Semantic-IoT Framework: Knowledge Graph Generation from IoT Platforms

Junsong Du[1,*], Yunheng Tian[1], Max Berktold[1], Rita Streblow[1] and Dirk Müller[1]

[1]*RWTH Aachen University, E.ON Energy Research Center, Institute for Energy Efficient Buildings and Indoor Climate, Germany*

Abstract

Semantic technologies, and knowledge graphs (KGs) in particular, show promising potential for enhancing interoperability among diverse IoT systems, especially in the building sector. However, the extensive manual effort required for semantic modeling limits the widespread adoption of these technologies in engineering practice. This paper introduces Semantic-IoT, a framework that generates KGs from IoT platforms using the RDF Mapping Language (RML). By employing a semi-automated approach for declaring RML rules alongside fully automated KG generation, the framework reduces the manual effort associated with semantic modeling. The generated KGs comprehensively represent both system information and data interactions, thereby facilitating the development and deployment of cross-platform applications. A building automation use case is presented to demonstrate the feasibility and effectiveness of the proposed approach.

Keywords

IoT Platform, Interoperability, Building Automation, RML

1. Introduction

The Internet of Things (IoT) and IoT platforms enable efficient data management and flexible service deployment in the building sector. However, the wide adoption of such technologies is hindered by the lack of interoperability, especially semantic interoperability, across different IoT platforms [1]. Specifically, the use of proprietary data structures, vocabularies, and different application programming interfaces (APIs) requires individual development efforts to bridge the gaps between different platforms [2].

Semantic Web technology (SWT) shows promising potential to address the interoperability issue [3]. However, the creation of KGs often relies on expert knowledge and the use of tools like Protégé[1], which requires significant manual effort during the initial setup [4]. Therefore, further approaches for generating KGs need to be explored.

In this context, this paper proposes to use IoT platforms as a source of information to generate KGs that describe the underlying systems and the data interactions via platform APIs. This approach is based on the fact that many modern IoT platforms, such as FIWARE[2], openHAB[3], and OpenRemote[4], can incorporate semantic information. Consequently, we introduce the Semantic-IoT framework, which integrates semantic technologies with conventional IoT architectures. This framework facilitates the generation of KGs for IoT systems and enables the use of SWT, such as reasoning and SPARQL, to improve information exchange between IoT platforms and application providers. In this way, cross-platform applications can be developed and deployed more efficiently, thereby enhancing the interoperability.

CEUR-WS.org/Vol-4064/PD-paper12.pdf

SEMANTiCS'25: International Conference on Semantic Systems, September 3–5, 2025, Vienna, Austria

*Corresponding author.

junsong.du@eonerc.rwth-aachen.de (J. Du)

0000-0003-2247-2423 (J. Du)

© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

[1]https://protege.stanford.edu/

[2]https://fiware.org/

[3]https://www.openhab.org/

[4]https://openremote.io/

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

2. Proposed Framework

Figure 1 illustrates the Semantic-IoT framework, which employs the RDF Mapping Language (RML) [5] and comprises two primary modules: the RML Generation and the Knowledge Graph Construction Pipeline (KGCP). The implementation and an example use case are available at https://github.com/N5GEH/semantic-iot.

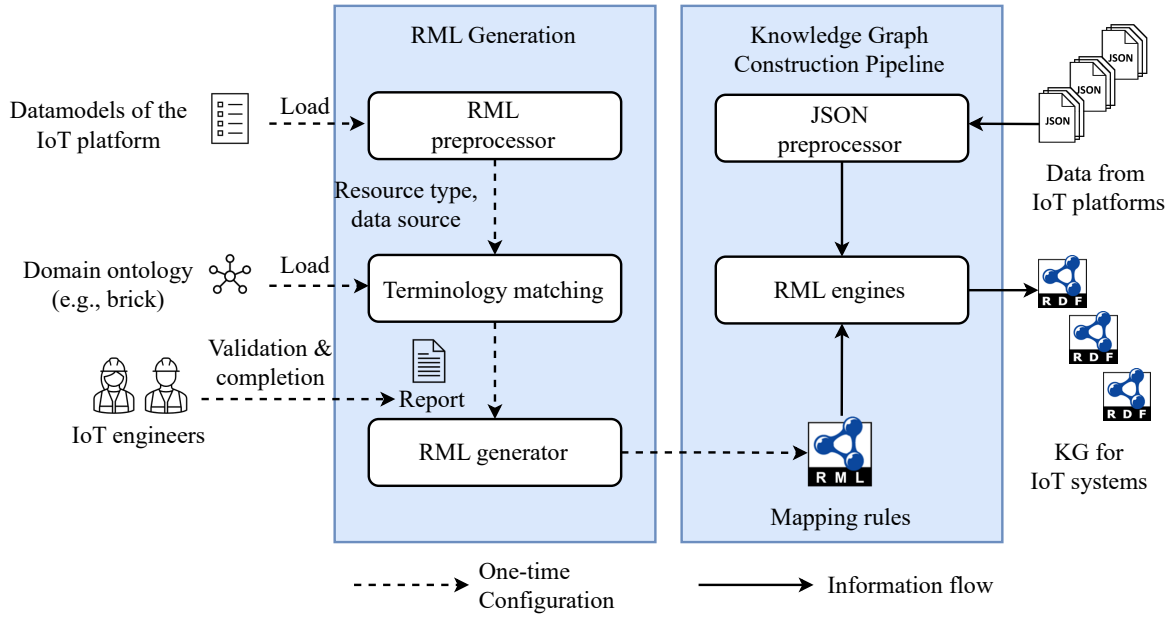


Figure 1: Functional overview of the Semantic-IoT framework

The RML Generation module is designed to semi-automatically create RML mapping rules for various IoT platforms. Initially, this module loads a dataset that represents the data models of an IoT platform. Typically, such a dataset comprises a list of virtual entities that encapsulate sensor data, actuation functions, and semantic information about the underlying system. From this dataset, information required by RML is extracted by dedicated submodules, as shown in Table 1.

Table 1
Overview of information retrieval in the RML Generation module. The *Mode* indicates whether the information is generated automatically or requires manual input or validation.

RML Section	Information Required	Submodule	Mode
logicalSource	JSONPath	RML preprocessor	Automatic
subjectMap	Resource type	RML preprocessor	Automatic
	IRI template	RML preprocessor	Automatic
	Subject terminology	Terminology matching	Semi-automatic
predicateObjectMap	Predicate terminology	Terminology matching	Semi-automatic
	Interrelationship	RML preprocessor	Automatic
	Data access	RML generator	Manual input

The RML preprocessor first identifies unique resource types from the dataset based on the type field of each entity. For each resource type, the corresponding JSONPath to locate entities of that type (e.g., *$[?(@.type=='TemperatureSensor')]*) and the IRI template for the RDF subject are generated. The extraction of interrelational information requires more complex processing as illustrated in Algorithm 1. Simply put, the RML preprocessor traverses the JSON object of each entity and identifies substructures that contain references to other entities. In this way, a structural skeleton of the specific data model is created.

Algorithm 1 Find interrelationship for a given entity

```
Input: An JSON object entity, a list of JSON objects allEntities
Output: A list foundRelationships for the given entity

foundRelationships ← an empty list
for otherEntity in allEntities do
    if entity.id ≠ otherEntity.id then
        for (path, value) in TRAVERSEJSON(entity) do
            for value = otherEntity.id do
                record ← new record with fields {path, relatedType}
                record.path ← path
                record.relatedType ← otherEntity.type
                Add record to foundRelationships
return foundRelationships
```

Subsequently, the subject and predicate terminologies are matched against the domain ontologies. This submodule loads the serialized domain ontologies and then uses the Levenshtein algorithm to compute string similarities between the terms of the data model and those in ontologies. Based on the highest similarity scores, matching suggestions are generated. These suggestions, along with the previously extracted information, are populated into a report for engineers to validate. Currently, manual intervention is required to validate the suggested terminology matches and to specify URL patterns for accessing API endpoints of the IoT platform, specifically when a resource type provides sensor data or supports actuation functions. Once the report is finalized, the RML mapping rule can be generated automatically, thereby eliminating the need for manual handling of RML syntax. In comparison to YARRRML [6], which enables a human-friendly declaration of RML mapping rules, the proposed framework is specifically tailored for IoT platforms and offers a higher degree of automation for this specific use case.

Once the RML mapping rule is generated, a platform-specific KGCP is established. This KGCP can be reused to automatically generate KGs from various datasets and across different platform instances. While the JSON preprocessor ensures the general applicability of the KGCP by normalizing the data from IoT platforms, the RML engine, MorphKGC [7], is employed to generate KGs. It is important not to confuse the proposed KGCP with a runtime virtualization layer. While the virtualization approach offers direct access to database [8], it often demands custom development, especially when integrating NoSQL databases like MongoDB. The KGCP, in contrast, is created through a lightweight configuration process based on the established RML mechanism. By relying solely on the exposed structural data of APIs, this approach is inherently more flexible for modern IoT platforms.

3. Use Case

The proposed framework is planned to be applied in the building sector as illustrated in Figure 2. Currently, the deployment of promising smart building applications—such as for automation, fault detection, and energy monitoring—is often a laborious task due to the heterogeneity of underlying building systems and their IoT platforms. Semantic-IoT addresses this challenge by facilitating the generation of KGs, which provide a unified, semantic representation of the available information. By leveraging SWT, such as reasoning and SPARQL, the deployment process can ultimately be automated.

3.1. Framework Demonstration

The proposed framework is demonstrated through a building automation use case[5]. In this use case, an IoT system has been developed for hotel buildings based on the FIWARE platform. Platform-specific data

[5] https://github.com/N5GEH/semantic-iot/tree/main/examples/fiware

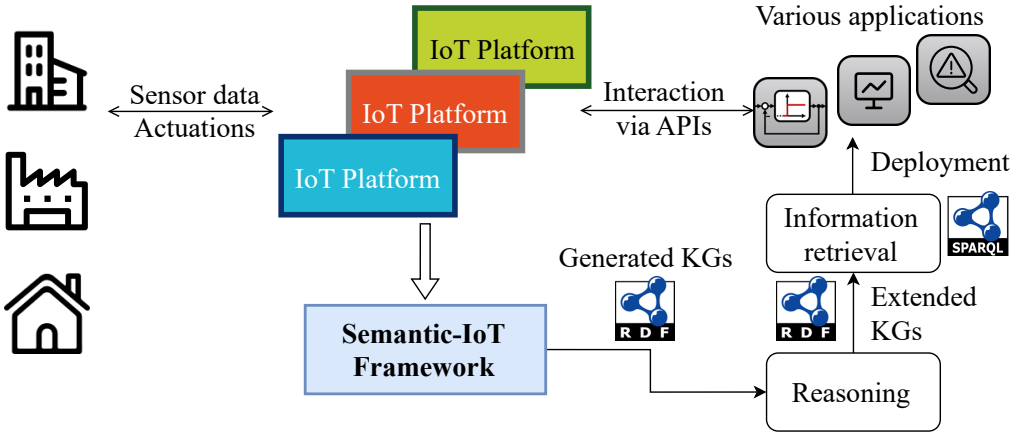


Figure 2: Use case of the Semantic-IoT framework in building sector.

models[6] have been designed in accordance with the NGSIv2 specification of FIWARE. To fully unlock the flexibility to connect different building automation services, the Semantic-IoT framework is used.

With the RML Generation module, twelve resource types are identified from the data models, including locations, sensors, and actuators. Brick[7], a widely-used ontology for building energy systems, is used to provide the semantic foundation for the terminology matching. Although Brick can theoretically be applied to all identified resource types, the current terminology matching achieves less than 60% accuracy. For example, while the resource type *PresenceSensor* should ideally match the class *brick:Occupancy_Count_Sensor*, the similarity score between them is only 0.5. Consequently, identifying suitable terminologies remains largely a manual work. In total, the complete RML mapping rules consists of 344 lines, with manual intervention limited to 18 lines for validation and 7 lines for manual input. Thus, the proposed framework significantly simplifies the declaration of RML mapping rules for IoT platforms.

With the generated RML mapping rules, a KGCP for the FIWARE-based platform is established, enabling the construction of KGs for any hotel that utilizes the same platform. To test its applicability, we then provisioned a range of virtual hotels, from a small 2-room layout to a large 1000-room one. For each hotel, we fetched the available data from the FIWARE NGSIv2 API to create the test datasets[8]. As a result, the KGCP generates KGs that represent the hotel buildings—including their rooms, sensors, and actuators—and integrate URLs for data interaction via the platform API.

3.2. Automatic Service Deployment

Subsequently, we demonstrate the possibility to automate the deployment of automation services for ventilation control[9]. The tasks are mainly twofold: first, to decide on control strategies based on the available sensors and actuators; and second, to establish reliable data interactions via the platform API. To optimize information retrieval, we employ OWL-RL[9] to infer the generated KGs. Numerous class subsumption is added; for example, the class *brick:Ventilation_Air_System* is inferred to be a subclass of *brick:Air_System* and *brick:HVAC_System*. Thus, generalized SPARQL queries can be applied to retrieve any possible actuators in the hotel air systems. The availability of sensors can also be queried in a similar way. As a result, configurations for the ventilation controller can be automatically generated.

[6] https://github.com/N5GEH/n5geh.data_models/tree/main/example_building_automation
[7] https://brickschema.org/
[8] https://github.com/N5GEH/semantic-iot/tree/main/examples/fiware/hotel_dataset
[9] https://github.com/N5GEH/semantic-iot/tree/main/examples/fiware/application_deployment

4. Conclusion and Future Work

In this paper, we introduce a framework that automates the generation of knowledge graphs (KGs) from IoT platforms. By integrating system information and data interactions into the generated KGs, our approach enhances interoperability across diverse IoT systems, thus simplifying the development and deployment of cross-platform applications.

In future work, we aim to leverage the HTTP Vocabulary[10] to enrich the semantic representation of platform APIs. Moreover, the current string similarity based terminology matching exhibits limited accuracy. Consequently, we plan to investigate alternative approaches, such as word embedding models. Ultimately, we intend to conduct comparative case studies that deploy advanced building automation programs across different IoT systems with various representative IoT platforms.

Acknowledgments

We gratefully acknowledge the financial support provided by the Federal Ministry for Economic Affairs and Climate Action (BMWK), promotional reference 03EN1030B.

Declaration on Generative AI

During the preparation of this work, the authors used Gemini in order to: Grammar and spelling check, Paraphrase and reword. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] M. Noura, M. Atiquzzaman, M. Gaedke, Interoperability in Internet of Things: Taxonomies and Open Challenges, Mobile Netw Appl 24 (2019) 796–809. URL: https://doi.org/10.1007/s11036-018-1089-9. doi:10.1007/s11036-018-1089-9.

[2] A. Hazra, M. Adhikari, T. Amgoth, S. N. Srirama, A Comprehensive Survey on Interoperability for IIoT: Taxonomy, Standards, and Future Directions, ACM Comput. Surv. 55 (2021) 9:1–9:35. URL: https://dl.acm.org/doi/10.1145/3485130. doi:10.1145/3485130.

[3] M. Ganzha, M. Paprzycki, W. Pawłowski, P. Szmeja, K. Wasielewska, Semantic interoperability in the Internet of Things: An overview from the INTER-IoT perspective, Journal of Network and Computer Applications 81 (2017) 111–124. URL: https://www.sciencedirect.com/science/article/pii/S1084804516301618. doi:10.1016/j.jnca.2016.08.007.

[4] H. Dibowski, J. Ploennigs, K. Kabitzsch, Automated Design of Building Automation Systems, IEEE Transactions on Industrial Electronics 57 (2010) 3606–3613. URL: https://doi.org/10.1109/TIE.2009.2032209. doi:10.1109/TIE.2009.2032209.

[5] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, R. Van de Walle, RML: a generic language for integrated RDF mappings of heterogeneous data, in: Proceedings of the 7th Workshop on Linked Data on the Web, volume 1184 of *CEUR Workshop Proceedings*, 2014. URL: http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf.

[6] D. Van Assche, T. Delva, P. Heyvaert, B. De Meester, A. Dimou, Towards a more human-friendly knowledge graph generation & publication, in: International Semantic Web Conference (ISWC) 2021: Posters, Demos, and Industry Tracks, 2021. URL: https://rml.io/yarrrml/assets/pdf/iswc2021.pdf.

[7] 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. URL: https://doi.org/10.3233/SW-223135. doi:10.3233/SW-223135.

[10] https://www.w3.org/TR/HTTP-in-RDF10/

[8] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Ontology-Based Data Access and Integration, in: Encyclopedia of Database Systems, Springer, New York, NY, 2018, pp. 2590–2596. doi:10.1007/978-1-4614-8265-9_80667.
[9] I. Herman, OWL-RL: OWL-RL: A simple OWL2 RL reasoner on top of rdflib, Zenodo, 2014. URL: https://doi.org/10.5281/zenodo.14543. doi:10.5281/zenodo.14543.