2024 European Conference on Computing in Construction
Chania, Crete, Greece
July 15-17, 2024

HARMONIZING INPUT DATA FOR DIGITAL BUILDING TWIN: ENERGY EFFICIENCY USE CASE

Audrey Bouet[1], Nicolas Bus[1], Stoyan Danov[2], Eloi Gabaldon[2], Edgar Martínez-Sarmiento[2], Nicolas Pastorelly[1] and Guillaume Picinbono[1]
[1] C.S.T.B, Sophia Antipolis, France
[2] CIMNE, UPC Campus Terrassa, Terrassa (Barcelona), Spain

Abstract

In this article, we present a "Harmonizer module", based on RML mapping language, SPARQL[1] and RDF, which allows importing heterogeneous data sources into a unique Building Energy Analysis-oriented ontology. This work is being performed in the context of the H2020 BIGG project[2]. The Reference Architecture Framework designed by the project leverages the specific ontology which is used as a harmonized input for the unique BIGG Data Analytics Toolbox that creates various building-energy related insights. The Harmonizer module receives inputs from different kind of building energy analysis data (such as: location, building structure, sensor data, time series), formats them and maps them to the concepts defined in the BIGG ontology. This paper will introduce the objectives of the BIGG project, a description of the BIGG ontology before describing the workflow and the functionalities of the Harmonizer module, it's implementation and current results.

Introduction

Prominent parts of the National Energy Efficiency Action Plans of the EU countries are dedicated to strategies for increasing the energy efficiency in buildings. These plans include overviews of countries' building stock, policies for stimulating the renovation rate, including estimates and projections about the energy savings and carbon dioxide emission reductions from applications of different measures.

The increasing availability of data from smart meters and digital sensors in buildings has enormous potential to support the more efficient use of buildings. However, this requires the capability to collect, harmonize, and jointly process data from a variety of different sources. This problem is tackled by the H2020 BIGG project which has the objective to substantially facilitate the deployment of energy-related big data analytics for buildings and to support a large variety of business cases. BIGG develops a Reference Architecture and an Analytics Toolbox, supported by a common data model (BIGG Ontology) and a Harmonizer module for aligning data from different sources and providing them as an input for the common Analytics Toolbox.

This paper focuses on the presentation of the Harmonizer module designed for the conversion of external heterogeneous data formats into RDF complying with a targeted ontology. It has been created and exemplified in the context of the BIGG ontology for Energy Oriented Building Digital Twin but can be applied to any other ontology. The BIGG ontology is briefly addressed, but a detailed discussion on it is out of the scope of this paper.

Related works

The BIMERR (Boutouni et al., 2021) project is a research project aimed at developing and promoting the use of Building Information Modelling (BIM) in the construction industry. This project provides a tool called BIMERR Interoperability Framework (BIF) to ensure data exchange among applications. BIF utilizes mechanisms that enable semantic and syntactic interoperability. BIF uses the RMLMapper library (Dimou et al.,2014) and benefits from RML mapping to convert and align heterogeneous data into interoperable RDF data. Unfortunately, BIF neglects how to convert data based on an external taxonomy to a common interoperable taxonomy.

Similar attempts to populate Digital Building Twin ontology from heterogeneous data have been initiated through the COGITO platform (Katsigarakis et al., 2022). COGITO mainly uses transformation tools based on RML mapping to transforms data concerning building description such as IFC and store them in a knowledge graph. Data collected by sensors are stored separately in a dedicated time series database. This approach doesn't benefit from knowledge graph to ensure interoperability between static data and timestamped observations.

Several solutions have been proposed to map semi-structured data to RDF. The SweoIG taskforce has built a list of implementations also called RDFizer (Iglesias et al., 2020). Most of them are dedicated to a specific format or model. We focused our analysis on those who reach the project requirements: extensible language with call to user defined functions, model agnostic and fully compliant with W3C recommendations.

RML is one of the approaches that allows mapping between various sources toward RDF. RML is an extension of the R2RML vocabulary to describe logical

[1] SPARQL. https://www.w3.org/TR/sparql11-query/
[2] Building Information aGGregation, harmonization and analytics platform, H2020 funded project. https://www.bigg-project.eu/

sources. It generates RDF from JSON, CSV, or XML sources.

SPARQL-GENERATE (Lefrançois et al., 2017) is based on a query language that queries the combination of an RDF dataset and a set of documents. Various formats can be supported thanks to the extensible set of SPARQL 1.1 binging functions and SPARQL-Generate iterator functions. SPARQL-Generate is appropriate for engineers that are familiar with SPARQL query language.

OpenRefine[3] (formerly Google Refine) helps make sense of tabular data through a set of tools to clean, transform and merge several datasets together. RDF Refine is an extension of Open Refine meant to align the tabular data loaded in Open Refine with existing ontologies, while reconciliating the obtained URIs with third party RDF data sets. The data management part of Open Refine as well as the alignment towards RDF in RDF Refine are fully performed through a Web user interface. Unfortunately, RDF Refine does not allow to map CSV input files to RDF. Open Refine can also handle JSON data but does not implement multi-level iterators that make the mapping complicated when processing deep structures.

The BIGG H2020 Project

General presentation of the BIGG project

The BIGG project aims at demonstrating the application of big data technologies and data analytic techniques for the complete building life cycle of more than 4000 buildings in 6 large-scale pilot test beds. The proposed solutions will be deployed and tested as a pilot with country validation of at least two business scenarios in Spain and Greece. For that, the project has developed: 1) The Open Source Architecture (BIGG Data Reference Architecture 4 Buildings) for collecting, funneling, processing and exchanging data from different sources (smart meters, sensors, BMS, existing data sets); 2) An interoperable buildings data specification (the BIGG Ontology); 3) An extensible, open, cloud-compatible service (BIGG Data Analytics Toolbox of service modules) for batch and real-time analytics that supports effective decision and policy making.

In that context the reference architecture framework (RAF) (Pastorelly et al., 2022) has been designed (see Figure 1) and implemented, and the BIGG consortium has developed open source BIGG components enabling big data analytics for different business cases: Benchmarking and building portfolio management; Energy certification in residential and tertiary buildings; Building life-cycle data interoperability: from planning to renovation; Energy Performance Contract support; Building optimisation for occupants; Flexibility potential of buildings.

Presentation of the overall architecture

The Reference Architecture Framework (RAF) can be seen as a dedicated implementation of the Digital Twin concept. The detailed RAF concept is presented in Figure 1.

The RAF consists in a pipeline of several components which can be orchestrated in different ways. The first step uses BIGG ingestion modules (1) that are software components acquiring specific data (2) from different custom sources using standard protocols like HTTP or MQTT. These ingested data are stored in data lakes for later processing and/or directly pushed through an event message bus like Apache Kafka[4] to the harmonization module (3). The harmonization module is flexible enough to only require a standardized mapping file (4) to enable conversion of the initial specific data to the BIGG Standard Data Model 4 Building (5) required as input by the BIGG Data Analytics Toolbox (6). Harmonized data are instances of the BIGG data model, expressed in RDF format, containing only information needed for a specific use case. They can be stored in "harmonized data lakes" (ex: triple stores) and/or directly pushed through an event bus to the toolbox. The toolbox leverages several data analytics and AI/ML curated technologies to process the building/energy related data and produce insights output and push them back into the BIGG Standard Data Model 4 Building (7). The Toolbox uses machine learning techniques to produce valuable predictions. The knowledge created by the toolbox can be stored in "harmonized lakes shores" and is used by specific dashboard and external system software components (8) that will transform back the harmonized data into the data formats required by external dashboards or systems.

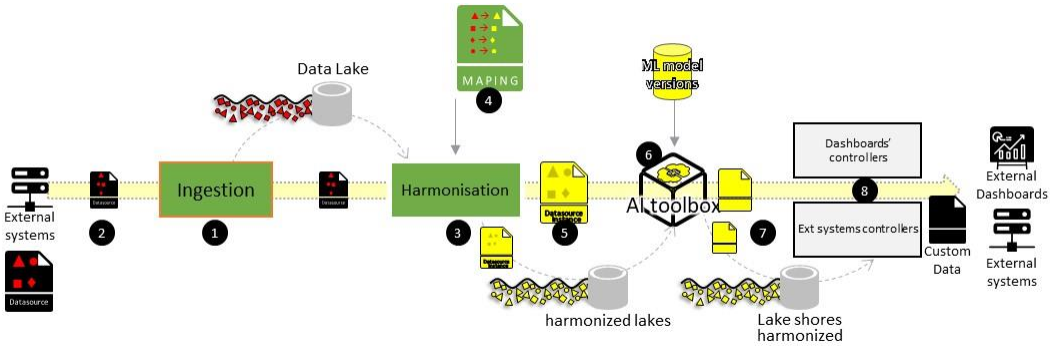

Figure 1: BIGG RAF detailed.

[3] http://openrefine.org/

[4] https://kafka.apache.org/

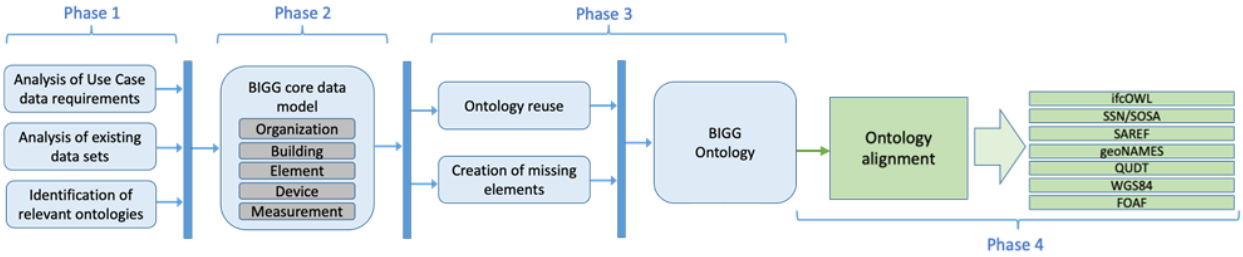

Figure 2: Methodological process for development of the BIGG Ontology.

BIGG Ontology for Energy Oriented Building Digital Twin

The creation of the BIGG ontology is rooted in the necessity to ensure harmonised input for the common Analytics Toolbox that allows developing data-driven services by using heterogeneous datasets generated within different business activities in the built environment. Conceived initially as a common data model developed in UML, it evolved to a graph data structure described in the Web Ontology Language (OWL) to benefit from the standardisation and data processing power offered by the semantic technologies.

The development process of the BIGG ontology followed a methodological approach similar to that suggested in (Radulovic et al., 2015). It is schematically illustrated in Figure 2 and has four phases, including the ontology alignment as a last step (once validated by the implementation of the business cases, BIGG ontology was aligned with already existing ontologies concepts coming from ifcOW, SAREF, SSN/SOSA, BOT, etc, and properly documented and published).

Overall, the BIGG ontology provides a standardised way to describe and organise data for buildings and applications requiring comparison and analysis of data related to performance of buildings, their systems, elements, and their usage.

Table 1: Main classes of the BIGG Ontology

Class	Description
bigg:Building	A building for which data is provided
bigg:BuildingSpace	A space that can represent one or more rooms, floors, or zones of a Building
bigg:Element	Generic element of the building
bigg:Device	Any meter, sensor, or actuator that can capture a signal or assume a state
bigg:Sensor	A collection of Measurements from the same Device
bigg:Measurement	Any time series record registered by a Device.

Currently, the BIGG ontology contains 100 classes, 181 DataProperties and 108 ObjectProperties. Table 1 provides a description of some of the main classes.

The BIGG Harmonizer

The Digital Building Twin involves a multitude of data models and formats coming from different sources such as open data providers, asset management software and sensors. Regarding data formats, relational databases and XML are still present, Open Data portals heavily rely on CSV, and web APIs on JavaScript Object Notation (JSON). The RDF data model is used as a federated model to reach semantic interoperability and querying of data having heterogeneous formats (Michel, 2017).

Presentation of the overall concept and input data

The BIGG harmonizer aims at converting any data in scope of energy performance of building, that fit with the BIGG Ontology, into RDF. Data that are not needed for use cases are filtered. As possible inputs in our approach, we focus on JSON for data coming from sensors (timeseries) and asset management software (building and systems description) as well as CSV coming from Open Data (geolocation) and RDF (data aligned on SAREF, SOSA or IFC) for alignment with existing ontologies (see Figure 3).

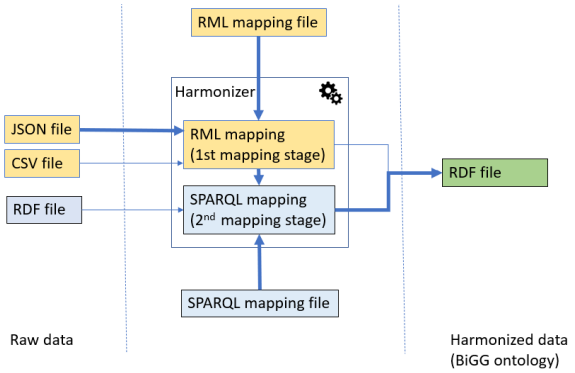

Figure 3: General workflow of the Harmonizer module

The harmonizer must generate data respectful of W3C recommendations and BIGG Ontology by providing the following functionalities: (F1) Converts JSON files into RDF compliant with BIGG Ontology; (F2) Converts CSV files into RDF compliant with BIGG Ontology; (F3) Aligns Data described using standard ontologies covering the same scope (ifcOWL, SAREF, SSN/SOSA, GeoNames, QUDT, WGS84, FOAF); (F4) Allows to map

input objects with BIGG classes; (F5) Allows to map input attributes with BIGG data properties; (F6) Interprets implicit links between objects through object properties; (F7) Reconciles input values with open registers, BIGG taxonomies and enumerations; (F8) Materializes data context and generate 5-stars data. The next parts describe how the harmonizer provides these functionalities.

Converting input formats to RDF (F1, F2)

The Digital Building Twin necessitates high-quality data known as 5-star data, as originally introduced in (Berners-Lee et al., 2009). Frequently, the raw data obtained by the Digital Building Twin necessitates cleaning and enrichment to enable full interoperability. The levels established by Tim Berners-Lee are: availability (level 1), structuring (level 2), openness (level 3), universal identification (level 4), and linkage to other data (level 5). In our case, raw data is mainly provided as JSON, but the same methodology could be applied to CSV-formatted data. In this project, the 5-star level is achieved by following the RDF specifications.

First stage of the harmonizer extracts inputs from a JSON file and a RML mapping file in order to produce a first RDF file matching the BIGG scope (see Figure 3). The RML mapping files are composed of iterators to identify data sources and class mapping declarations that refer to iterators. Mapping files can either be produced manually or by using dedicated tools that simplify the mapping process such as RMLEditor[5] or Matey[6]. This procedure is detailed in the Implementation section.

The RML processing is based on the RMLMapper library developed by Ghent University. Firstly, the relevant parts of the JSON are identified according to iterators declared in the mapping file. Then the mapping itself is performed by creating, for each mapping data, an instance of the mapped ontology class, by generating a URI, and by generating triples according to data properties. Each URI is generated from the context (provider or building owner and building) and the local ID defined in the JSON.

Interpreting implicit link (F6)

The underlying structures of JSON are arrays and trees. Relations between objects are implicitly declared by using the native parent-child relation provided by the tree structure. As BIGG Ontology defines several relations, the harmonizer needs to match the parent-child relation between two JSON objects with one of the BIGG Ontology relation. Figure 4 illustrates how two parent-child relation from the same JSON file can be interpreted as distinct relations in the BIGG Ontology.

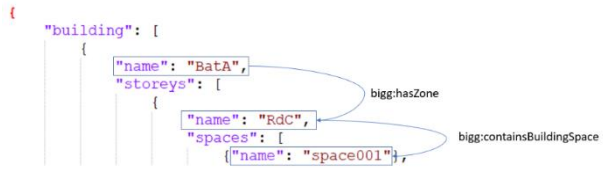

Figure 4: Sample of JSON file describing building structure.

The materialization of implicit links is also ensured by the harmonizer's first mapping stage. Object properties (as declared in the mapping file) map instances with each other's by using their URI. As for instance generation, JSON samples corresponding to the subject and the object of a triple are identified using absolute or relative expressions. As URI are always generated by using the same convention, URIs remain consistent. This way, URIs generated when populating instances match with URIs generated when building relations between instances.

Reconciliating data with enumerations (F7)

The BIGG harmonizer uses GeoNames as a universal register for public buildings. By reconciling GeoNames buildings with BIGG building descriptions, the project allows for the federation of data provided by building owners with data provided by the Open Data community. This is achieved by querying the GeoNames endpoint to retrieve the building's public URI, official name, and address, using the building location as a reference. The harmonizer's first stage generates a graph, which is then used to execute a federated SPARQL query in the second stage to replace the temporary building URI with the GeoNames URI and insert complementary information requested from GeoNames. Alternative public endpoints can be used to provide data on building height, roof inclination, and envelope materials.

The BIGG Ontology is a valuable resource that provides taxonomies for the classification of various objects, including buildings, spaces, and sensors. Within the context of building usage classification, an excerpt of the building usages taxonomy is presented in Figure 5, which outlines the primary services provided by a building.

In situations where the classification of raw data differs from the BIGG classification, the harmonizer's second stage is used to translate between the two. For example, the French building classification term "Université" must be converted to its corresponding term in the BIGG taxonomy, namely "University".

Property	Taxonomy 1st level	Taxonomy 2nd level
buildingSpaceUseType	EducationAndResearch	ExtracurricularEducationCenter
		Laboratory
		MilitaryOrPoliceAcademy
		OtherEducationAndResearch
		Preschool
		PrimarySchool
		ResearchCenter
		SecondarySchool
		University
		(empty)
	Healthcare	DayCareHospital
		HomelessShelter

Figure 5: Sample of one of the BIGG taxonomies classifying building space usages.

A thesaurus is embedded in the BIGG ontology. It employs the SKOS ontology to describe taxonomies (skos:scheme), translations (rdf:label "Université"@fr, "University"@en) and cross-taxonomy semantic links

[5] RMLEditor, https://rml.io/tools/rmleditor/

[6] Matey, https://rml.io/yarrrml/matey/#

(<medical center> skos:match <hospital>). In the harmonizer's second stage, the reconciliation process is accomplished through a federated query that utilizes the thesaurus to translate from the taxonomy used in the raw data to the harmonized taxonomy.

Materializing data context (F8)

When describing energy consumption as JSON time series (Figure 6), the contextual information regarding the source (Who), location (Where), and method (How) of measurement is often overlooked or indirectly specified


```
{
    "_id" : "60f7fcd6e594e218826ca9f2",
    "datetime" : "2021-07-20T21:00:00.000Z",
    "cups" : "ES0031406110149001EJ0F",
    "consumptionKWh" : 12.0,
    "obtainMethod" : "Real"
}
```


Figure 6: Example of JSON time series missing building or provider context

in the file name. A standardized approach is required to extract and retrieve this contextual information from the pre-existing database or other data sources. The identification of data providers and building owners can be incorporated into the mapping file (in the SPARQL mapping), which can be further utilized by the harmonizer to enhance the graph. In instances where the timestamp is absent, the current date and time can be used to trace the incoming data. As an additional improvement, the harmonizer can also extract pertinent data, such as building and sensor identification, from the file name to further augment the graph.

Aligning data (F3)

In addition to the raw data collected by the Digital Twin as a JSON file, the harmonizer enables the ingestion of RDF data that implements standardized ontologies.

The BIGG Ontology has been aligned with several existing ontologies related to Digital Building Twins, such as BOT (for building topology description), SAREF (for energy system description), and SSN/SOSA (for sensor and timeseries description).

This alignment consists of triples that declare corresponding classes and corresponding predicates from one ontology to the other. The data alignment is generally performed on a given opportunity via a semantic reasoner embedded in the triple store but, if necessary, the harmonizer can materialize class and properties alignments by substituting the original vocabulary with BIGG vocabulary.

The BOT ontology is a minimal ontology that defines relationships between the sub-components of a building. It has been suggested as an extensible baseline for use with more domain-specific ontologies, following general W3C principles of encouraging reuse and keeping the schema no more complex than necessary. The BIGG Ontology defines some classes that are equivalent to BOT classes. For example, bigg:BuildingElement, which describes building components, is equivalent to bot:Element, and bigg:Zone is equivalent to bot:Zone. With respect to object properties, bigg:hasZone is defined as an equivalent property to bot:containsZone, and bigg:containsBuildingSpace is an equivalent property to bot:hasSpace.

SAREF is an ontology supported by the ETSI SmartM2M standard that enables interoperability among IoT projects and can be extended to any IoT vertical domains, such as smart buildings or energy. The BIGG Ontology reuses a few SAREF concepts, such as saref:Device, saref:Sensor, and saref:UnitOfMeasure, to describe sensor networks and time series. In the BIGG Ontology, a sensor is located (bigg:isContainedInSpace) in a space, observes a building element (bigg:featureOfInterest), and provides measurements (bigg:hasMeasurements).

There are two ways to extend the harmonizer's alignment capabilities. The first way is to extend the ontology with new class and property equivalences. The second way is to add SPARQL mapping queries that will convert classes and properties from one vocabulary to another (2nd mapping stage in Figure 3).

Implementation

The harmonizer is a generic way to convert any building energy related data into BIGG digital twin compliant data. As it would be too much effort to code serializers for each kind of source by using generic programming language, the harmonizer is based on the RML mapping language that simplify mapping specification for data providers.

Choice of the mapping language

The BIGG project focuses on RML as a mapping language for the following technical reasons: 1) RML is based upon RDF that is consistent with the harmonized data; 2) RML mappings are reproducible and maintainable; 3) RML is an extension of a W3C specifications: R2RML (Das et al., 2012); 4) RML is not limited to one type of input format and can easily handle JSON, CSV and XML format. But mostly, in the scope of the BIGG project, it was important that the mapping files can be generated (or at least updated) by non-informatician persons responsible for the different business cases. In this perspective, we have concluded that RML was the best option, thanks to the YARRRML language and Matey tools that offers the most suitable solution of our project. To this, we added a second mapping stage based on SPARQL to align URIs controlled values and taxonomies.

Development of the mapping files

YARRRML[7] is a human-readable mapping language developed by the University of Ghent, it allows to define rules and to convert them into RML or R2ML mapping language. Therefore, the generated RML mapping file contains rules to be used to transform input data into an RDF format. The web-based tool named Matey offers the

[7] https://rml.io/yarrrml/spec/https://rml.io/yarrrml/spec/

Figure 7: Example of a JSON input file (top left) with the corresponding YARRRML mapping (right) and the RDF file created by the mapping (bottom left)

possibility to write YARRML rules in a user-friendly way.

Matey user interface consists of 4 windows. Top left window contains input data (for example a JSON file). Top right window is the YARRRML editor, where user can write the YARRRML mapping script. Pushing the "Generate RML" button will automatically translate YARRRML script to RML and fill in the bottom right windows. Finally, pushing the "Generate LD" button will fill in the bottom left window with the RDF file resulting from the mapping of the input data using RML rules.

In the case of the BIGG harmonizer, Matey is used to develop and test the mapping rules in YARRRML language, generate RML rules and export them into a file. The YARRML mapping file is composed of three main parts (see Figure 7):

The declaration of prefixes: Some prefixes and namespaces are predefined by default in YARRRML, and a set of customs ones can be added by adding the '*prefixes'* collection to the root of the document (for example prefixes for the BIGG Ontology).

The declaration of sources:

Under the tag "sources" are listed all the "entry points" into the input data. Each entry point is defined by the file name, the format and an iterator acting as a regular expression to filter data. For example:

- Iterator: "$.building[*]" will iterate over all "building" entities in the file.
- Iterator: "$..contents[?(@.type=="SITE"&& @.categoryCode=="BUILDING")]" will iterate over all "contents" entities at any level in the JSON tree, and which have specific values of "type" and "categoryCode" properties.

The mapping rules:

Finally, the '*mappings'* collection contains the declaration of the mapping rules explaining how to translate the current entity pointed by the iterator into triples (subjects, predicates, objects) of the BIGG Ontology. For the example of Figure 7, the mapping named "building" takes as an input source the iterator "data1", and instantiate an object named "$(name)" as a bigg::Building, which as a bigg::name egal to "$(name)" and has a relation to a space declared as "space_$(name)". Instantiation of the bigg::BuildingSpace will be done in the next mapping rule.

Providing the harmonizer as a service

SDM-RDFizer is an interpreter of mapping rules that allows the transformation of (un)structured data into RDF knowledge graphs. The current version of the SDM-RDFizer assumes mapping rules are defined in the RDF Mapping Language (RML). The SDM-RDFizer

```
python harmonizer.py --input inputFile [--mapping RMLFile] [--sparql SparqlFiles] [--output outputFilename]
```

Figure 8: Command line calling the Harmonizer module.

implements optimized data structures and relational algebra operators that enable an efficient execution of RML triple maps even in the presence of big data. SDM-RDFizer is able to process data from heterogeneous data sources (CSV, JSON, RDB, XML) processing each set of RML rules (TriplesMap) in a multi-thread safe procedure. RocketRML (Simsek et al., 2019) is also an implementation of the RML mapper specification based on NodeJS. Whereas RocketRML appears to be faster than RMLMapper library, it only supports a subset of the RML specification that is needed for our use cases.

RMLMapper developed in java (*rmlmapper.jar*) is the reference implementation of the RML mapper specification as it covers all the features. Furthermore, it is more widely used and benefits from a more active community regarding version control insights (Heyvaert et al., 2018). For those reasons, RMLmapper is the one used in BiGG to process RML mapping rules on a selected dataset to generate an RDF document. The integration of this library into a Python module allows to experiment the harmonization of input data from multiple sources automatically. The Python module created to harmonize input data with the BIGG Ontology is composed of two stages, the first one is related to the RML mapping file, and the second one is related to the alignment of standards ontologies with the BIGG Ontology.

The conversion step is the first stage of the Python module, it corresponds to the use of the java library to convert an input JSON file into an RDF file thanks to the mapping rules defined in the RML file. The module can output two serialization formats of RDF, the TTL (Turtle) or the JSON-LD format.

The second stage of the Python module, corresponds to the use of SPARQL queries in order to add, translate or complete the RML stage. The second stage can also be used to align data based on standardized Ontology into the BIGG compliant RDF. For instance, it allows to align data based on standards ontologies like ifcOWL, SOSA or SAREF, with the BIGG Ontology.

The execution and test of the Python module can be done in a Jupiter Notebook, with the command line of Figure 8.

Experimentation with data provided by pilot sites

The harmonizer process can be illustrated with a simple example of a building containing storeys, spaces, and some devices. Figure 7 shows the corresponding JSON input file on the top left, and the YARRRML mapping file on the right. Once the RML mapping file is generated, we can use it to convert the input file into an RDF file aligning with the BIGG Ontology (bottom left).

The Python module allows the use of the RML mapping rules to align the input data with the BIGG Ontology into an RDF document. Automatically, it generates instances of BIGG objects like bigg:Building, bigg:BuildingSpace and bigg:Device with the related relations (Figure 9).

The same approach has been successfully applied with input data from other sources: group, systems and sensors structure exported from exploitation control software, or time series coming directly from sensors. Each specific data is mapped to instantiate a specific part of the BIGG ontology.

In a second phase, the harmonization module is also used to create rules to include import data in standard format (e.g. ifcOWL) into an instance of BIGG model. Indeed, the harmonizer second stage allows to execute SPARQL queries to specify correspondences between ontologies (Figure 10).

IFCOWL	BIGG
:IfcBuilding	bigg:Building
:IfcSpace	bigg:BuildingSpace
:IfcZone	bigg:Zone
:IfcDistributionElement	bigg:Device

Figure 9: Example of Alignment between ifcOWL Ontology and BIGG Ontology

Discussion & perspectives

The BIGG harmonizer implements and orchestrates transformations on heterogeneous data. This article demonstrates how the harmonizer can convert any data covering building and sensors description as well as energy related time series. It is fully compliant with W3C recommendations by composing an RML transformation with SPARQL transformations. This solution is extensible as any data providers can add his own mapping files to process his own data stream to the unified BIGG data toolbox. The main limitation of the current implementation is the use of files as data buffers between providers and the digital twin.

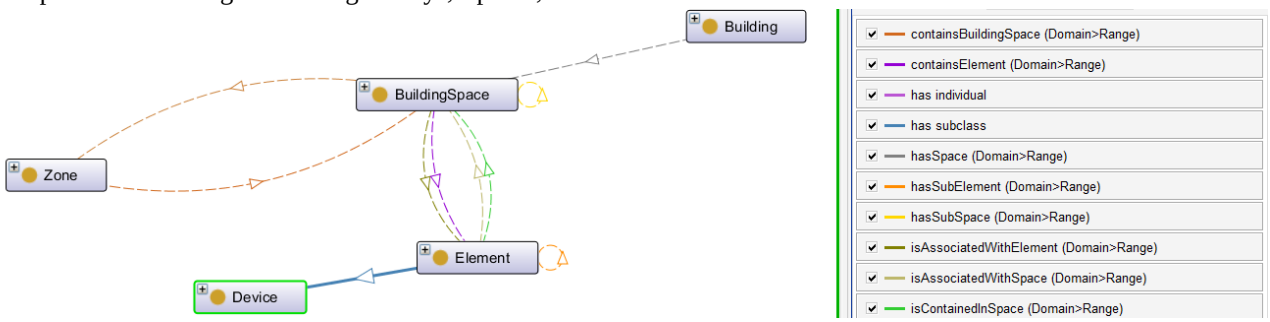

Figure 10: Visualization of objects from the BIGG Ontology to be instantiated with the Python module.

At this stage of the project, a dozen of RML mapping files have been developed for real input data (several initial tests were done with manually developed input files), covering 5 different Use Cases. Two of the involved partners were able to generate these mapping file quite easily using Matey tool, and three other partners have been trained to learn YARRRML language and are now able generate their own mapping files.

Matey tool and YARRRML language allowed us to develop most of the mapping we need. We encountered mapping difficulties on only one input file, which have a more complex structure: the depth of element decomposition depends on the type of element. This requires implementing some “Conditions” in YARRRML and managing “Iterators” in a different way. These aspects have been investigated during the last months of the project and seems very promising.

From a data integration point of view, the main objective of the BIGG project was to design, develop, and test a specific architecture allowing to address as many use cases as possible through a common data analysis pipeline. By designing a dedicated BIGG ontology, and developing a generic data harmonization tool, based on RML mapping language, to feed it from heterogeneous data sets, we achieved our objective, while using standard, open, and extensible technical solutions.

The RAF describes BIGG components and state-of-the-art techniques to coordinate BIGG components, may the actual architecture deployment on a local server or on a cloud infrastructure. The next step is the integration of the harmonizer into the whole stream-oriented architecture. Each BIGG component implements its business logic using specific technologies and specific supporting open sources libraries or frameworks (e.g., Kafka (Noac'H et al., 2017) (Hiraman et al., 2018)). Then business logic code is containerized using Docker (Wan et al., 2018). Components get their features accessible via APIs (Application Programming Interface), with event message compatible interface. For the latter point, the Kafka event streaming message system and RMLStreamer (Min et al., 2022) has been chosen to allow the platform to process continuous data streams.

References

N. Boutouni, F. Lampathaki, S. Kousouris, A. Tsitsanis, G. Vafeiadis (2021) The BIMERR Interoperability Framework: Towards BIM Enabled Interoperability in the Construction Sector.

A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In Proceedings of the Workshop on Linked Data on the Web, Seoul, Korea, 2014.

K. Katsigarakis, G. N. Lilis, D. Rovas, S. González-Gerpe, S. Bernardos, A. Cimmino, M. Poveda-Villalón, R. García-Castro (2022) Digital Twin Platform Generating Knowledge Graphs for Construction Projects.

N. Pastorelly. BIGG, D2.2 - Initial technical specifications and preliminary design of BIGG Architecture building blocks. H2020-LC-SC3-EE-2020-1/LC-SC3-B4E-6-2020.

Radulovic, F., Poveda-Villalón, M., Vila-Suero, D., Rodríguez-Doncel, V., García- Castro, R. and Gómez-Pérez, A., 2015. Guidelines for Linked Data generation and publication: An example in building energy consumption. Automation in Construction, 57, pp.178-187.

F. Michel. Integrating heterogeneous data sources in the Web of data. Other [cs.OH]. Université Côte d’Azur, 2017. English. ffNNT : 2017AZUR4002ff. fftel-01508602v3f

Berners-Lee, T. 2009. Linked data: Design issues. Accessed March 02, 2023. https://5stardata.info/en/

E. Iglesias, S. Jozashoori, D. Chaves-Fraga, D. Collarana, and M.E. Vidal, 2020. SDM-RDFizer: An RML Interpreter for the Efficient Creation of RDF Knowledge Graphs

M. Lefrançois, A. Zimmermann, N. Bakerally, 2017. SPARQL-Generate. A SPARQL extension for generating RDF from heterogeneous formats.

S. Das, S. Sundara, R. Cyganiak, R2RML: RDB to RDF Mapping Language, 2012. https://www.w3.org/TR/r2rml/

U. Simsek, E. Karle, and D. Fensel. RocketRML - A NodeJS implementation of a use-case specific RML mapper. In Proceeding of the 1st International Workshop on Knowledge Graph Building, 2019.

Heyvaert, P., De Meester, B., Dimou, A., Verborgh, R. (2018). Declarative Rules for Linked Data Generation at Your Fingertips!. In: , et al. The Semantic Web: ESWC 2018 Satellite Events. ESWC 2018. Lecture Notes in Computer Science(), vol 11155. Springer, Cham. https://doi.org/10.1007/978-3-319-98192-5_40. https://rml.io/yarrrml/matey/

L. Noac'H, A. Costan, and L. Bougé, "A performance evaluation of Apache Kafka in support of big data streaming applications", in Big Data (Big Data), 2017 IEEE International Conference on, 2017, pp. 4803-4806.

Hiraman B., Chapte M. and Abhijeet C. (2018). A Study of Apache Kafka in Big Data Stream Processing. 1-3. 10.1109/ICICET.2018.8533771.

Wan X., Guan X., Wang T., Bai G. and Choi B. 2018. Application deployment using Microservice and Docker containers: Framework and optimization. Journal of Network and Computer Applications. 119. 10.1016/j.jnca.2018.07.003.

S. Min Oo, G. Haesendonck, B. De Meester, A. Dimou. RMLStreamer - an RDF stream generator from streaming heterogeneous data. The Semantic Web – ISWC 2022. Springer International Publishing, (2022)