptions.languageCode MATCHING
    langcode_two_digit_equivalent] ;
  ...
}
```

External functions: The possibility to execute certain functions for transforming the output or conditionally generating a triple based on a certain condition is a ubiquitous functionality among different declarative mapping languages, we argue though that their implementation in the ShExML engine allows for a fast prototyping and extensibility. Unlike other languages, these functions are not gathered in a centralised hub/library nor defined in a declarative fashion (e.g., FnO [27]) but rather implemented in pure Scala code in an external file²¹, similar to the mechanism proposed in [28] for RML using Python. This approach provides the full power of a Turing-complete language with a large standard library and allows for rapid prototyping due to the absence of intermediate abstraction layers and boilerplate. The ShExML engine will load these definitions and employ reflection to call the defined functions passing the relevant data as arguments of the function.²² Listing 5 shows an example of a function declaration.

Listing 5: Example of the declaration of a function used to generate the IRIs for the different institutions' addresses.

```
class Transformers {
  def combinedPathInstitutionAddress(institutionID: String, index: String): String = {
    institutionID + "/addresses/" + (index.toInt + 1)
  }
}
```

Built-in functions: Whereas external functions can cover a lot of the use cases where an extension of ShExML is required, there are still some specific functionalities that are only easily solvable natively within the language and/or engine. During this project we identified one of those cases as we needed to recover the iteration index for creating a unique IRI for each generated entity in the iteration loop. For this purpose we initially tried to use the ShExML's autoincrement identifiers²³, along with the previously described external functions feature. Autoincrement IDs, however, were designed to work on a global scope and not capable of attending to nested contexts, whereas external functions were (by design) stateless and similarly not context-aware.

Listing 6: Excerpt of the institutions' mapping rules showing how the **index()** built-in function is used in combination with the function defined in Listing 5 to produce a unique IRI for each address. . . . indicates additional content which has been omitted for brevity reasons.

```
ehri:InstitutionAddress ehri_institutions:[transformers.combinedPathInstitutionAddress(organization.
  descriptions.addresses.repository_id, organization.descriptions.addresses.repository_id.index())]
{
  a ehri:InstitutionAddress ;
  ...
}
```

Relying on the query language for the iteration index, alternatively, can return ambiguous results as, for example, while XPath is able to retrieve information from the surrounding context (albeit in a complex manner) the same is not possible with JSONPath. Therefore, we came up with a new feature for the ShExML language called built-in functions²⁴, as they need to be aware of the current execution context and are, thereby, only implementable within the engine. For the moment only one of such

²¹<https://shexml.herminiogarcia.com/spec/#functions>

²²<https://shexml.herminiogarcia.com/spec/#invoking-functions>

²³<https://shexml.herminiogarcia.com/spec/#autoincrement-ids>

²⁴<https://shexml.herminiogarcia.com/spec/#built-in-functions>

functions has been implemented, `index()` (see Listing 6), whose functionality is similar to that of the index key in the RML Logical Views specification²⁵.

4.2. Validation phase

In an effort to assess the validity of the outputs generated by the batch transformation, we developed a validation workflow which could highlight formatting and normalisation errors in the data. For this purpose, we employed a set of rules in SHACL and ShEx, drafted, in a first instance, automatically by the ShExML engine using its ability to generate RDF shapes from mapping rules and materialised data, and further fine-tuned according to our knowledge of the domain model. The motivation to simultaneously use SHACL and ShEx is two-fold: on the one hand, while both of them are used for the same purpose, i.e., validating RDF data, their foundations and semantics are different and, therefore, the supported features, and their implementation, can vary [29, Ch. 7]; on the other hand, as our implementation closely follows RiC-O these can be re-used by third-party users which will greatly benefit from having both formats available. Consequently, a script was developed to validate each file (per entity type and per results page) against the pair of shapes files and produced a consolidated report highlighting the number of errors and their type (see Listing 7). Following the same rationale used for the conversion process allows us to deliver a much more readable output than just validating against the entire graph and enables a fine-grained inspection of the input and output files relative to the mapping rules. This process has not only allowed us to improve some small details and to detect some human-induced errors in the mapping rules but rather more importantly to identify errors and non-normalised data in the EHRI Portal.

Listing 7: Excerpt of the validation report showing the validation status for some institutions' data and the total for this type of entity. This report is completed by another file that hosts the validation errors as outputted by the validation engine. . . . indicates additional content which has been omitted for brevity reasons.

```
...
Validating shexmlOutputInstitutions/institutions_6.ttl: 92 conformant, 8 non-conformant
Validating shexmlOutputInstitutions/institutions_7.ttl: 95 conformant, 5 non-conformant
Validating shexmlOutputInstitutions/institutions_8.ttl: 95 conformant, 5 non-conformant
Validating shexmlOutputInstitutions/institutions_9.ttl: 71 conformant, 29 non-conformant
Institutions: 1771 conformant, 524 non-conformant
...
```

4.3. Continuous update

Considering that the resulting graph is mainly the output of the batch conversion process, and that the EHRI Portal is continuously receiving new input and ongoing curation [30], there exists a risk of rapidly accumulating a considerable amount of stale data for longer intervals between batch conversion runs. As running the batch conversion takes a considerable amount of resources and time, and can be challenging to operate in a fully unattended manner (mainly due to the possibility of networking and mapping errors, or even infrastructural ones, like power outages), we set out to create a parallel strategy for continuously keeping the data sources synchronised.

Updating materialised data produced by mapping rules is the subject of active research, for instance, IncRML [31] was recently proposed as a way to detect changes on materialised data and handling updates in a surgical manner without rematerialising the whole graph. However, while making use of this approach or even implementing it in ShExML was a possibility – and it will certainly be considered as future work for the presented architecture – we required a method that was straightforward to implement and, at the same time, easy to modify, flexible (so as to permitting very fine-grained operations), and very efficient.

Taking all these requisites into account, we built upon a system already implemented in the EHRI Portal which exposes a Server-Sent Events (SSE) feed emitting information about lifecycle events such as

²⁵<https://kg-construct.github.io/rml-lv/spec/docs/#dfn-index-key>

the creation, update, or deletion of Portal entities. For processing these events, the shexml-streaming²⁶ library was developed which encapsulates the ShExML engine under a Reactive API and adds support for streaming sources such as SSE or Websockets, or distributed event streaming platforms like Apache Kafka²⁷. This allows us to perform all the operations in a non-blocking, asynchronous, event-based manner as the output will be encapsulated in an **Observable**²⁸ which is to be executed and consumed by the client rather than the library.

From this point, we considered two different options: the use of an Object-Relational Mapping (ORM)-like system for RDF (e.g., DMAOG [32]), or building an *ad-hoc* solution. While the ORM approach would make it possible to avoid creating a lot of task-specific code and would automatically create the necessary SPARQL Update queries, it would also delete triples that should not be removed due to lacking necessary context from the mapping rules that generated them. As a consequence, we created a small module dubbed the EHRI KG Update Service²⁹.

This service builds upon the previously introduced shexml-streaming library and leverages the **Observable** output to perform its operations in a monoid-like fashion. For each event, the module performs the following operations in sequential order: 1) parses the event data (received from shexml-streaming), 2) downloads the new data from the EHRI Portal GraphQL API, 3) transforms the data using the ShExML engine, 4) transactionally executes the DELETE and/or INSERT SPARQL queries against the SPARQL Update endpoint based on a previously configured SPARQL template (see Listing 8 for an example), and 5) computes the difference between the entity's data before and after the update and outputs a report.

Listing 8: Template used by the update service for the deletion of archival descriptions.

```
PREFIX ehri_units: <http://lod.ehri-project-test.eu/units/>
PREFIX rico: <https://www.ica.org/standards/RiC/ontology#>
PREFIX ehri: <http://lod.ehri-project-test.eu/ontology#>

DELETE {
  ?s ?p ?o .
} WHERE {
  {
    # All the main triples where the unit is the subject except those that are generated as part of
    # other set of mapping rules
    ?s ?p ?o .
    FILTER(?s = ehri_units:<$entityId>
      && ?p != rico:hasOrHadSubject
      && ?p != ehri:isCopyOf
      && ?p != ehri:hasCopy
      && ?p != rico:hasCreator)
  }
  UNION
  {
    # Connected instances that can be completely removed as they are only bound to the unit
    ?s ?p ?o ;
    a ?connectedClasses ;
    ?inverseProperty ?unit .
    VALUES ?connectedClasses { rico>Title rico:Activity rico:Identifier }
    FILTER(?unit = ehri_units:<$entityId>)
  }
  UNION
  {
    # Other triples that need to be cleaned up on a per-case basis
    ?s ?p ?o .
    VALUES ?p { rico:isOrWasHolderOf rico:isOrWasTitleOf rico:isBeginningDateOf rico:isEndDateOf rico:
      includesOrIncluded ehri:isOrWasScriptOf rico:isOrWasIdentifierOf }
    FILTER(?o = ehri_units:<$entityId>)
  }
}
```

²⁶<https://github.com/herminiogg/shexml-streaming>

²⁷<https://kafka.apache.org/>

²⁸<https://reactivex.io/documentation/observable.html>

²⁹<https://github.com/EHRI/ehri-kg-update-service>

This module is highly configurable and even allows for the hot swapping of mapping rules, SPARQL templates or GraphQL queries, making it very amenable to an iterative evolution of the conversion process. However, as detailed in Section 6, this aspect of the architecture is still under active development and we will seek to improve it further and test different approaches.

5. Discussion

The architecture presented here is currently being used to populate the EHRI-KG and, while not totally finished, it already contains the large majority of data planned for its first iteration. Presently, the data coming from the EHRI Portal amounts to around 1.2 GB in JSON files. Once converted to RDF the final Turtle file contains almost 7 million triples containing almost 1 million unique entities and 108 different relations. This is, of course, our baseline, and the architecture is implemented, as introduced earlier, in a way that allows for a steady growth of the EHRI Portal and, as a consequence, of the EHRI-KG. In the rest of this section we elaborate on some of the key decisions taken in order to inspire future developments within the community.

The most prominent aspect of this architecture is the orchestration of different sets of mapping rules, a domain which declarative mapping languages and engines themselves have mostly side-stepped, delegating this to other parts of the KGC pipeline [33]. Looking at the Extract Transform Load (ETL) paradigm from an architectural point of view, declarative mapping languages have always fallen under the Transform step, without going further in terms of Extract and Load. While there are some initiatives that try to go a bit further, like the RML-IO module³⁰ for the new RML specification proposal [34] (which not only defines the `rml:LogicalSource` but also the `rml:LogicalTarget`, hitherto widely relegated to the implementing engine) and the possibility in SPARQL-Generate [35] to call REST APIs in an iterative manner using different parameters, there is still a common lack of advanced mechanisms that allow performing more complex queries against commonly used endpoints. For example, being able to perform POST requests (in addition to the by-default GET method used by most of the existing mapping languages), where the user can define a custom payload (in this example the GraphQL query) and navigate automatically through a set of paginated results, would reduce a lot of orchestration boilerplate, as well as helping to standardise approaches and make them more reproducible. The recently proposed HTTP Request Access and Dynamic Targets RML extensions [36] seek to implement some of this functionality (e.g., the possibility to perform authenticated POST requests), though it does not seem that they are at the moment able to provide a comprehensive solution for the aforementioned scenario.

Similarly, the possibility to apply a set of mapping rules over a predetermined set of files or endpoints will reduce the efforts needed for merging outputs – not because employing a syntactic merge approach is too cumbersome but as a usability affordance for final users – and could further enable the transformation to be parallelised per source. In ShExML, wildcards³¹ grant the possibility to iterate over a set of files, albeit in a fairly inflexible manner, and parallelisation has been recently introduced in the ShExML engine. As disclosed in the documentation, however, not all mapping rules benefit equally from the use of parallelisation techniques.³² Potentially, parallelising on the level of the file will supersede these limitations as we are doing within the orchestration scripts in the presented architecture. Moreover, in order to optimise CPU and RAM usage, the introduction of mini-batches could allow users to pre-define their size, maximise the benefits of a parallel approach and avoid computing bottlenecks, similar to how mini-batches are used in deep learning for gradient descent. Many programming languages APIs already perform this in a heuristic manner like, for example Scala’s parallel collections³³ (used by the ShExML engine) or the Parallel Language-Integrated Query (PLINQ) in C#³⁴, but offering additional configuration options will help users to optimise their data conversions even further. Parallelisation strategies have been explored in the past by the community, like for example, RMLStreamer [37] which leverages

³⁰<https://kg-construct.github.io/rml-io/spec/docs/>

³¹<https://shexml.herminiogarcia.com/spec/#wildcards>

³²<https://github.com/herminiogg/ShExML?tab=readme-ov-file#parallelisation>

³³<https://docs.scala-lang.org/overviews/parallel-collections/overview.html>

³⁴<https://learn.microsoft.com/en-us/dotnet/standard/parallel-programming/introduction-to-plinq>

Apache Flink to compute input files or streams in a parallel and streaming manner or Morph-KGC [38] which leverages the concept of mapping partitions to generate disjoint subsets of the target KG, however, empowering users with the possibility to provide fine-grained configurations and decide which aspects are parallelised will help tools to cover different use cases in a more efficient manner.

When dealing with large mapping rules files with a lot of conditions, functions and repeated patterns, the ability to modularise the mapping rules is of great importance. Even in the case of ShExML, whose syntax is fairly succinct in comparison with other mapping languages, the inclusion of the **IMPORT** clause as a way to reuse mapping rules and organise the different aspects of the mapping file, enabled us to have a much more coherent, organised, and maintainable codebase for our conversion architecture, making the process more sustainable and future proof.

Finally, a key step when publishing a KG is to ensure that the generated data is valid. As introduced earlier, SHACL and ShEx can be leveraged for this purpose, or even automatically produced from inputs of one of the processes within the KGC pipeline [33]. Using scripts, or other external tools, as introduced in our architecture, is a viable option, yet requires the user to include this step in their workflow and to analyse the results separately. We contend, therefore, that the incorporation of such feature within the mapping languages and/or engines would ease the burden on users to validate data and make the process more approachable and trustworthy. This could be articulated through the declaration of an external file containing the shapes to which the results of a certain set of mapping rules should conform. Thereby, at materialisation time, the user could already glimpse the validation results and adjust the mapping rules accordingly, in a more interactive and iterative manner.

6. Future Work

As it stands, the architecture presented here is able to deliver a comprehensive KG as a semantically-enabled replica of the EHRI Portal, and already fulfils the requirements we set out to deliver. Hence, we do not expect big changes on the overall architecture and our efforts will be mainly dedicated to improving and streamlining some specific aspects to ensure its future sustainability and scalability.

Based on the discussion established in Section 5, our main line of work will be dedicated to the optimisation of the batch conversion, to ensure that it fully utilises all the available resources and reduces, as a consequence, the materialisation time. At present, the Python-based orchestration of the ShExML engine results in repeatedly loading both the JVM itself and the mapping rules (along with any external Scala-based functions), and this adds a non-negligible overhead to the overall process. As a possible solution we will investigate the idea, previously introduced, of using mini-batches for the transformation process which, in combination with the parallelisation over files, could provide some performance improvements.

Similarly, rather than waiting for a GraphQL query to fetch all pages before starting conversion, this task could potentially be triggered immediately after each page download to increase throughput, at the expense, however, of introducing new side effects and potential errors. Therefore, if we were to couple this process, we would investigate declarative ways to traverse endpoints (like REST and GraphQL APIs) which could be integrated in or linked to ShExML, reducing the required orchestration code and the overall complexity, while relying on the inner well-tested coordination mechanisms of the engine as well as enabling the efficient parallelisation discussed above.

In contrast, as the newest layer of this architecture, the update service could evolve rather more, not necessarily because it does not cover the required functionality, but to make it more reliable and easier to maintain. On the reliability side, the main line of work would be the deprecation of the SSE API in favour of a distributed event streaming platform, such as Apache Kafka, that would enable a more distributed and fault-tolerant set-up, a persisted log-based storage (which could additionally serve as a source of truth), and the ability to replay events. Moreover, it would allow us to lean more on the new shexml-streaming library and reshape the update layer architecture as a set of data pipelines within Apache Kafka's topics system. As for the maintainability, it once again hinges on the capability to represent this process in a more declarative manner without incurring in additional performance

costs. Additionally, the generalisation of the update service will be studied as a means to offer a generic update framework over streaming sources.

Finally, although not germane to the functioning of the conversion architecture, the inclusion of an optional validation step immediately after materialisation will be considered for inclusion in the ShExML engine. This should provide a more interactive framework for developing declarative mapping rules and assist in the earlier detection of issues, especially in combination with the ability to scaffold validation shapes from data mapping rules.

7. Conclusions

The architecture presented here has allowed EHRI to provide the data of the EHRI Portal as a KG, making it more interoperable, linkable, and aligned with new archival standards such as RiC-O, as well as improving the FAIR compliance of the overall Research Infrastructure. Given the constraints around rebuilding the existing technical stack, the current implementation allows us to maintain an up-to-date replica of the EHRI Portal which also enables new features like the ability to perform arbitrary queries or even executing Federated SPARQL queries throughout linked KGs.

One of the main conclusions of this paper is that even though a great deal of work can be handled by declarative mapping technologies there are still many operations not supported by them, and for which *ad-hoc* solutions are required. However, as discussed above, we see a lot of room for improvement where new features, arising from the necessity of real-world use cases, can enrich the development of these tools and inform new research lines within the field. Until these advancements make their way into practical tools the architecture designed here, and the techniques used to overcome some of the existing limitations and challenges, can inspire other practitioners facing similar scenarios.

By highlighting the deficiencies of existing declarative data mapping techniques we are by no means undermining their value. On the contrary, if this work was at all possible – or achievable in a somewhat short timeframe – it is mainly due to the productivity advantage that these tools provide, making the process more amenable, maintainable and, especially, much more efficient. Therefore, this paper describes our endeavours in bringing a production-ready architecture for the EHRI-KG and builds upon them to signal future advancements in the field which could make these kinds of ventures more feasible and approachable, thereby increasing the productivity of users and reducing project delivery time.

Supplemental material availability

The source code for the described batch conversion as well as its associated resources are openly available on GitHub (<https://github.com/EHRI/ehri-kg-mapping>) and under the following permanent DOI: <https://doi.org/10.5281/zenodo.17359136>. The source code for the update service is available on GitHub: <https://github.com/EHRI/ehri-kg-update-service>.

Funding

This work has been carried out in the context of the OSCARS project, which has received funding from the European Commission's Horizon Europe Research and Innovation programme under grant agreement No. 101129751.

Declaration on Generative AI

During the preparation of this work the authors used ChatGPT and Gemini in order to: grammar and spelling check, paraphrase and reword, and improve writing style. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

- [1] T. Blanke, M. Bryant, M. Frankl, C. Kristel, R. Speck, V. V. Daelen, R. V. Horik, The european holocaust research infrastructure portal, *J. Comput. Cult. Herit.* 10 (2017). doi:10.1145/3004457.
- [2] H. García-González, M. Bryant, The holocaust archival material knowledge graph, 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*, Springer Nature Switzerland, Cham, 2023, pp. 362–379. doi:10.1007/978-3-031-47243-5_20.
- [3] F. Clavaud, T. Wildi, Ica records in contexts-ontology (ric-o): a semantic framework for describing archival resources, in: *Linked Archives 2021: Proceedings of Linked Archives International Workshop 2021 co-located with 25th International Conference on Theory and Practice of Digital Libraries (TPDL 2021)*, 2021, pp. p–79. URL: https://ceur-ws.org/Vol-3019/LinkedArchives_2021_paper_13.pdf.
- [4] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al., The fair guiding principles for scientific data management and stewardship, *Scientific data* 3 (2016) 1–9. doi:10.1038/sdata.2016.18.
- [5] D. Van Assche, T. Delva, G. Haesendonck, P. Heyvaert, B. De Meester, A. Dimou, Declarative rdf graph generation from heterogeneous (semi-)structured data: A systematic literature review, *Journal of Web Semantics* 75 (2023) 100753. doi:10.1016/j.websem.2022.100753.
- [6] A. Meroño-Peñuela, A. Ashkpour, M. van Erp, K. Mandemakers, L. Breure, A. Scharnhorst, S. Schlobach, F. van Harmelen, Semantic technologies for historical research: A survey, *Semantic Web* 6 (2015) 539–564. doi:10.3233/SW-140158.
- [7] J. Chen, H. Dong, J. Hastings, E. Jiménez-Ruiz, V. López, P. Monnin, C. Pesquita, P. Škoda, V. Tamma, Knowledge Graphs for the Life Sciences: Recent Developments, Challenges and Opportunities, *Transactions on Graph Data and Knowledge* 1 (2023) 5:1–5:33. doi:10.4230/TGDK.1.1.5.
- [8] T. Hubauer, S. Lamparter, P. Haase, D. M. Herzig, Use cases of the industrial knowledge graph at siemens, in: M. van Erp, M. Atre, V. López, K. Srinivas, C. Fortuna (Eds.), *Proceedings of the ISWC 2018 Posters & Demonstrations, Industry and Blue Sky Ideas Tracks co-located with 17th International Semantic Web Conference (ISWC 2018)*, Monterey, USA, October 8th - to - 12th, 2018, volume 2180 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018. URL: <https://ceur-ws.org/Vol-2180/paper-86.pdf>.
- [9] U. Simsek, K. Angele, E. Kärle, J. Opdenplatz, D. Sommer, J. Umbrich, D. Fensel, Knowledge graph lifecycle: Building and maintaining knowledge graphs, 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: <https://ceur-ws.org/Vol-2873/paper12.pdf>.
- [10] R. Pernisch, M. Poveda-Villalón, D. Conde-Herreros, D. Chaves-Fraga, L. Stork, When ontologies met knowledge graphs: Tale of a methodology, in: A. Meroño-Peñuela, Ó. Corcho, P. Groth, E. Simperl, V. Tamma, A. G. Nuzzolese, M. Poveda-Villalón, M. Sabou, V. Presutti, I. Celino, A. Revenko, J. Raad, B. Sartini, P. Lisena (Eds.), *The Semantic Web: ESWC 2024 Satellite Events - Hersonissos, Crete, Greece, May 26-30, 2024, Proceedings, Part I*, volume 15344 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 286–290. doi:10.1007/978-3-031-78952-6_43.
- [11] A. Sakor, S. Jozashoori, E. Niazmand, A. Rivas, K. Bougiatiotis, F. Aisopos, E. Iglesias, P. D. Rohde, T. Padiya, A. Krithara, G. Paliouras, M.-E. Vidal, Knowledge4covid-19: A semantic-based approach for constructing a covid-19 related knowledge graph from various sources and analyzing treatments' toxicities, *Journal of Web Semantics* 75 (2023) 100760. doi:10.1016/j.websem.2022.100760.
- [12] A. Dimou, M. V. Sande, P. Colpaert, R. Verborgh, E. Mannens, R. V. de Walle, RML: A generic language for integrated RDF mappings of heterogeneous data, in: C. Bizer, T. Heath, S. Auer, T. Berners-Lee (Eds.), *Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW 2014)*, Seoul, Korea, April 8, 2014, volume 1184 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2014. URL: <https://ceur-ws.org/Vol-1184/>

ldow2014_paper_01.pdf.

- [13] C. Guasch, G. Lodi, S. V. Dooren, Semantic knowledge graphs for distributed data spaces: The public procurement pilot experience, in: U. Sattler, A. Hogan, M. Keet, V. Presutti, J. P. A. Almeida, H. Takeda, P. Monnin, G. Pirrò, C. d'Amato (Eds.), *The Semantic Web – ISWC 2022*, Springer International Publishing, Cham, 2022, pp. 753–769. doi:10.1007/978-3-031-19433-7_43.
- [14] U. Simsek, J. Umbrich, D. Fensel, Towards a knowledge graph lifecycle: A pipeline for the population of a commercial knowledge graph, in: A. Paschke, C. Neudecker, G. Rehm, J. A. Qundus, L. Pintscher (Eds.), *Proceedings of the Conference on Digital Curation Technologies (Qurator 2020)*, Berlin, Germany, January 20th - 21st, 2020, volume 2535 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2020. URL: https://ceur-ws.org/Vol-2535/paper_10.pdf.
- [15] J. A. Rojas, M. Aguado, P. Vasilopoulou, I. Velitchkov, D. Van Assche, P. Colpaert, R. Verborgh, Leveraging semantic technologies for digital interoperability in the european railway domain, in: A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue, Y. Ding, P. Barnaghi, A. Haller, M. Dragoni, H. Alani (Eds.), *The Semantic Web – ISWC 2021*, Springer International Publishing, Cham, 2021, pp. 648–664. doi:10.1007/978-3-030-88361-4_38.
- [16] P. Heyvaert, B. D. Meester, A. Dimou, R. Verborgh, Declarative rules for linked data generation at your fingertips!, in: A. Gangemi, A. L. Gentile, A. G. Nuzzolese, S. Rudolph, M. Maleshkova, H. Paulheim, J. Z. Pan, M. Alam (Eds.), *The Semantic Web: ESWC 2018 Satellite Events - ESWC 2018 Satellite Events*, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers, volume 11155 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 213–217. doi:10.1007/978-3-319-98192-5_40.
- [17] M. Scrocca, M. Comerio, A. Carenini, I. Celino, Turning transport data to comply with EU standards while enabling a multimodal transport knowledge graph, in: J. Z. Pan, V. Tamma, C. d'Amato, K. Janowicz, B. Fu, A. Polleres, O. Seneviratne, L. Kagal (Eds.), *The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference*, Athens, Greece, November 2-6, 2020, *Proceedings, Part II*, volume 12507 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 411–429. doi:10.1007/978-3-030-62466-8_26.
- [18] 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 ESWC 2023*, Hersonissos, Greece, May 28, 2023, volume 3471 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2023. URL: <https://ceur-ws.org/Vol-3471/paper9.pdf>.
- [19] D. Chaves-Fraga, O. Corcho, F. Yedro, R. Moreno, J. Olías, A. De La Azuela, Systematic construction of knowledge graphs for research-performing organizations, *Information* 13 (2022). doi:10.3390/info13120562.
- [20] R. Pernisch, D. Chaves-Fraga, L. Stork, D. Conde-Herreros, M. Poveda-Villalón, Lot4kg: A joint methodology for the ontology and knowledge graph lifecycle, Under review, 2026.
- [21] L. Brazzo, S. Mazzini, From the holocaust victims names to the description of the persecution of the european jews in nazi years: the linked data approach and a new domain ontology, *Book of abstract of DH* (2015).
- [22] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, *Commun. ACM* 57 (2014) 78–85. doi:10.1145/2629489.
- [23] D. Diefenbach, M. D. Wilde, S. Alipio, Wikibase as an infrastructure for knowledge graphs: The EU knowledge graph, in: A. Hotho, E. Blomqvist, S. Dietze, A. Fokoue, Y. Ding, P. M. Barnaghi, A. Haller, M. Dragoni, H. Alani (Eds.), *The Semantic Web - ISWC 2021 - 20th International Semantic Web Conference*, ISWC 2021, Virtual Event, October 24-28, 2021, *Proceedings*, volume 12922 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 631–647. doi:10.1007/978-3-030-88361-4_37.
- [24] R. Kitchin, G. McArdle, What makes big data, big data? exploring the ontological characteristics of 26 datasets, *Big Data & Society* 3 (2016) 2053951716631130. doi:10.1177/2053951716631130.
- [25] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Hae-sendonck, P. Colpaert, Triple pattern fragments: A low-cost knowledge graph interface for the web,

- Journal of Web Semantics 37-38 (2016) 184–206. doi:<https://doi.org/10.1016/j.websem.2016.03.003>.
- [26] H. García-González, I. Boneva, S. Staworko, J. E. L. Gayo, J. M. C. Lovelle, ShExML: improving the usability of heterogeneous data mapping languages for first-time users, *PeerJ Compututer Science* 6 (2020) e318. URL: [10.7717/peerj-cs.318](https://doi.org/10.7717/peerj-cs.318). doi:[10.7717/peerj-cs.318](https://doi.org/10.7717/peerj-cs.318).
 - [27] B. De Meester, T. Seymoens, A. Dimou, R. Verborgh, Implementation-independent function reuse, *Future Generation Computer Systems* 110 (2020) 946–959. doi:[10.1016/j.future.2019.10.006](https://doi.org/10.1016/j.future.2019.10.006).
 - [28] J. Arenas-Guerrero, P. Espinoza-Arias, J. A. Bernabé-Díaz, P. Deshmukh, J. L. Sánchez-Fernández, Ó. Corcho, An RML-FNML module for python user-defined functions in morph-kgc, *SoftwareX* 26 (2024) 101709. doi:[10.1016/J.SOFTX.2024.101709](https://doi.org/10.1016/J.SOFTX.2024.101709).
 - [29] J. E. L. Gayo, E. Prud’hommeaux, I. Boneva, D. Kontokostas, Validating RDF Data, *Synthesis Lectures on the Semantic Web: Theory and Technology*, Morgan & Claypool Publishers, 2017. doi:[10.2200/S00786ED1V01Y201707WBE016](https://doi.org/10.2200/S00786ED1V01Y201707WBE016).
 - [30] H. García-González, M. Bryant, S. Swartz, F. Rovigo, V. V. Daelen, Running a data integration lab in the context of the ehri project: Challenges, lessons learnt and future directions, 2025. URL: <https://arxiv.org/abs/2505.02455>. arXiv:2505.02455.
 - [31] D. Van Assche, J. A. Rojas, B. De Meester, P. Colpaert, Incremental knowledge graph construction from heterogeneous data sources, *Semantic Web* (2025).
 - [32] H. García-González, M. Bryant, V. Vanden Daelen, Stop writing repetitive code! scaffolding a semantic data access layer to abstract developers from semantic technologies, *Journal of Web Semantics* 88 (2026) 100874. doi:[10.1016/j.websem.2025.100874](https://doi.org/10.1016/j.websem.2025.100874).
 - [33] M. Hofer, D. Obraczka, A. Saeedi, H. Köpcke, E. Rahm, Construction of knowledge graphs: Current state and challenges, *Information* 15 (2024). doi:[10.3390/info15080509](https://doi.org/10.3390/info15080509).
 - [34] 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*, Springer Nature Switzerland, Cham, 2023, pp. 152–175. doi:[10.1007/978-3-031-47243-5_9](https://doi.org/10.1007/978-3-031-47243-5_9).
 - [35] M. Lefrançois, A. Zimmermann, N. Bakerally, A SPARQL extension for generating RDF from heterogeneous formats, in: E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler, O. Hartig (Eds.), *The Semantic Web - 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 - June 1, 2017, Proceedings, Part I*, volume 10249 of *Lecture Notes in Computer Science*, 2017, pp. 35–50. doi:[10.1007/978-3-319-58068-5_3](https://doi.org/10.1007/978-3-319-58068-5_3).
 - [36] E. de Vleeschauwer, G. Haesendonck, B. D. Meester, P. Colpaert, Extending RML to support permissioned data sharing with multiple views, in: D. Chaves-Fraga, I. Dasoulas, C. Debruyne, A. Dimou, U. Serles, D. V. Assche (Eds.), *Proceedings of the 6th International Workshop on Knowledge Graph Construction co-located with 22nd Extended Semantic Web Conference (ESWC 2025)*, Portorož, Slovenia, June 1, 2025, volume 3999 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2025. URL: <https://ceur-ws.org/Vol-3999/paper2.pdf>.
 - [37] G. Haesendonck, W. Maroy, P. Heyvaert, R. Verborgh, A. Dimou, Parallel rdf generation from heterogeneous big data, in: *Proceedings of the International Workshop on Semantic Big Data, SBD ’19*, Association for Computing Machinery, New York, NY, USA, 2019. doi:[10.1145/3323878.3325802](https://doi.org/10.1145/3323878.3325802).
 - [38] J. Arenas-Guerrero, D. Chaves-Fraga, J. Toledo, M. S. Pérez, O. Corcho, Morph-kgc: Scalable knowledge graph materialization with mapping partitions, *Semantic Web* 15 (2024) 1–20. doi:[10.3233/SW-223135](https://doi.org/10.3233/SW-223135).