Extending Semantic RML Mappings with Additional Source Formats

Johannes Theissen-Lipp[1,2][a], Niklas Schäfer[3][b], Max Kocher[1][c], Philipp Hochmann[1][d], Michael Riesener[3][e] and Stefan Decker[1,2][f]

[1]*Chair of Databases and Information Systems, RWTH Aachen University, Aachen, Germany*

[2]*Fraunhofer Institute for Applied Information Technology FIT, Sankt Augustin, Germany*

[3]*Laboratory for Machine Tools and Production Engineering (WZL), RWTH Aachen University, Aachen, Germany*

Keywords: RML, RDF Mappings, Heterogeneous Formats, Domain-Specific Language, IFC, UML.

Abstract: Across many domains, the growing amount of data presents a challenge in extracting meaningful insights. A significant hurdle is the accurate interpretation and integration of data from diverse sources, often dictated by their specific applications. The RDF Mapping Language (RML), based on the W3C recommendation R2RML, can be used to transform heterogeneous data formats to RDF using defined mappings. However, existing RML implementations only support a limited set of (semi-)structured data sources such as CSV, SQL, XML, and JSON, neglecting numerous use-cases relying on other formats. This work overcomes this limitation by proposing a methodology to flexibly extend RML to support additional source formats. We systematically analyze RML and its implementations to derive a generic concept for the extension of RML. Our contributions include a general workflow for extending RML with new formats and demonstrative implementations of the RML Mapper for two examples from Building Information Modeling (BIM) and UML class diagrams. Leveraging open-source code forks and a demonstrative domain-specific language ensures easy portability to any other source format. The evaluation covers authoring of mappings, runtime performance, and practical applicability. The results affirm the effectiveness of our generic methodology for extending RML mappings to include additional source formats.

1 INTRODUCTION

The growth of web-based platforms and digitalization across industries has led to a rise in data accumulation, facilitating decision making and cross-domain collaboration. To ensure effective sharing, semantically accurate and contextually aware data from different domains such as production, development, and operations are required (Schuh et al., 2019). The adoption of the graph-based Resource Description Framework (RDF) as a standard data model enables the representation of information and knowledge in a machine-understandable and accessible manner. RDF facilitates the merging of data sets from disparate networks using global identifiers and supporting tools such as validators and reasoners. However, the applicability of RDF as a general-purpose format is limited due to potential data enlargement and inefficient processing (Ravindra et al., 2011). Certain use cases require specialized data structures for efficient data manipulation, leading to interoperability challenges. As a result, larger data sets often remain in their native format and are selectively converted to RDF for specific purposes. Generating interpretable RDF requires the inclusion of metadata, which is often implicit in the source data. Various mapping languages facilitate the manual addition of semantics from the input data model to the conversion process.

The RDF Mapping Language R2RML, established as a W3C recommendation in 2012 by the RDB2RDF working group, facilitates the mapping of relational databases to RDF citer2rml-tr. In addition, Dimou et al. introduced source-independent mappings, extending the capabilities of RML to convert formats beyond relational databases (Dimou et al., 2013; Dimou et al., 2014; Dimou et al., 2020). RML serves as a key mapping language for RDF, supported by valuable tools such as a graph-based mapping edi-

[a] https://orcid.org/0000-0002-2639-1949
[b] https://orcid.org/0000-0003-4329-9244
[c] https://orcid.org/0000-0001-9801-0300
[d] https://orcid.org/0000-0001-8740-5606
[e] https://orcid.org/0000-0001-8633-7728
[f] https://orcid.org/0000-0001-6324-7164

Theissen-Lipp, J., Schäfer, N., Kocher, M., Hochmann, P., Riesener, M. and Decker, S.
Extending Semantic RML Mappings with Additional Source Formats.
DOI: 10.5220/0012551300003690
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 26th International Conference on Enterprise Information Systems (ICEIS 2024) - Volume 1*, pages 201-208
ISBN: 978-989-758-692-7; ISSN: 2184-4992
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

tor (Heyvaert et al., 2016) and an active research community. Experts analyze the syntactic and semantic data structures of input data, and formulate and execute mappings to merge and transform heterogeneous sources into RDF. Currently, RML supports hierarchical (e.g., JSON, XML) and tabular (e.g., RDB, CSV) formats. Despite extensive research, the challenge of conveniently extending RML's supported source formats to accommodate new ones based on domain or application requirements remains.

Enabling RML mappings for different source formats is critical (Lipp et al., 2020). Incorporating application-specific metadata is more advantageous than a generalized approach. Extending RML to support new graph-based data models enhances knowledge graph construction. The potential and limitations are explored through two use cases: *Unified Modeling Language* (UML) diagrams from *Draw.io* and *Industry Foundation Classes* (IFC). Although these formats have a graph data model, they lack RML support.

IFC files in *Building Information Modeling* (BIM) represent 3D building and factory models typically created in proprietary BIM authoring tools such as Autodesk's *Revit*. Handling the IFC data model is challenging because it was originally intended as an exchange format (Malcolm et al., 2021). Converting relevant components of a model to RDF allows for reasoner-based evaluations to verify implicit building information and validate design scenarios (Beetz et al., 2021; Burggräf et al., 2021). For example, the calculation of escape route dimensions could automatically assists factory planners. RML mappings could assist in the partial and thus scalable conversion of IFC files for further analysis.

Draw.io can create diagrams such as UML class diagrams representing object-oriented models. Storing the domain knowledge of class diagrams as an ontology in an RDF model is desirable. However, automatic conversion is hindered by the lack of an established standard for mapping class diagram semantics to ontology semantics. RML mappings could describe how components are converted to the Web Ontology Language (OWL), potentially accelerating model-driven systems engineering.

This paper presents a general methodology for systematically extending RML with new input formats (see fig. 1). It addresses the research questions on (i) data format requirements for RML conversion, (ii) enhancing user-friendliness of source-specific mapping components, and (iii) software components and requirements for RML implementation.

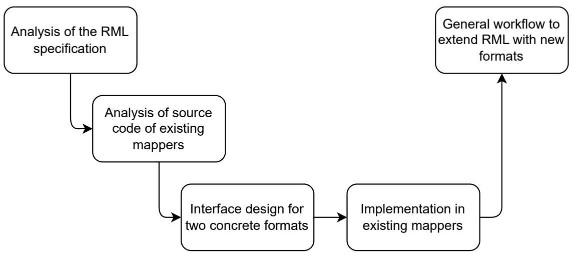

Analysis of the RML specification
Analysis of source code of existing mappers
Interface design for two concrete formats
Implementation in existing mappers
General workflow to extend RML with new formats

Figure 1: Our five-step approach for designing generic RML extensions starts with an analysis of RML specifications, includes concrete implementations in use cases, and finally derives a general RML extension workflow.

2 BACKGROUND

In the following, RML and its relation to IFC and class diagrams are described. IFC defines a data schema and exchange file format structure for BIM data and is used as a demonstration format to develop a general workflow for extending RML.

2.1 RDF Mapping Language (RML)

Established as a widely used mapping language, RML builds on W3C recommendations such as R2RML (Das et al., 2012) and is influenced by related research such as RML-star (Iglesias-Molina et al., 2022). It is influenced by Dimou et al. (Dimou et al., 2013; Dimou et al., 2014) and enables users to convert heterogeneous data formats such as CSV/SQL/XML into RDF triples. RML mappings are typically defined in RDF format using terms from the RML ontology[1] and executed by RML processors. A mapping consists of one or more *TriplesMaps* defining mappings from a source file to RDF triples. The resulting nodes can be joined to generate linked data from heterogeneous sources. While most of a mapping is source-independent and semantically describes the creation of the output graph, source-dependent parts include *iterators* and *references* that access the contents of the input file during the mapping process. Listing 1 provides an example RML mapping, which extracts data from a JSON file by iterating over each "contact" object, creating triples with the contact's name and phone field.

2.2 Industry Foundation Classes (IFC)

Defined in the international standard ISO 16739-1:2018, the buildingSMART organization maintains IFC as an open specification for representing 3D

[1] http://semweb.mmlab.be/ns/rml

Listing 1: Example RML mapping with an iterator and two references highlighted in yellow.

```
<#PhoneMapping>
  a rr:TriplesMap ;
  rml:logicalSource [
    rml:source "phone-numbers.json" ;
    rml:referenceFormulation ql:JSONPath ;
    rml:iterator "$.contacts[*]"];
  rr:subjectMap [
    rr:template
        "http://example.com/{name}";
    rr:class foaf:Person];
  rr:predicateObjectMap [
    rr:predicate foaf:phone ;
    rr:objectMap [rml:reference "phone"]].
```

building and factory data for construction and facility models over their lifecycle. To enable information sharing and collaboration among users and software applications, the standard specifies a data schema to represent facilities with 3D models and their related metadata. The data schema accords to the EXPRESS data specification language by which industrial product data is defined (International Organization for Standardization, 2018). As an example, listing 2 contains an excerpt of the definition of a factory door in the IFC format. The graph-based structure contains information about the entity, such as height and width, and relationships to other building elements.

Listing 2: Extract from the EXPRESS definition for a `IfcDoor` entity in IFC 4.3.rc.2.

```
ENTITY IfcDoor
SUPERTYPE OF (ONEOF
  (IfcDoorStandardCase))
SUBTYPE OF (IfcBuiltElement);
  OverallHeight : OPTIONAL
      IfcPositiveLengthMeasure;
  OverallWidth : OPTIONAL
      IfcPositiveLengthMeasure;
[...]
END_ENTITY;
```

Software support for IFC is given, e.g., by the BIMserver software package (Beetz et al., 2010) that provides parsing and a Java API to use the contained entities as Java objects. The API is directly generated from the EXPRESS schema with the help of the Eclipse Modeling Framework[2]. To represent IFC data in RDF, the ontology *IfcOWL* (Pauwels and Terkaj, 2016) has been defined by Pauwels and Terkaj. It is directly related to the EXPRESS schema and includes a class for each entity type and a property for each relationship.

Several IFC data query languages have been proposed. They aim to provide shortcuts and abstractions from the IFC data model to make accessibility easier for authors. BimSPARQL is an extension of SPARQL that lets the user query IfcOWL data more conveniently as it defines special operators and geometric algorithms for common use cases (Zhang et al., 2018). The BIM query language *BimQL* (Mazairac and Beetz, 2013) is a new domain specific language whose syntax is similar to SQL and SPARQL. It defines shortcuts to hide the technicalities of the EXPRESS data model and allows binding of variables, aggregation and comparison of data. BIMserver comes with a simple filter language[3] based on JSON that can filter entities based on their type, attributes, and relationships. However, is does not support any data transformation such as binding of variables or arithmetic operators. Graph-based search algorithms have been proposed to evaluate a query on IFC data (Tauscher et al., 2016; Ismail et al., 2017).

2.3 UML Class Diagrams

To aid in model-driven development processes, the conversion of UML class diagram data to RDF or OWL is of particular interest to reduce modeling efforts. For example, relevant transformation rules to create ontologies in OWL are described (Vo and Hoang, 2020). Similarly, existing tools such as UMLtoOWL[4] allow for automized conversion. However, the available solutions require specific source formats that are mostly XML-based and thus already addressed by current RML mappers. In practice, domain experts design their class diagrams with end-user software such as draw.io or the yEd Graph Editor (redacted, project-internal survey), which provide their own data formats. An implementation converting visualizations of ontologies designed in Draw.io to RDF/OWL was given by Chávez-Feria et al. (Feria et al., 2021)

3 EXTENDING RML WITH ADDITIONAL SOURCE FORMATS

An overview of our methodology to extend RML with new source formats is depicted in fig. 1. The following five subsections cover these five steps in detail, respectively.

[2]https://eclipse.dev/modeling/emf/

[3]https://github.com/opensourceBIM/BIMserver/wiki/JSON-Queries

[4]https://www.sfu.ca/~dgasevic/projects/UMLtoOWL/

3.1 Analysis of the RML Specification

In this section, *document* refers to the input file to be mapped to RDF. *Mapping* refers to the resulting RML mapping file. We derive a general interface that needs to be implemented to access the data of the document. For this purpose, the functions I (Iterator) and E (Reference Extractor) define the semantics of two formal languages, i.e. they map strings to a meaning and are used to access the content of the document. I and E are format-specific, because query languages are. For example, I_{XML} and E_{XML} are defined using *XPath*. A document is composed of *records*. The iterator language I is used to specify the records to convert. Each record is converted into an RDF node during RML's iteration loop. Let D denote the set of documents and R_d denote the set of records contained in a document $d \in D$. I maps a document d to a subset of records using a string specified as `rml:iterator` in `rml:LogicalSource`. The iterator can be omitted for tabular input.

$$I(d,i) \in \mathfrak{P}(R_d),\ d \in D,\ i \in Strings$$

I maps a string and a document context to a list of records.

Records are still part of the document's data model and cannot be directly mapped to RDF. A *reference extractor E* extracts a list of strings from a record using a reference text (hereafter called *reference*) at various points during the iteration loop. These strings are inserted into RDF, e.g. as labels, predicates or URIs. Thus, the evaluation of a reference serves as the actual conversion between data models. It converts the input's data model to a list of strings that can be put directly into RDF.

$$E(r,e) \in \mathfrak{P}(Strings),\ r \in R_d,\ e \in Strings$$

E maps a string with context of a record to a list of strings.

With this proposal, RML is extended with IFC and Draw.io Class Diagrams by designing and implementing I_{IFC}, I_{CD}, E_{IFC} and E_{CD}.

3.2 Analysis of Source Code of Existing Mappers

For the purpose of choosing the best suited RML processor to fork and extend, existing processors have been analyzed regarding the interface described in section 3.1 can be identified in their source code and to what extend their architecture is suitable to be extended by further input formats. Here, RML Mapper[5] (reference implementation), RocketRML[6] and SDM-RDFizer (Iglesias et al., 2020) have been analyzed. In terms of extensibility and code quality, *RML Mapper* and *Rocket RML* perform similarly well and outperform the SDM-RDFizer. Interfaces for I and E can be identified in the source code, decoupling the logic of RML from support for different input formats. Since the reference implementation RML-Mapper is guaranteed to cover all of RML's features, it is chosen for the here discussed extension by creating a fork of the repository[7].

3.3 Extending RML with IFC

IFC is a structured file format, and entities, entity attributes, and links between entities are defined in the associated EXPRESS schema. The following section describes the implementation of IFC file support in RML Mapper. For this purpose, the new ontology *ifcrml* is introduced. To extend RML by a new format, as described in section 3.1, I and E for the new format must be implemented. The following sections describe design and implementation of I_{IFC} and E_{IFC}.

IFC Iterator. The iterator extracts records from the document, converting each into an RDF subject, ensuring its individual significance. In IFC, records correspond to entities defined in the EXPRESS schema. While IFC files contain thousands of entities, not all are relevant as subjects in the result. Query languages, including BimQL, are typically used to filter out relevant records, though textual queries are rare in practice. BIMserver supports a JSON-based filter language for selecting IFC entities, used in tools like *bimvie.ws* for interactive model viewing. While expressive, BimQL's limited development and adoption pose challenges. To address limitations, Java's full expressiveness is leveraged using the Function Ontology (FnO) to implement a filter function. This Java function receives entity lists and other parameters, returning processed and mapped IFC entities. FnO defines function signatures linked to Java methods, facilitated by ifcrml ontology for parameter assignment (De Meester et al., 2020).

IFC ReferenceExtractor. A ReferenceExtractor extracts text locally from a specific record. Each IFC entity contains various attributes that are specified in the EXPRESS schema. These attributes can be labels or identifiers of other entities for cross-referencing. A clean syntax has been created that ex-

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

[6]https://github.com/semantifyit/RocketRML

[7]https://github.com/PHochmann/rmlmapper-java

tracts text from an attribute by its name. For label attributes, the reference evaluates its text. If the attribute is a cross-reference to another entity the statement must be continued with an attribute of the referenced entity until it ends with a label. Attribute references are separated by a period. Additionally, a pseudo attribute `IfcType` has been implemented that evaluates the name of the EXPRESS class of the entity. This can be used to structure URIs of entities based on their classes (such as `rr:template "https://standards.buildingsmart.org/IFC/DEV/IFC2x3/FINAL/OWL/IFCType"`, which uses the `IFCType` attribute to define an URI template). The described functionality was implemented using BIMserver which operates on IFC files. Communication between our fork of RML Mapper and a BIMserver instance follows the architecture from fig. 2.

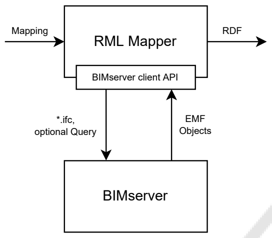


Figure 2: Architecture of our extended RML Mapper that communicates with a BIMserver to execute mappings for the IFC format. It transforms *.ifc files from a local file system to the BIMserver (Beetz et al., 2010) for parsing. We offer optional filtering via provided queries. Returning Eclipse Modeling Framework objects represent the queried IFC entities and are finally transformed to RDF.

3.4 Extending RML with Draw.io Class Diagrams

UML can express object-oriented structures using class diagrams. Classes are shown with their names, attributes and methods, as well as relationships between classes. *Draw.io* class diagrams can be exported as XML, so it would be possible to use the already existing implementation of RML for XML to convert information from class diagrams to RDF. Elements could be selected from XML using XPath. Because a class diagram is a generic graph, and XML documents are tree-structured, it is challenging to extract information directly using XPath. In addition, the semantics of a class diagram are not fully reflected in its serialization, e.g. which class an arrow points to. To overcome these problems, we propose a query language for class diagrams to hide the intricacies of XML serialization from the mapping author.

Class Diagram Iterator. This section outlines the design of I_{CD}, the iterator tailored to the class diagram input format. Identifying the records within a data model is critical to designing an iterator. In essence, an iterator accesses these records and facilitates their availability to the RML mapping engine. The data model of UML class diagrams is similar to a generic graph because classes can be connected by arrows indicating inheritance or dependencies. The key to a class diagram is the classes, their attributes, and the relationships between them. However, working only with classes as records may not be intuitive. For example, to map attributes that satisfy some condition X, mapping authors would first have to select classes that potentially contain those attributes. They could then use a reference such as ”`attributes.name where` X” to establish a link between the containing class and the labels of its attributes. Although this method allows multiple values to be extracted with a single reference, linking the class to other resources would require repeating the constraint in each reference. In addition, RML's joins could not later link the attributes together because only nodes that emerge from records can be joined, compromising the quality of the resulting linked data.

To address these challenges, a simplified data model for class diagrams is advocated, consisting of the record types *classes*, *attributes*, and *arrows* with a limited number of properties. The iterator is conceptualized as a context-free language.An excerpt of the producing grammar is shown in fig. 3.The start rule S specifies that each iterator consists of a *class selection* and, optionally, a prefix and a where clause.A class selection embodies a recursive statement that encodes a breadth-first search of the diagram, returning a set of selected classes.It serves as the basis for selecting either the classes themselves as records, their attributes, or arrows.The prefixes `attributes of` and `usages of/by` modify the record type.If the prefix is omitted, the classes themselves are extracted as records.A *WhereClause* allows filtering based on individual properties of the record. A simple example is `usages by Student` to select all arrows coming from *Student* classes.

Implementation. Draw.io class diagrams can be exported as XML files. Since the meaning of the document is not well reflected in its syntax, the implementation was comparatively complex. After using a standalone XML parser, the deserialized data must first be converted to the target data model. XML has a tree structure, so IDs and cross-references are used to represent the arbitrary graph of a class diagram. In addition, Draw.io's serialization to XML is more fo-

```
           S →[Prefix] ClassSelection [WhereClause]
      Prefix →attributes␣of␣
      Prefix →usages␣ (of␣ | by␣)
 WhereClause →␣where␣ Restriction
 Restriction →Reference PropOp String
              [PropOp Restriction]
```

Figure 3: Excerpt from the production rules of the context-free grammar that produces the iterator language.

cused on the graphical representation of the diagram using shapes, rather than the semantics of the data being represented. For example, attributes and functions within a class rectangle are distinguished by comparing their y-coordinate to the y-coordinate of the line separating them. Serialized documents created by domain experts contain errors such as missing or incorrect arrow attachments, which are geometrically reconstructed in post-processing.

3.5 General Workflow to Extend RML with New Formats

This section addresses the research question of this work by proposing a a workflow for extending RML with new source formats.

Feasibility Analysis. The criteria for assessing whether a given format can be supported by RML in relation to the semantics of the data it contains, while the syntax influences the technical implementation and effort. The data model of the source must define mutually delimitable, enumerable records that contain meaningful data to be converted into an RDF resource. If individual records can be identified, it is necessary to decide how to convert parts of a record into a textual representation. Records should be sufficiently similar to be treated by a maintainable amount of TriplesMaps. For each record, similar RDF subgraphs are created. Variation of these subgraphs are generated by joins, non-constant predicates and objects that can be defined in `rr:PredicateMap`s and `rr:ObjectMap`s. However, the reference texts that are used to extract a string from a record are always the same within each TriplesMap. Records should be kept as simple as possible, to avoid too much variety, and to keep references simple. At the same time, they should be complex enough to carry meaning themselves. If a logical concept is spread over several records, it must be reconstructed in the mapping via joins and cannot be identified with a single URI. Limited variety and semantic independence are therefore conflicting goals.

Design of *I* and *E*. The iterator language must provide means for navigating within a document as well as filtering records based on their shape. A trade off between expressive strength and ease of use is needed. In the case of graph-based data models, complex expressions of navigation patterns on the data might be needed to retrieve meaningful records. The reference extractor language should be kept as simple as possible. To improve maintainability, a reference is only an attribute name without requiring much navigation within the record.

Implementation Phase. The complexity of the implementation phase depends on the serialization of the documents and the complexity of aforementioned languages. First, a parser is needed to deserialize the document. The result of the parsing is an object of a programming language, whose structure reflects the data model of the source format. It provides access to the contained data. Records are selected from these objects using the iterator. If records are selected in such a way, that they cannot be derived directly from the syntactic structure of the document, the parsed data must be post-processed. Afterwards, the reference extractor which takes a record and extracts text locally can be implemented.

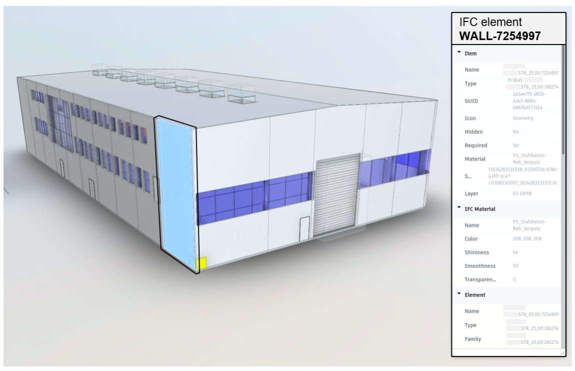
4 EVALUATION

The evaluation of the two demonstrated extension formats for RML includes several aspects: Author-friendliness evaluates whether all mapping components are intuitive and conversion via RML is time-efficient. Expressiveness evaluates whether all information can be extracted and converted. Performance evaluates whether large files can be processed quickly. All run times were measured using Apache Commons' `StopWatch` on an AMD Ryzen 7 PRO 3700U @2.7GHz, Linux 5.10.84-1-MANJARO with OpenJDK 17.0.1.

4.1 Conversion of IFC

In the following section, the IFC implementation is evaluated using an IFC file provided by the use case. See fig. 4 for a visual representation of the file, which consists of a model of a demonstration factory. The goal of this use case is to map all coverings (e.g., windows, doors) of a single wall that is specified by its `GlobalID`-attribute to RDF. Each covering should be converted to a named node and additional labels specifying its entity-properties, such as name, dimensions, type of door, and so on.

The lack of expressiveness in the BIMserver filter language requires entity selection through a Java



IFC element
WALL-7254997

Item

Name	[redacted] [redacted]:STB_25,00:7254997
Type	IfcWall: [redacted] [redacted]:STB_25,00:381274
GUID	1a0ae7f5-283b-4143-868a-6867bd1731b4
Icon	Geometry
Hidden	No
Required	No
Material	FO_Stahlbeton-Roh_Verputz
S...	t1634283131018_615b0716-638c-43f0-ace7-c03b9530ff67_1634283131372.ifc
Layer	EG OKRB

IFC Material

Name	FO_Stahlbeton-Roh_Verputz
Color	208, 208, 208
Shininess	64
Smoothness	50
Transparen...	0

Element

Name	[redacted] [redacted]:STB_25,00:7254997
Type	[redacted] [redacted]:STB_25,00:381274
Family	[redacted] [redacted]:STB_25,00:381274

Figure 4: Visual representation of an experimental factory plan provided by a use case as an IFC file.

function, which is challenging because it requires detailed knowledge of the EXPRESS schema, especially for complex cross-references such as *IfcRelVoidsElement*, *IfcOpeningElement*, *IfcRelFillsElement*, and *RelatedBuildingElement*. However, these entities contain few attributes, and obtaining additional information about windows or doors increases the complexity of the Java function. While the current state of the art approach involves converting the entire IFC model to IfcOWL and validating it using SPARQL or SHACL, this results in large mapped files compared to the proposed solution which, while less user friendly, leverages existing technology without having to learn a new language with limited versatility. The measured runtimes are dominated by the check-in of the IFC files to the BIMserver and take up to 46 seconds for an 8 MB file with 140253 entities, including conversion into Java objects. Further processing takes milliseconds, even though several thousand objects from the Eclipse Modeling Framework are iterated to select the appropriate objects. Caching these objects and avoiding multiple check-ins of a file could reduce the runtime even more.

The proposed methodology has demonstrated its effectiveness in partially converting IFC files. Extraction of records for RDF mapping occurs within milliseconds, allowing for deeper analysis of use-case relevant information not possible with full representations such as IfcOWL. This solution opens up new possibilities for analyzing building and plant data with Semantic Web tools, such as validating planning scenarios with reasoner-based evaluations or inferring implicit information (Beetz et al., 2021; Burggräf et al., 2021). Proper industrial application may be limited to experts because of limitations such as the weak documentation of the IFC query language, the approaches still in the experimental stage, the limited expressiveness of the BIMserver filter language, and the complexity of the Java filter function.

4.2 Conversion of Class Diagrams

This section uses a real-world use case to evaluate support for class diagrams, with the goal of mapping the diagram to RDF/OWL by describing an equivalent ontology. A comparison can be made with an ontology created in an ontology editor such as Protégé. While the semantics of the diagram is different from the standard UML class diagram, it has strong similarities as a metamodel with a class concept. UML classes correspond to OWL classes, arrow relationships correspond to ObjectProperties, and attributes correspond to DatatypeProperties, with arrows labeled with corresponding ObjectProperty names and attributes specifying DatatypeProperty names. The RML mapping consists of five TriplesMaps and took about an hour to formulate, requiring only basic OWL knowledge, with a runtime of about 100 milliseconds. The use of RML for class diagram conversion proved successful in our application scenario, minimizing the end-user workload by eliminating the need for manual ontology creation in Protégé. However, the output is limited to components of the ontology that can be described by recognized UML syntax elements, excluding ontology meta-information from the output.

5 CONCLUSION AND FUTURE WORK

In this paper, we have extended the RML Mapper to support two new formats and demonstrated a generic workflow for RML extension. Our exploration of IFC file conversion demonstrated the practical applicability of our methodology, albeit the complexity of the IFC data schema. Support for UML class diagrams demonstrated author-friendly mappings using an iterator language that reflects the semantics of the source data model. Lessons learned from both example implementations influenced the development of a generic workflow for extending RML.

Our work has not only extended the functionality of RML, but also addressed specific challenges in practical implementations. The proposed enhancements, such as the introduction of the `rml:condition` property and considerations for query language implementation, pave the way for further advances in semantic data integration. The evolution of RML to support a broader range of formats while adhering to the principles of linked data underscores our commitment to advancing data interoperability.

ACKNOWLEDGMENTS

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC-2023 Internet of Production – 390621612.

Funded by the FAIR Data Spaces project of German Federal Ministry of Education and Research (BMBF) under the grant numbers FAIRDS05, FAIRDS14 and FAIRDS15.

REFERENCES

Beetz, J., Pauwels, P., McGlinn, K., and Tormä, S. (2021). *Linked Data im Bauwesen*, pages 223–242. Springer Fachmedien Wiesbaden, Wiesbaden.

Beetz, J., van Berlo, L., de Laat, R., and van den Helm, P. (2010). Bimserver.org – an open source ifc model server. In *Proceedings of the CIP W78 conference*, page 8.

Burggräf, P., Dannapfel, M., Ebade-Esfahani, M., and Scheidler, F. (2021). Creation of an expert system for design validation in bim-based factory design through automatic checking of semantic information. *Procedia CIRP*, 99:3–8. 14th CIRP Conference on Intelligent Computation in Manufacturing Engineering, 15-17 July 2020.

Das, S., Sundara, S., and Cyganiak, R. (2012). R2RML: RDB to RDF mapping language. Technical report, W3C. Retrieved from https://www.w3.org/TR/r2rml/, on 05.05.2022.

De Meester, B., Seymoens, T., Dimou, A., and Verborgh, R. (2020). Implementation-independent function reuse. *Future Generation Computer Systems*, 110:946–959.

Dimou, A., Sande, M. V., De Meester, B., Heyvaert, P., and Delva, T. (2020). RDF Mapping Language (RML). Technical report, IDLab - imec - Ghent University. Retrieved from https://rml.io/specs/rml/, on 05.05.2022.

Dimou, A., Vander Sande, M., Colpaert, P., Mannens, E., and Van de Walle, R. (2013). Extending r2rml to a source-independent mapping language for rdf. In *International Semantic Web Conference (Posters & Demos)*, volume 1035, pages 237–240.

Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., and Van de Walle, R. (2014). Rml: A generic language for integrated rdf mappings of heterogeneous data. In *LDOW*.

Feria, S. C., García-Castro, R., and Poveda-Villalón, M. (2021). Converting UML-based ontology conceptualizations to OWL with chowlk. In *ESWC2021 Poster and Demo Track*.

Heyvaert, P., Dimou, A., Herregodts, A.-L., Verborgh, R., Schuurman, D., Mannens, E., and Van de Walle, R. (2016). Rmleditor: A graph-based mapping editor for linked data mappings. In *European Semantic Web Conference*, pages 709–723. Springer.

Iglesias, E., Jozashoori, S., Chaves-Fraga, D., Collarana, D., and Vidal, M.-E. (2020). Sdm-rdfizer: An rml interpreter for the efficient creation of rdf knowledge graphs. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 3039–3046.

Iglesias-Molina, A., Arenas-Guerrero, J., Delva, T., Dimou, A., and Chaves-Fraga, D. (2022). RML-star. Technical report, W3C Knowledge Graph Construction Community Group. Retrieved from https://kg-construct.github.io/rml-star-spec/, on 05.05.2022.

International Organization for Standardization (2018). Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries — Part 1: Data schema. Standard, International Organization for Standardization, Geneva, CH.

Ismail, A., Nahar, A., and Scherer, R. (2017). Application of graph databases and graph theory concepts for advanced analysing of bim models based on ifc standard.

Lipp, J., Gleim, L., and Decker, S. (2020). Towards Reusability in the Semantic Web: Decoupling Naming, Validation, and Reasoning. In *Proceedings of the 11th Workshop on Ontology Design and Patterns*. CEUR Workshop Proceedings.

Malcolm, A., Werbrouck, J., and Pauwels, P. (2021). Lbd server: Visualising building graphs in web-based environments using semantic graphs and gltf-models. In Eloy, S., Leite Viana, D., Morais, F., and Vieira Vaz, J., editors, *Formal Methods in Architecture*, pages 287–293, Cham. Springer International Publishing.

Mazairac, W. and Beetz, J. (2013). Bimql – an open query language for building information models. *Advanced Engineering Informatics*, 27:444–456.

Pauwels, P. and Terkaj, W. (2016). Express to owl for construction industry: Towards a recommendable and usable ifcowl ontology. *Automation in Construction*, 63:100–133.

Ravindra, P., Hong, S., Kim, H., and Anyanwu, K. (2011). Efficient processing of RDF graph pattern matching on MapReduce platforms. In *Proceedings of the second international workshop on Data intensive computing in the clouds - DataCloud-SC '11*. ACM Press.

Schuh, G., Prote, J.-P., Gützlaff, A., Thomas, K., Sauermann, F., and Rodemann, N. (2019). Internet of production: Rethinking production management. In Wulfsberg, J. P., Hintze, W., and Behrens, B.-A., editors, *Production at the leading edge of technology*, pages 533–542, Berlin, Heidelberg. Springer Berlin Heidelberg.

Tauscher, E., Bargstädt, H.-J., and Smarsly, K. (2016). Generic bim queries based on the ifc object model using graph theory. In *Proceedings of the 16th International Conference on Computing in Civil and Building Engineering, Osaka, Japan*, pages 6–8.

Vo, M. H. L. and Hoang, Q. (2020). Transformation of uml class diagram into owl ontology. *Journal of Information and Telecommunication*, 4(1):1–16.

Zhang, C., Beetz, J., and Vries, d. (2018). Bimsparql: Domain-specific functional sparql extensions for querying rdf building data. *Semantic Web*, pages 1–27.