From Opaque Streams to Explainable Systems: Semantic MQTT Integration at the Edge

Niklas Doerner * and Maria Maleshkova

Posted Date: 22 May 2026

doi: 10.20944/preprints202605.1509.v1

Keywords: MQTT; ontology engineering; data streams, sensor ontology; semantic sensor network; message protocol; smart sensors; node-RED; RML; semantic integration framework

 doi:10.20944/preprints202605.1509.v1

Article

From Opaque Streams to Explainable Systems: Semantic MQTT Integration at the Edge

Niklas Doerner * and Maria Maleshkova

Helmut-Schmidt-University, Holstenhofweg 85, 22043 Hamburg, Germany
* Correspondence: doernern@hsu-hh.de

Abstract

Industrial systems increasingly rely on MQTT-based message streaming to enable automated, data-driven production processes at the network edge. While semantic models such as the SSN/SOSA ontology enable machine-interpretable descriptions of observations and actuations, an explicit model of message transport is rarely considered. Consequently, MQTT-based communication remains opaque, particularly regarding information processing, hindering the semantic analysis of application-specific topic structures and the behavior of transport protocols. To close this gap, this work introduces the revised MQTT4SSN ontology as a key contribution, extending existing semantic models with protocol-aware representations of MQTT entities, control packets, and transport-level interactions. MQTT4SSN enables end-to-end semantic traceability, from sensor observations and actuator controls to the underlying message transmission within distributed systems. Building on this contribution, the MQTT2RDF integration framework incorporates MQTT4SSN as its core to capture live MQTT traffic and represent both payload meaning and transport-level provenance within an RDF knowledge graph. This work presents a novel approach for representing edge computing and information processing over MQTT, addressing two key challenges. First, a semantic topic-naming approach automatically derives MQTT topic hierarchies and payload content structures from observation and actuation semantics. This approach facilitates the setup of edge computing systems and enables context-aware subscription management and structured data formatting, thereby improving interoperability between heterogeneous deployments. Second, transport-level provenance analytics support automated detection, classification, and root cause analysis of malformed MQTT packets and protocol-level errors. The approach provides explainable, traceable information processing through transport provenance, which is essential for safety-critical industrial environments. The contributions are validated through an industrial use case from a production environment, demonstrating its applicability for system monitoring, troubleshooting, and semantic analytics of MQTT-based infrastructures.

Keywords: MQTT; ontology engineering; data streams, sensor ontology; semantic sensor network; message protocol; smart sensors; node-RED; RML; semantic integration framework

1. Introduction

Industrial technical systems increasingly depend on continuous data streams to support automated and data-driven decision-making in real time. In modern industrial Internet of Things (IoT) environments, sensors and actuators continuously generate high-frequency data that must be processed, transmitted, and interpreted across distributed systems. At the same time, there is a growing need for solutions that can integrate and map information from multiple heterogeneous sources, in various formats, into a unified semantic representation. The importance of this requirement is particularly evident in industrial IoT settings, where continuously generated sensor data streams must be semantically interpreted and sequenced in real time rather than being mapped once from static datasets.

 doi:10.20944/preprints202605.1509.v1

A common architectural pattern to address these requirements is edge computing, where data is processed close to its source and selectively forwarded to higher-level systems for further complex computations [1,2]. In such architectures, sensors and actuators are connected to edge devices, which exchange data with backend services through scalable machine-to-machine communication protocols. One of these, the Message Queuing Telemetry Transport (MQTT) protocol [3] has become an established standard in IoT due to its lightweight Publish/Subscribe architecture, low bandwidth requirements, and high scalability [4,5]. With MQTT, clients publish messages to a central broker using topics that typically consist of hierarchical paths. The broker distributes them to clients who subscribe to the corresponding topics. This broker-based architecture decouples publishers and subscribers of messages, enabling efficient, scalable data exchange in highly dynamic, resource-constrained environments.

However, despite its widespread adoption, MQTT introduces significant interoperability and semantic challenges. Topic structures and payload contents often encode meaning only implicitly through vendor-specific naming conventions and proprietary data formats. As a result, the interpretation of transmitted messages depends heavily on application-specific knowledge, limiting interoperability, traceability, and automated reasoning across distributed systems. Regarding message transport metadata, protocol-level information such as message provenance, session context, and communication events remains opaque in conventional MQTT deployments. Aside from that, the Web of Things (WoT) community recommends the W3C SSN/SOSA ontology as a standard model for describing sensors, observations, and actuations in a machine-interpretable manner [6–8].

To illustrate these challenges, consider a modern industrial production environment in which multiple machines continuously exchange data with local and remote systems. Sensor observations and control commands must be transmitted reliably across distributed infrastructures to support monitoring, automation, and higher-level analytics. The environment shown in Figure 1 consists of several machines (clients) equipped with sensors (e.g., temperature, vibration, or pressure sensors) and actuators (e.g., robot arms). Each machine communicates with an MQTT broker at an edge device by publishing and subscribing to MQTT topics. The MQTT broker then interacts with a remote server, also by publishing and subscribing to MQTT topics for semantic integration and analysis. Summarized, in the factory environment, monitoring applications, control systems, and data-processing services from multiple clients subscribe to relevant topics to process incoming streams in real time. At the same time, the remote server performs higher-level semantic integration and analytics.

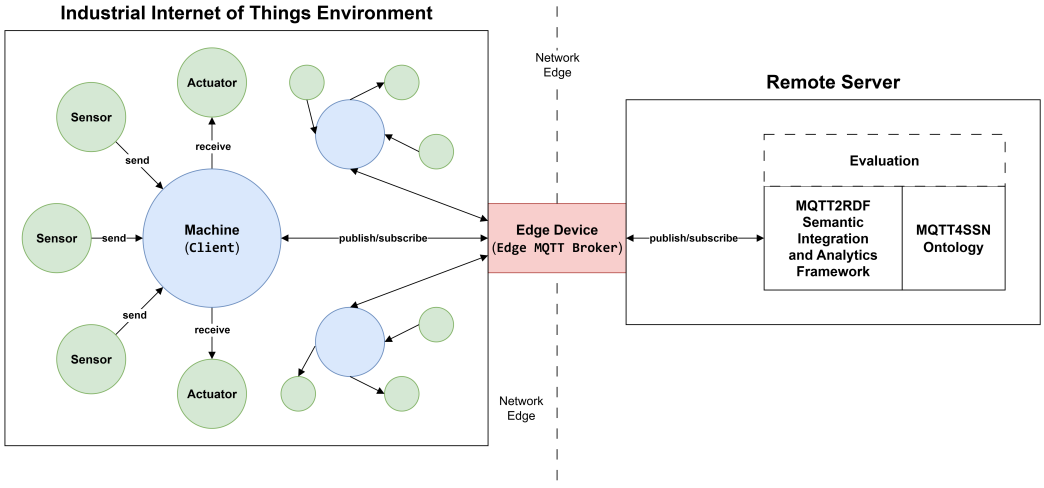

Figure 1. Industrial IoT architecture showing sensor–actuator systems connected via MQTT broker at an edge device to a remote server for semantic integration and analysis.

While this setup enables scalable and efficient communication among many clients, the exchanged messages remain semantically ambiguous. For instance, the meaning of a topic string or the structure of a payload is often only known to specific system components due to vendor specifications,

 doi:10.20944/preprints202605.1509.v1

hindering interoperability and automated integration. While SSN/SOSA provides rich semantics for observational processes and physical system deployment, it does not model machine-to-machine (M2M) communication protocols such as MQTT. In addition, heterogeneous payload structures and character encodings are outside its scope, limiting the interoperable integration of transmitted data streams [9]. Hence, the existing semantic models fail to capture the entire process of data transmission, from generation to interpretation, thus leading to incomplete semantic representation and limited analyses.

To address these limitations, this paper introduces MQTT2RDF, a reusable semantic integration framework that bridges the gap between sensor semantics and M2M communication semantics across distributed IoT systems. The framework enables transforming live MQTT message streams into RDF-based knowledge graphs, thereby supporting real-time semantic analytics. It provides artifacts for ontology population that capture MQTT traffic directly from the broker, instantiate RDF triples, and stream them into a triplestore. At its core, MQTT2RDF incorporates MQTT4SSN [10], an ontology that explicitly represents MQTT network entities, control packets, and session contexts. The ontology enables a comprehensive semantic representation of the MQTT protocol and aligns it with the observation, actuation, and sampling semantics of the W3C SSN/SOSA ontology [7,8]. By making message meanings explicit and machine-interpretable, the approach facilitates interoperability, traceability, and advanced semantic analytics in IoT environments. The MQTT4SSN ontology and the MQTT2RDF framework are openly available, adaptable, and extensible for use across various domains. This work makes the following contributions:

- *MQTT4SSN Ontology*: Further development of the previous version, providing a comprehensive representation that covers all MQTT 5.0 control packets, including their complete contents and structure. Enrichment of the network infrastructure with session context.
- *MQTT2RDF Semantic Integration and Analytics Framework*: A modular framework that integrates heterogeneous MQTT message streams into RDF-based knowledge graphs in real time. The framework supports semantic querying, provenance-aware analytics, anomaly detection, and explainable analysis across distributed IoT environments.
- *Ontology and Framework Evaluation*: A thorough evaluation of the ontology based on its coverage of the MQTT 5.0 specification [3] and a use case in the industrial context, complemented by runtime analyses of the framework, evaluating semantic processing overhead and practical applicability.

The remainder of this paper is structured as follows: Section 2 introduces an industrial usage scenario and motivates the development of a semantic integration and analytics framework. Section 3 provides an overview of related ontologies 3.1 that represent the MQTT protocol, its associated concepts, and its automatic population 3.2. Section 4 describes the overall research methodology, including requirements definition 4.1, the ontology engineering process 4.2, integration and analytics framework design and implementation 4.3, and considerations for publication and reusability 4.4. Section 5 addresses the ontology development process of MQTT4SSN, covering its core concept 5.1, ontology description 5.2. Section 6 introduces MQTT2RDF, a framework for the semantic integration of continuous MQTT data streams into a triplestore. Section 7 presents the evaluation of the proposed approach, including ontology verification 7.2 and validation 7.3 based on competency questions, as well as an industrial use-case setup 7.1. Finally, Section 8 concludes the paper and gives a direction for future work.

2. Motivating Industrial Usage Scenario

Consider a manufacturing company that produces customized packaging units. Packages are transported on a conveyor belt and handled by a robot arm. Each package is identified by an RFID tag, allowing the automation production system to associate sensor measurements, processing steps, and actuation commands with an individual product instance. During operation, a vision sensor mounted on the robot arm measures the three-dimensional grip position of each piece and publishes the measurements as MQTT messages. An edge device receives these messages, calculates correction

 doi:10.20944/preprints202605.1509.v1

offsets, and sends the resulting actuation commands back to the robot arm via MQTT, allowing the robot to adjust its grip before picking the packaging piece. Although this distributed system architecture is efficient and scalable, it creates several practical problems for integration, monitoring, and quality assurance.

Message Content Analytics: First, the meaning of the transmitted payloads is not explicit. Transmitted data can only be correctly interpreted if the receiving system already knows the payload structure, the unit, the associated sensor or actuator, and the production context. If a new sensor vendor introduces, e.g., a different payload format, the integration logic must often be adapted manually.

Semantic Topic Analytics Second, MQTT topics are typically defined according to local or vendor-specific naming conventions. For example, a topic may indicate a production line, a robot, a sensor, or a measured property, but MQTT itself does not define the semantics of the topic levels. As a result, systems cannot reliably discover which topics provide observations of a certain property or which topics carry actuation commands for a specific robot arm without additional application-specific knowledge.

Transport and Provenance Analytics Third, quality problems in the production process are difficult to trace across distributed systems. If a robot grips a package incorrectly, several causes are possible: the vision sensor may be malfunctioning, the edge device may have calculated incorrect values, the actuation command may have been delayed, or a message may have been lost or malformed during transmission. Conventional MQTT-based infrastructures transport messages but do not provide an integrated semantic representation that connects product identifiers (e.g., RFID), sensor observations, actuation commands, topics, clients, brokers, and available transport metadata. Semantic end-to-end traceability can support debugging and accountability analysis, for example, through Message Transport Provenance [11,12].

MQTT2RDF addresses these issues by transforming MQTT message streams into an RDF-based knowledge graph (KG). The MQTT4SSN ontology provides the semantic model for representing MQTT clients, brokers, topics, control packets, payloads, application messages, and their links to SSN/SOSA concepts such as sensors, observations, actuators, and actuations. In the packaging scenario, this enables the system to represent a measured grip position as a semantic observation related to a specific RFID-tagged packaging piece, and a correction value as an actuation-related command addressed to a robot arm.

The scenario, therefore, motivates three central analytics requirements addressed by MQTT2RDF: First, heterogeneous message contents must be transformed into semantically interpretable representations. Second, MQTT topics must be linked to the SOSA semantics they refer to. Third, application-level observations and actuation commands must be linked to their respective MQTT communication contexts to support provenance-aware traceability and diagnostics.

3. Related Work

The related work is structured according to the two main levels addressed in this paper: the semantic data model level and the semantic integration framework level. At the data model level, a structured literature review was conducted on related work on the semantic representation of the MQTT protocol using ontologies, as discussed in subsection 3.1. At the framework level, we examine approaches to transformation. Furthermore, we searched for related solutions to integrate MQTT messages with their metadata into RDF-based knowledge graphs and for enabling semantic querying and analytics over these representations, as discussed in subsection 3.2.

3.1. Related Data Models

The W3C Semantic Sensor Network (SSN) ontology and its SOSA core are widely adopted for describing sensors, observations, actuations, and sampling in semantic IoT systems in a machine-interpretable way [6,13]. SOSA results from the design and generalization of the original Stimulus Sensor Observation (SSO) core, offering a unified pattern for observation, sampling, and actuation [6]. Architecturally, SOSA is a lightweight core vocabulary, while SSN imports SOSA and adds richer

 doi:10.20944/preprints202605.1509.v1

axioms and modules to enable more expressive reasoning over systems, deployments, and capabilities [13,14]. The 2023 Semantic Sensor Network Ontology Working Draft, designated as 2023 Edition, added a number of new terms to the SSN Ontology [15]. Within this scope, SSN/SOSA deliberately stay at the conceptual layer: they do not define how observation streams are transported, how protocols such as MQTT relate to observations, or how messages are framed. Nor do they model heterogeneous payload formats or character encoding. Consequently, a semantic gap remains between SSN/SOSA-based descriptions and the message-level integration required for interoperable IoT data exchange.

The QUDT (Quantities, Units, Dimensions, and Data Types) vocabulary provides a machine-readable and unambiguous representation of quantity kinds, units of measure, and dimensional relations [16,17]. QUDT is commonly reused alongside semantic sensing ontologies, such as SSN/SOSA, to annotate observation results with explicit units and scales, thereby improving the interoperability of sensor data and associated technical metadata [6,7].

To the best of our knowledge, the only semantic representations of the MQTT messaging protocol is the MQTT to RDF draft ontology defined in the W3C WoT Binding Templates, hereafter referred to as MQV [18], and our previously published MQTT4SSN ontology [19]. MQV is denoted as work in progress and lacks an incomplete representation of MQTT.

MQTT4SSN models the semantics of the MQTT protocol and aligns them with the well-established SSN/SOSA ontology, enabling end-to-end traceability between sensing systems and message transfer [19]. While SSN/SOSA provides a solid foundation for modeling sensing systems [6,7], MQTT4SSN complements this by explicitly modeling the structure and metadata of the transmitted data [19]. MQTT4SSN models MQTT-specific entities, including brokers and clients, as well as control packet structures, their payloads, and associated topics [19]. Following best practices in ontology engineering, MQV was considered a potential ODP [20] and used as the starting point for the ontology design [18,19]. MQTT4SSN adapted most MQV concepts by aligning equivalent concepts, enabling direct inference across both ontologies and enabling future interoperability. Additional elements facilitate the semantic relationship with sensing systems represented using SSN/SOSA [13,19]. The ontology enables the representation of heterogeneous payload formats and character set encodings of the application message, carried within the publish packet's payload, to handle structured data such as CSV or JSON [19]. The QUDT Vocabulary was integrated into MQTT4SSN to represent measurement results in a precise and interoperable manner, ensuring that units are machine-readable, semantically precise, and avoiding ambiguity in data interpretation [16,17,19]. Previous work on MQTT4SSN has focused on MQTT version 3.1.1. Aside from that, the model is limited to basic message properties of the publish, subscribe, and unsubscribe control packets [19]. The integration bridges the gap between sensor semantics and M2M communication semantics across distributed IoT systems with seamless alignment between observation, actuation, and sampling semantics and the message protocol. For future work, extending the ontology with the current MQTT 5.0 specification [3], which introduces enhanced message properties, coverage of all control packets, session control, and user-defined metadata features, was planned.

3.2. Semantic Integration and Knowledge Graph Population

While the previous subsection focused on semantic data models, this subsection outlines related work on semantic integration and knowledge graph population for framework-level approaches. In the context of IoT systems, such frameworks must transform device data, messages, and their metadata into RDF for querying and analysis. Related work addresses RDF mapping languages, approaches to KG materialization and virtualization, a solution for KG construction, and an overview of suitable brokers.

R2RML is the W3C standard mapping language for defining customized mappings from relational databases (RDB) to RDF datasets [21]. RML extends R2RML to support mappings for other structured data formats, including JSON, CSV, and XML [21,22]. Further, xR2RML extends both R2RML and RML to cover mappings from non-relational (NoSQL) databases [23].

 doi:10.20944/preprints202605.1509.v1

These mapping languages serve as a foundation for generating RDF from heterogeneous data sources. However, they primarily focus on mapping and do not define a comprehensive framework for continuously capturing and semantically representing live MQTT communication. Canim et al. describe two main strategies for exposing data as RDF graphs: materialization and virtualization [24]. During materialization, data from the sources is transformed into RDF triples and stored in an RDF-based graph database (GDB) or triplestore. In virtualization, a virtual knowledge graph (VKG) is generated on top of existing data sources, storing the source data rather than the generated RDF itself. This approach is also known as ontology-based data access [25,26]. For the materialization approach, several processors execute mapping rules to generate a KG. In terms of RML, according to the RML Implementation Report [27], the RMLMapper [28] is by far the most reliable processor. Ontop is an example of a virtualization approach that translates SPARQL queries into SQL queries, expressed by the KG and executed by the underlying RDB [26]. Ivanovic et al. presented MontoFlow, a semantic integration framework that adopts the Ontop approach over real-time sensor data, enabling SPARQL-based access without fully materializing RDF triples [29]. However, virtualization-based approaches limit the solution to relational data sources. Still, it illustrates the emerging need for stream RDF solutions aimed at real-time KG Population, where the goal is not only integration but also continuous semantic lifting of sensory data.

To address heterogeneous and streaming IoT data beyond purely relational sources, recent work has explored Extract, Transform, Load (ETL) oriented KG construction pipelines. Listl et al. provide an ETL implementation in Node-RED to construct manufacturing knowledge graphs from heterogeneous industrial automation data [30]. Node-RED is an open-source, low-code visual programming tool for wiring together physical devices using nodes [31], employed in both home automation projects and professional industrial IoT solutions. Scrocca et al. integrate domain knowledge into IoT systems using Semantic Web technologies and a low-code DataOps toolbox that interacts with devices through API-documented interfaces based on standard protocols, such as HTTP(S), MQTT, and CoAP [32].

Since semantic integration pipelines in MQTT-based IoT environments depend on broker-mediated communication, broker behavior and performance are relevant for practical deployment. Bender et al. evaluate several popular open-source MQTT broker implementations with respect to interoperability, resource consumption, and latency [33]. Their results show that Mosquitto [34] is suitable for resource-constrained devices, while EMQX [35] consistently delivers the best overall performance among the tested brokers.

4. Methodology

This work follows a design-oriented research approach to develop a semantic integration solution for real-time MQTT data streams in IoT environments. The proposed solution consists of two complementary contributions: (i) the MQTT4SSN ontology for semantic modeling and (ii) the MQTT2RDF framework, including supporting artifacts for runtime data transformation and knowledge graph population. The methodology begins with the definition of overall requirements for both semantic modeling and the framework 4.1. It subsequently integrates ontology engineering 4.2 with the design and implementation of an integration and analytics framework 4.3. The process is guided by the SABiOx ontology engineering methodology [36], which structures development into five phases: requirements gathering, setup, knowledge capture, ontology design, and ontology implementation. In addition, principles from semantic web engineering are applied to ensure that the resulting artifacts are both theoretically sound and practically applicable. Finally, considerations regarding publication and long-term reusability of the developed artifacts are addressed 4.4.

4.1. Requirements Definition

The requirements for the proposed solution are derived from the challenges identified in the introduction 1 and the industrial usage scenario 2. These requirements can be grouped into two categories: semantic modeling requirements and framework-level requirements. From a semantic modeling perspective, the solution must provide a formal representation of MQTT communication,

 doi:10.20944/preprints202605.1509.v1

including control packets, topics, payloads, and session context. Furthermore, it must enable the semantic description of heterogeneous payload formats and data structures, while also supporting alignment with the established SSN/SOSA concept for representing sensor observations and actuations. From a framework-level perspective, the solution must be able to capture live MQTT message streams from a broker and interpret topics and payloads based on configurable mappings. In addition, it must instantiate RDF triples in real time according to the ontology and stream the resulting data into a queryable knowledge graph to enable further analysis. These requirements form the foundation for both the ontology engineering process 4.2 and the integration framework architecture 4.3.

4.2. Ontology Engineering Process

The development of the MQTT4SSN ontology follows the SABiOx methodology, considering requirements gathering, setup, knowledge capture, ontology design, and ontology implementation [36]. A key aspect of the engineering process is the parallel iterative development of the ontology and the framework. The ontology defines the semantic structure that the framework must instantiate, while the framework provides the practical applicability. During development, CQs 1 were continuously applied against the knowledge graph using the framework, leading to iterative refinements of both the semantic model and the framework's transformation logic.

Requirements Gathering: To capture the relevant conceptual requirements of the ontology systematically, a structured set of competency questions (CQs) was defined. These questions were derived from both the MQTT 5.0 specification [3] and real-world industrial scenarios, ensuring compliance with the standard and practical applicability. The CQs serve as a central instrument for defining the scope and granularity of the knowledge representation and divide the model into five key areas: 1) the Control Packet Hierarchy, 2) the Fixed and Variable Headers, 3) the Payload with the Application Message, 4) the Network Infrastructure, and 5) the Topic Subject. Within each of these areas, the questions are further divided into general CQs, which capture fundamental structural and semantic requirements, and use-case-specific CQs, which focus on concrete analytical and operational scenarios. The CQs form the basis for subsequent formalization into SPARQL queries and ensure that the ontology can answer relevant analytical questions in real-world applications.

Setup: In the setup phase, we set up the technical environment for the iterative ontology engineering process. As the ontology was developed using RDF/OWL as a formal representation language, the free, open-source ontology editor Protégé [37] was used, which supports iterative modeling and validation. For version control and collaborative development, a git based repository was used. To ensure logical consistency and support iterative testing of the ontology, the OntOlogy Pitfall Scanner! (OOPS!) [38] was used. The MQTT2RDF framework was set up in parallel to enable the transformation of MQTT messages into RDF, allowing early validation 7.3 of the ontology against real data streams using SPARQL queries. For the automatic generation of the ontology documentation, including a WebVowl [39] visualization, the WIDOCO assistant [40] was used.

Knowledge Capture: As the model is explicitly designed for the MQTT message protocol, the official MQTT 5.0 specification [3] was systematically analyzed to extract relevant concepts, relationships, and constraints. This includes control packet structures, message flows, session management, and error handling mechanisms. Additionally, concepts from edge computing and IoT architectures were considered.

Ontology Design: The ontology design phase was guided by the previously defined CQs, which iteratively refined the conceptual structure and ensured that all relevant aspects of the domain were sufficiently represented. The CQs served as a reference for identifying required classes, properties, and relationships, as well as for validating whether the evolving model meets the requirements. In the ontology design phase, ontology design patterns (ODP) [20] were used as suitable, reusable modeling solutions. Existing ontologies and vocabularies, including SSN/SOSA [6,7], QUDT [16,17], and MQV [18], were reused and, if necessary, expanded. These reusable models were identified in the literature review 3.1.

 doi:10.20944/preprints202605.1509.v1

Ontology Implementation: The semantic model was implemented in RDF/OWL and enriched with annotations, labels, and documentation to ensure usability and interoperability. The final ontology is applied with the Semantic Integration and Analytics Framework 4.3, and published in accordance with the FAIR principles 4.4.

4.3. Semantic Integration and Analytics Framework Design

In parallel with the ontology development, the conceptual design of the MQTT2RDF framework was defined to operationalize the MQTT4SSN ontology in live MQTT-based IoT communication. The design goal was not to specify a simple RDF transformation of individual MQTT payloads, but to define a reusable semantic integration process that reconstructs the context in which MQTT messages are produced, transmitted, and received. In this way, the framework design supports transforming implicit communication semantics into an explicit, queryable knowledge graph for integration, traceability, and semantic analytics. The resulting conceptual pipeline is depicted in Figure 2.

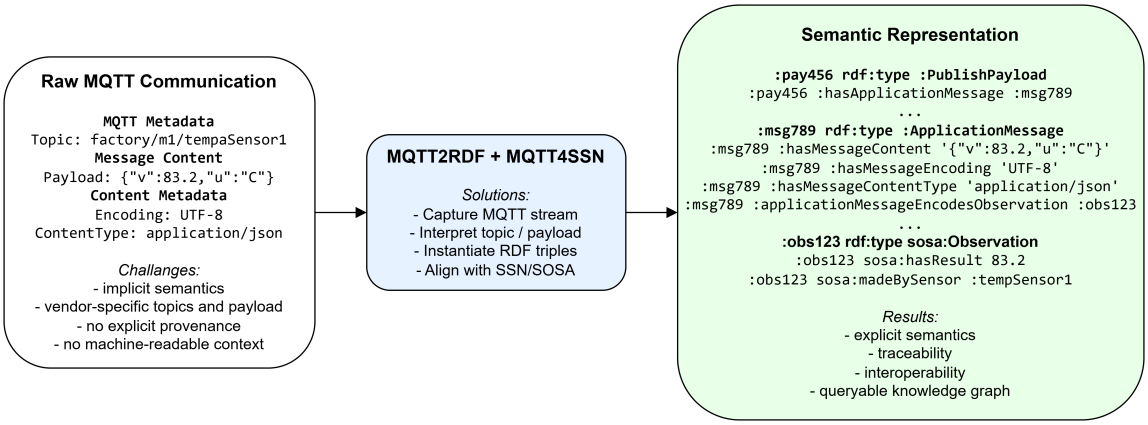

Figure 2. Transformation of raw MQTT communication into a semantic representation using MQTT2RDF and MQTT4SSN: The left panel illustrates a typical MQTT message with implicit semantics and limited machine-readable context. The middle component outlines the processing steps, including stream capture, topic and payload interpretation, RDF triple generation, and alignment with SSN/SOSA ontology. The right panel illustrates the resulting semantic representation with explicit semantics, enabling improved traceability, interoperability, and queryability within a KG.

At first, the pipeline captures MQTT traffic from the broker and enriches incoming messages with metadata. Further, the topic structure and message payload are transformed according to configurable parsing and mapping rules. This allows the framework to handle heterogeneous payload formats and different topic naming conventions used in industrial deployments. The interpreted data is then semantically enhanced into RDF by instantiating MQTT4SSN concepts. The generated triples are continuously inserted into a triple store, thereby forming an incrementally constructed KG of the running MQTT-based system.

The framework was conceptually structured as a set of loosely coupled components. The capture component interfaces with the MQTT environment and obtains the message stream. The interpretation component extracts the meaning of topics and payloads using configurable rules, enabling different deployments and payload formats to be supported without changing the ontology itself. The semantic lifting component creates RDF resources and relationships according to MQTT4SSN concepts. Finally, the storage component writes the generated triples to a triplestore, making the continuously populated KG available for SPARQL queries and higher-level analysis.

In addition to the semantic integration pipeline, the framework includes a semantic analytics dashboard. The dashboard provides a user interface for exploring the generated KG and, for example, inspecting network entities, network sessions, topics, messages, and payloads. The dashboard presents a semantically enriched representation of the communication stream instead of raw topic strings or

 doi:10.20944/preprints202605.1509.v1

payloads. This allows users to search for relevant data streams by semantic criteria, such as observed property, feature of interest, sensor, or topic relation. The dashboard addresses the three usage scenarios introduced earlier: message content analytics, semantic topic analytics, transport, and provenance analytics.

The modular design makes the framework extensible and adaptable to different MQTT deployments. This enables reuse and integration across deployments where MQTT is used for device communication, but where topic structures, payload formats, and semantic requirements vary.

4.4. Publication and Reusability

To ensure transparency and reuse, both the MQTT4SSN ontology and the MQTT2RDF framework are Findable, Accessible, Interoperable, and Reusable (FAIR) [41], we have made MQTT4SSN openly available on GitHub[3] and permanently archived it via Zenodo[4] under a DOI [10], along with an open-use license[8]. The ontology is richly annotated with labels, comments, machine-readable metadata, and human-readable documentation[7] to promote reuse, extension, and integration into other semantic IoT applications. The documentation includes a WebVowl visualization[1] [39] that was automatically generated by the WIDOCO assistant [40].

5. MQTT4SSN Ontology

Previous work on MQTT4SSN has focused on MQTT version 3.1.1. and only covers basic message properties, limited to the three control packets PUBLISH, SUBSCRIBE, and UNSUBSCRIBE. This version is not suitable for a complete end-to-end traceability of the message transfer, for example, for analyzing transport provenance.

5.1. Core Concept

The core concepts of the MQTT4SSN ontology capture the semantic representation of the MQTT message protocol, incorporating both observational and actuatable semantics, thereby enabling end-to-end traceability between sensing systems and message transfer. SSN/SOSA [7,8] provides rich semantics for observations and actuations in connection with their observational and actuatable properties, as well as the feature of interest. MQTT4SSN complements this by explicitly modeling the structure and metadata of the transmitted data. The representation of MQTT Control Packets, including their contents and structure, the network infrastructure, the application message, as well as the alignment of the topic with SOSA/SSN, forms the core of the ontology. Where possible, QUDT [17] vocabularies were used to enrich the ontology. The alignment with the MQV [18] ontology remains, but is not further discussed in the work due to the draft's incompleteness. We further developed the previous version of MQTT4SSN, adding additional classes and properties and modifying existing concepts. The concept is expanded to encompass all control packets of the MQTT 5.0 Specification [3], including complete contents, and incorporates a network session into the network infrastructure. Beyond the specification of conceptual axioms, logical constructs were incorporated to enhance semantic rigor and ensure consistency across the ontology:

Class and Property Specialization and Generalization: We define special classes and properties for a general class, respectively, to achieve more semantic depth. In the other direction, we have generalized some classes or properties for general queries or to cluster common semantic backgrounds.

Class and Property Disjointness: We explicitly define class and property disjointness among subclasses and, respectively, among subproperties to enhance ontological clarity and enable more precise reasoning and querying over instances.

Inverse Propery: Almost every object property received an inverse, allowing more targeted queries in both directions.

[1] https://doernern.github.io/MQTT4SSNOntology/documentation/webvowl/, accessed: May 22, 2026

 doi:10.20944/preprints202605.1509.v1

Equivalences: The original MQT4SSN ontology integrated the concepts of MQV [18] through equivalent classes and properties with `owl:equivalentClass` and `owl:equivalentProperty`, respectively. Due to the draft's incompleteness, the ontology is not discussed further in this work. However, the MQV ontology remains in MQTT4SSN for possible future developments.
Semantic Annotation: To provide a clear and reusable ontology, we enriched all classes, properties, and the ontology itself with `rdfs:label` and `rdfs:comment` annotations. In addition, we annotated all MQTT-related datatype properties with `skos:example` based on the MQTT 5.0 Specification [3], helping with better understanding at the instantiation.
Namespace: We use the prefix `mqtt:` to denote classes and properties defined in the MQTT ontology namespace. The prefixes `qudt:` and `sosa:` denote the namespaces for QUDT [16,17] and SOSA [6–8]. To specify the XML Schema Datatypes [42], the prefix `xsd:` is used.

5.2. Ontology Description

At the core of this model are five major class clusters: 1) the Control Packet Hierarchy, 2) the Fixed and Variable Headers, 3) the Payload with the Application Message, along with its relation to SSN/SOSA, 4) the Network Infrastructure, and 5) the Topic Subject alignment with SSN/SOSA semantics.

1. Control Packet Hierarchy: In the previous, restricted version of MQTT4SSN [19], only the three main control packets PUBLISH, SUBSCRIBE, and UNSUBSCRIBE were considered. This version extends the control packet hierarchy to encompass all fifteen control packets defined in the MQTT 5.0 OASIS Standard [3]. Let CP denote the set of all MQTT control packet types:

$$CP = \{\text{Auth}, \text{Connack}, \text{Connect}, \text{Disconnect}, \text{Pingreq}, \text{Pingresp}, \text{Puback}, \text{Pubcomp}, \text{Publish}, \text{Pubrec}, \text{Pubrel}, \text{Suback}, \text{Subscribe}, \text{Unsuback}, \text{Unsubscribe}\}. \tag{1}$$

Let $\mathcal{C}$ denote the set of ontology classes, and define a mapping $\mathcal{CP} : CP \to \mathcal{C}$ that assigns each control packet type $\text{cp} \in CP$ to a corresponding class in the `mqtt` namespace, defined as $\mathcal{CP}(\text{cp}) =$ `mqtt:`[cp]Packet. All resulting classes are modeled such that:

$$\forall\, \text{cp} \in CP : \ \mathcal{CP}(\text{cp})\ \texttt{rdfs:subClassOf}\ \texttt{mqtt:ControlPacket}. \tag{2}$$

In general, MQTT control packets are structurally represented by means of the classes `mqtt:FixedHeader`, `mqtt:VariableHeader`, and `mqtt:Payload` shown in Figure 3. Accordingly, the control packet types CP that include a payload, a variable header, or a fixed header are captured by the subsets PL, VH, and FH, respectively, where

$$PL \subseteq CP,\ VH \subseteq CP,\ FH \subseteq CP. \tag{3}$$

 doi:10.20944/preprints202605.1509.v1

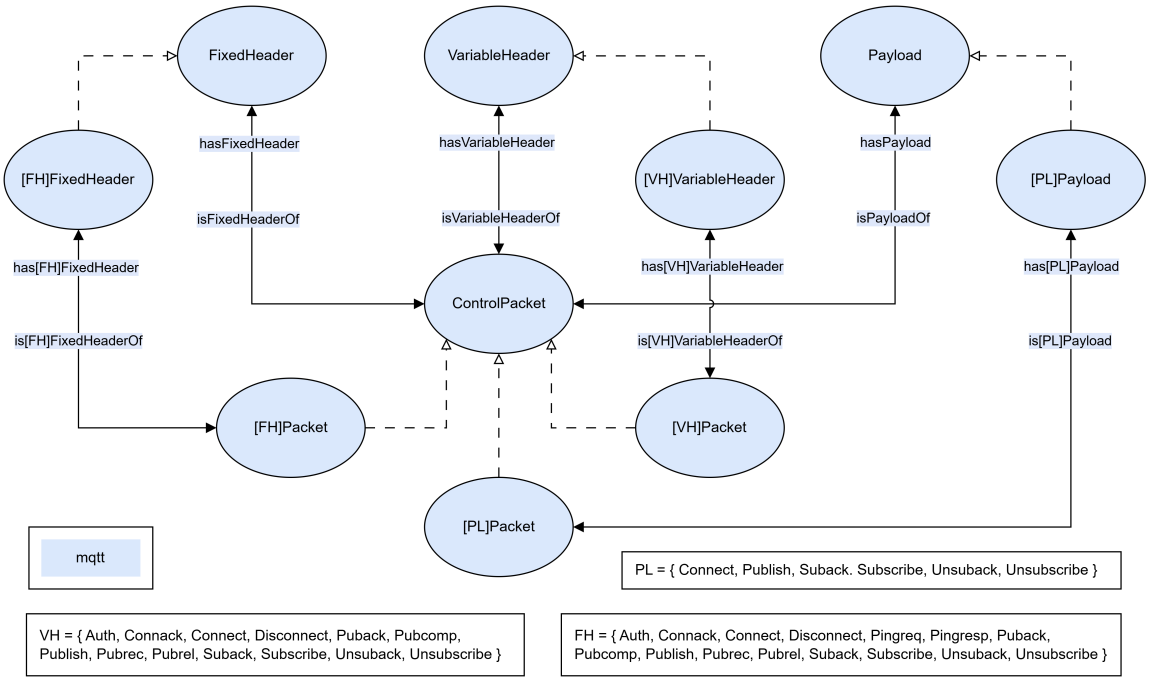

Figure 3. Control Packets with Headers and Payload

These subsets are defined as follows:

$$\begin{aligned} PL &= \{\text{Connect}, \text{Publish}, \text{Suback}, \text{Subscribe}, \text{Unsuback}, \text{Unsubscribe}\}, \\ VH &= \{\text{Auth}, \text{Connack}, \text{Connect}, \text{Disconnect}, \text{Puback}, \text{Pubcomp}, \\ &\quad \text{Publish}, \text{Pubrec}, \text{Pubrel}, \text{Suback}, \text{Subscribe}, \text{Unsuback}, \text{Unsubscribe}\}, \\ FH &= \{\text{Auth}, \text{Connack}, \text{Connect}, \text{Disconnect}, \text{Pingreq}, \text{Pingresp}, \text{Puback}, \text{Pubcomp}, \\ &\quad \text{Publish}, \text{Pubrec}, \text{Pubrel}, \text{Suback}, \text{Subscribe}, \text{Unsuback}, \text{Unsubscribe}\}. \end{aligned} \tag{4}$$

We further define mappings for the structural components of control packets in accordance with the MQTT specification. The mappings $\mathcal{F} : FH \to \mathcal{C}$, $\mathcal{V} : VH \to \mathcal{C}$, and $\mathcal{P} : PL \to \mathcal{C}$ assign each fixed header $\text{fh} \in FH$, variable header $\text{vh} \in VH$, and payload $\text{pl} \in PL$ to their corresponding classes in the `mqtt` namespace, defined as:

$$\begin{aligned} &\mathcal{F}(\text{fh}) = \texttt{mqtt:}[\text{fh}]\text{FixedHeader}, \quad \mathcal{V}(\text{vh}) = \texttt{mqtt:}[\text{vh}]\text{VariableHeader}, \\ &\mathcal{P}(\text{pl}) = \texttt{mqtt:}[\text{pl}]\text{Payload}. \end{aligned} \tag{5}$$

All resulting classes are modeled uniformly as subclasses of their respective superclasses:

$$\begin{aligned} &\forall\, \text{fh} \in FH : \ \mathcal{F}(\text{fh})\ \texttt{rdfs:subClassOf}\ \texttt{mqtt:FixedHeader}, \\ &\forall\, \text{vh} \in VH : \ \mathcal{V}(\text{vh})\ \texttt{rdfs:subClassOf}\ \texttt{mqtt:VariableHeader}, \\ &\forall\, \text{pl} \in PL : \ \mathcal{P}(\text{pl})\ \texttt{rdfs:subClassOf}\ \texttt{mqtt:Payload}. \end{aligned} \tag{6}$$

In addition to the class hierarchy illustrated in figure 3, we introduce object properties that capture the structural relationships between control packets and their respective headers and payloads. In particular, we define the generic relations:

$$\begin{aligned} &\texttt{mqtt:ControlPacket mqtt:hasFixedHeader mqtt:FixedHeader}, \\ &\texttt{mqtt:ControlPacket mqtt:hasVariableHeader mqtt:VariableHeader}, \\ &\texttt{mqtt:ControlPacket mqtt:hasPayload mqtt:Payload}. \end{aligned} \tag{7}$$

 doi:10.20944/preprints202605.1509.v1

To support bidirectional navigation and reasoning, inverse object properties are explicitly linked using `owl:inverseOf`. For a more fine-grained representation, we define specialized subproperties for each control packet type, which are generally defined as:

$$\forall \text{fh} \in FH: \texttt{mqtt:has[fh]FixedHeader rdfs:subPropertyOf mqtt:hasFixedHeader}. \quad (8)$$

Analogous definitions apply for all vh $\in$ VH and pl $\in$ PL, yielding subproperties of `mqtt:hasVariableHeader` and `mqtt:hasPayload`, respectively. Correspondingly, inverse specialized properties are defined and linked via `owl:inverseOf`.

2. *Fixed and Variable Headers*: Figures 4 and 5 illustrate the ontology design for modeling fixed and variable headers of the MQTT protocol in accordance with the MQTT 5.0 specification.

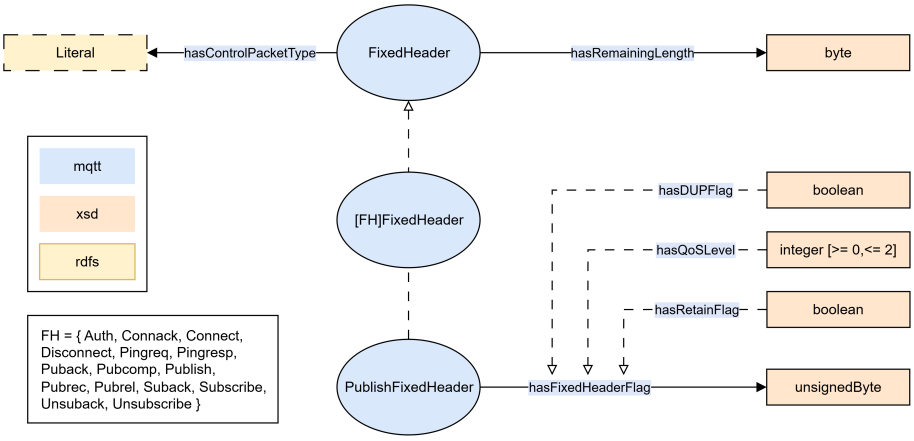

Figure 4. Fixed Header with datatype properties

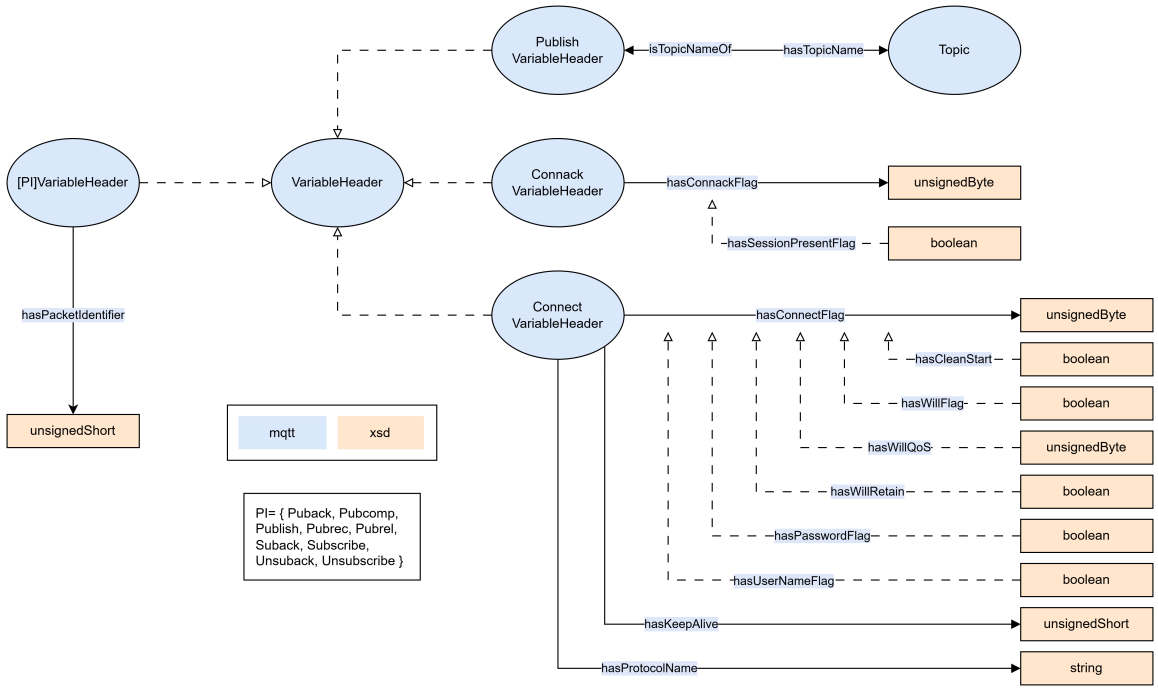

Figure 5. Variable Header with datatype properties

The fixed header class represents the common structural elements shared by all control packets. Specifically, each fixed header includes a remaining length and a control packet type, both modeled as

 doi:10.20944/preprints202605.1509.v1

datatype properties. The ontology allows flexible `rdfs:Literal` representation of the control packet type during instantiation. For example, depending on the application's needs, encoded either as a string (e.g., PUBLISH) or as its corresponding byte representation (e.g., 0011). In addition, certain control packet types include specialized header fields. For example, the PUBLISH fixed header extends the generic fixed header by adding fixed-header flags, represented either as a single byte value or as their individual components, modeled as subproperties. This dual modeling approach supports both low-level protocol representations and high-level semantic interoperability.

The variable header class represents the variable part of MQTT control packets, whose structure varies by packet type. Let $PI \subseteq VH$ denote the subset of variable headers for which each pi $\in PI$ is mapped to a class `mqtt:`[pi]VariableHeader which has a packet identifier, with:

$$PI = \{\text{Puback}, \text{Pubcomp}, \text{Publish}, \text{Pubrec}, \text{Pubrel}, \text{Suback}, \text{Subscribe}, \text{Unsuback}, \text{Unsubscribe}\}. \tag{9}$$

Furthermore, the PUBLISH variable header establishes a semantic connection to the topic via an object property, with a corresponding inverse property that enables bidirectional navigation. Similar to the PUBLISH fixed header, the CONNECT and CONNACK variable headers support both compact and decomposed representations of flags and configuration fields. These can be encoded as aggregated byte values or represented by dedicated subproperties that correspond to individual flags, thereby enabling fine-grained semantic descriptions.

MQTT 5.0 introduces a versatile property mechanism, which is generally represented in the ontology by the properties class and specifically by variable header subclasses, shown in Figure 6. Certain property sets allow multiple occurrences of user-defined properties. These are modeled via the user property class, which defines the name and value as a key-value pair. A special case is given by will properties, which extend the connect properties within the CONNECT payload in a similar way, shown in Figure 8.

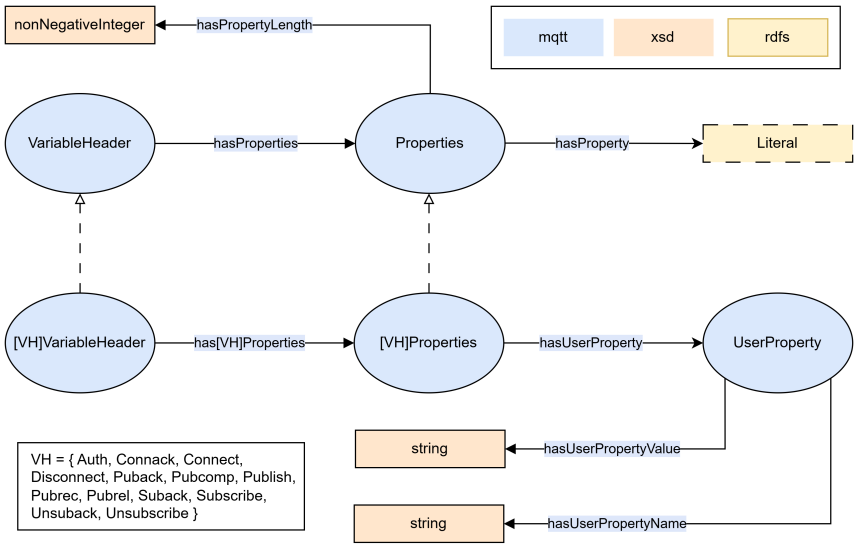

Figure 6. Properties of Variable Header

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 22 May 2026 doi:10.20944/preprints202605.1509.v1

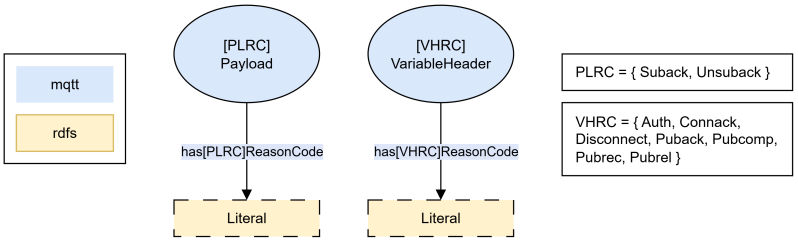

Figure 7. Reason Codes located in Payload and Variable Header

Reason codes are modeled as datatype properties that may occur either in the variable header or in the payload. Accordingly, the subsets $VHRC \subseteq VH$ and $PLRC \subseteq PL$ capture all control packet types that include reason codes in the variable header and payload, respectively, with:

$$\begin{aligned} VHRC &= \{\text{Connack}, \text{Puback}, \text{Pubrec}, \text{Pubrel}, \text{Pubcomp}, \text{Disconnect}, \text{Auth}\}, \\ PLRC &= \{\text{Suback}, \text{Unsuback}\}. \end{aligned} \tag{10}$$

The datatype properties `mqtt:has[vh]ReasonCode` for $vh \in VHRC$ and `mqtt:has[pl]ReasonCode` for $pl \in PLRC$ are defined as subproperties of `mqtt:hasVariableHeaderReasonCode` and `mqtt:hasPayloadReasonCode`, respectively, and further generalized as subproperties of `mqtt:hasReasonCode`. Reason codes can be instantiated as literal representations, reflecting application-specific encoding.

3. Payload and Application Message: Figure 8 illustrates the ontology design for modeling MQTT payloads and their relation to the transferred application message.

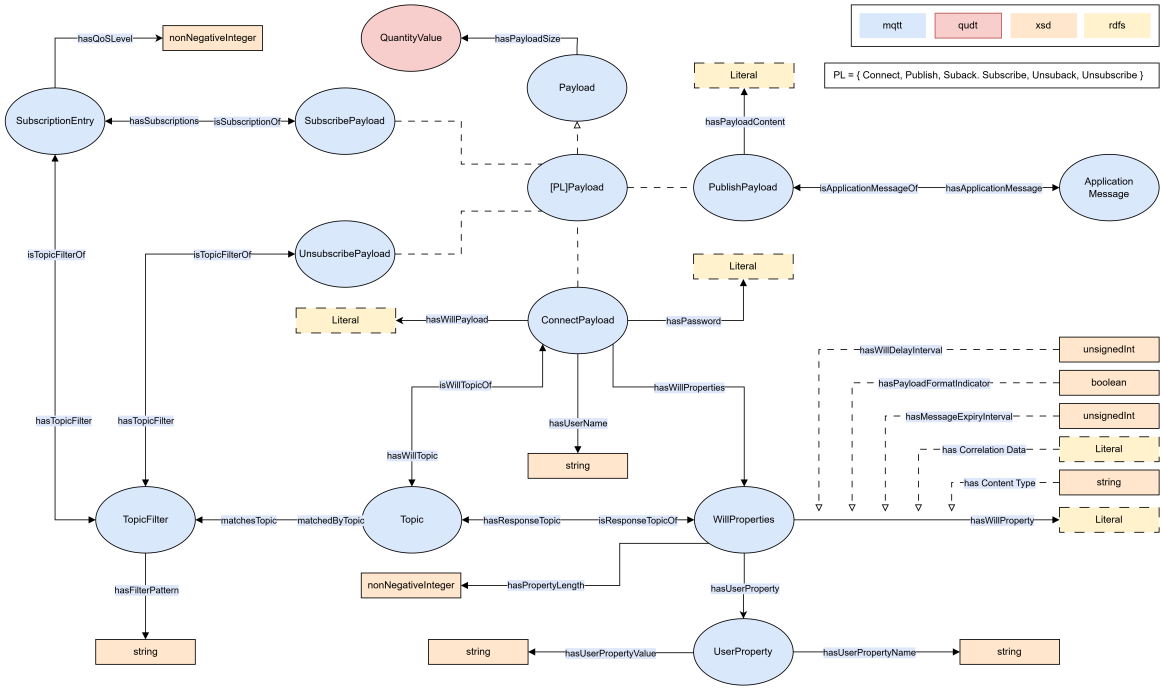

Figure 8. Payload

In accordance with the MQTT 5.0 specification, the ontology represents the payload as the message component that carries application-specific content or packet-specific data structures. The concrete structure of a payload depends on the corresponding MQTT control packet type. The CONNECT payload captures all optional connection-related payload elements, including the will payload, username, and password. In addition, it may be linked to a will topic, which is modeled as an

 doi:10.20944/preprints202605.1509.v1

instance of the topic class. The inverse relation enables bidirectional navigation between a CONNECT payload and the topic under which the will message is to be published. The CONNECT payload may further include multiple will properties that capture protocol-specific metadata associated with the will message. These properties are represented as datatype properties and are attached to an overarching will properties class. Similar to the general MQTT properties, shown in Figure 6, user-defined metadata plays a special role within the will properties. Furthermore, will properties may include a response topic, which is again connected to the topic class. This allows the ontology to express not only the content of a will message but also its intended interaction context.

Distinguishing between subscription- and publication-related payload structures, both are linked to the topic filter class, which in turn is semantically associated with the topic class. This separation reflects the MQTT distinction between concrete topics used for publication and topic filters used for selective subscription and unsubscription, but with clear semantic linkage. The UNSUBSCRIBE payload is directly connected to the topic filter, while the SUBSCRIBE payload is not linked directly to a topic filter, but via an intermediate subscription entry class. This design choice reflects the fact that a subscription is not solely characterized by a topic filter, but also by a quality of service value (QoS) represented as a datatype property. Through corresponding inverse relations, it becomes possible to reconstruct which SUBSCRIBE and UNSUBSCRIBE payloads, and thus which corresponding packets and clients, are associated with a particular topic filter. This supports tracing de- and subscription events semantically across the communication structure.

The content transmitted by the PUBLISH payload is represented by a datatype property, whose values may, depending on the implementation, be instantiated as literal values. Through this, the ontology remains compatible with protocol-level byte-oriented representations while also supporting higher-level semantic interpretation.

To bridge the protocol and application layers, we introduce the `mqtt:ApplicationMessage` class shown in Figure 9. This class abstracts from the raw payload encoding and offers an explicit semantic representation of the transmitted message as an application-level entity. The PUBLISH payload is linked to the `mqtt:ApplicationMessage` via a bidirectional relationship. The PUBLISH payload content is represented at two complementary levels. At the protocol level, the payload content is modeled via `mqtt:hasPayloadContent`, which may be instantiated as a literal value, reflecting the actual transmitted byte-oriented representation. In contrast, the application-level interpretation is captured by the `mqtt:ApplicationMessage`, which represents the content in a normalized, semantically accessible form.

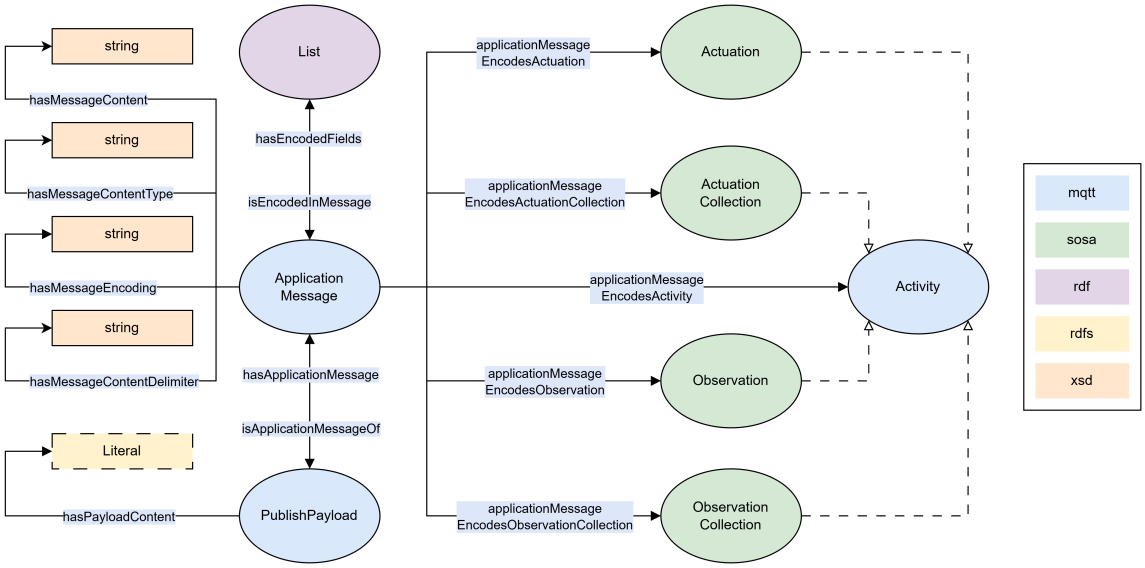

Figure 9. Application Message

 doi:10.20944/preprints202605.1509.v1

In particular, the application message provides the datatype property `mqtt:hasMessageContent`, which represents the decoded message content exclusively as a string. In addition, the structure and encoding of the message are explicitly described by further datatype properties, serving as descriptive metadata for the message content. Datatype properties specify the format of the message (e.g., `CSV`, `JSON`), character encoding (e.g., `UTF-8`), and, depending on the content type, additional structural information such as a content delimiter (e.g., comma-separated values). For structured message formats containing multiple values, such as CSV, the ontology captures the set of fields (e.g., column names) as `rdf:List`. This allows for an explicit definition of the message schema at the application level and helps interpret the raw message content semantically without relying on implicit assumptions about its structure.

The `mqtt:ApplicationMessage` serves as the entry point for linking MQTT messages to the SSN/SOSA ontology. To provide a uniform abstraction, these SOSA classes are generalized within the ontology by an activity class. Correspondingly, the object properties linking application messages to SOSA entities are also generalized, enabling unified querying across different types of sensor-related activities. This design allows MQTT payloads to be interpreted not only as transport containers but as semantically rich carriers of domain knowledge.

4. *Network Infrastructure*: Figure 10 illustrates the ontology design for modeling the MQTT network infrastructure.

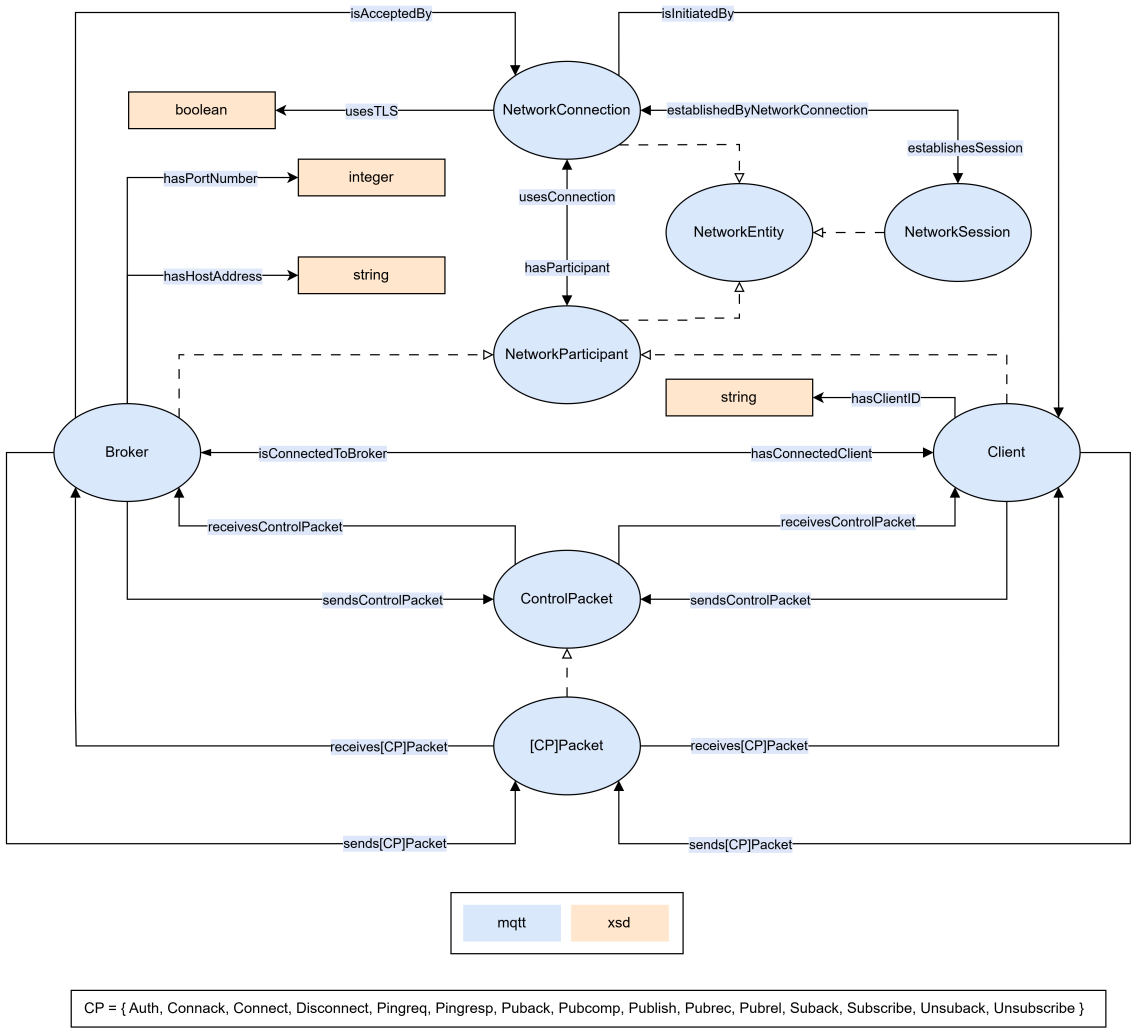

Figure 10. Network Infrastructure

 doi:10.20944/preprints202605.1509.v1

Within the network infrastructure, clients and brokers, both types of network participants, use a network connection as a communication channel, which can optionally be encrypted with the TLS protocol. The network connection class facilitates the interaction between network participants. A broker has a host address and a port number, while a client has a client identifier. Furthermore, the ontology captures network sessions which represent the logical communication context between a client and a broker. A session is established via a network connection and can persist over multiple connections. This persistence is reflected in object properties that link sessions to sequences of network connections in both directions. Finally, object properties are defined to explicitly associate network participants with control packets, indicating which client or broker is involved in sending or receiving a specific packet. This facilitates tracing communication flows across the network infrastructure at both the protocol and semantic levels.

5. Topic Subject: Figure 11 illustrates the ontology design for modeling MQTT topics and their semantic alignment with SOSA.

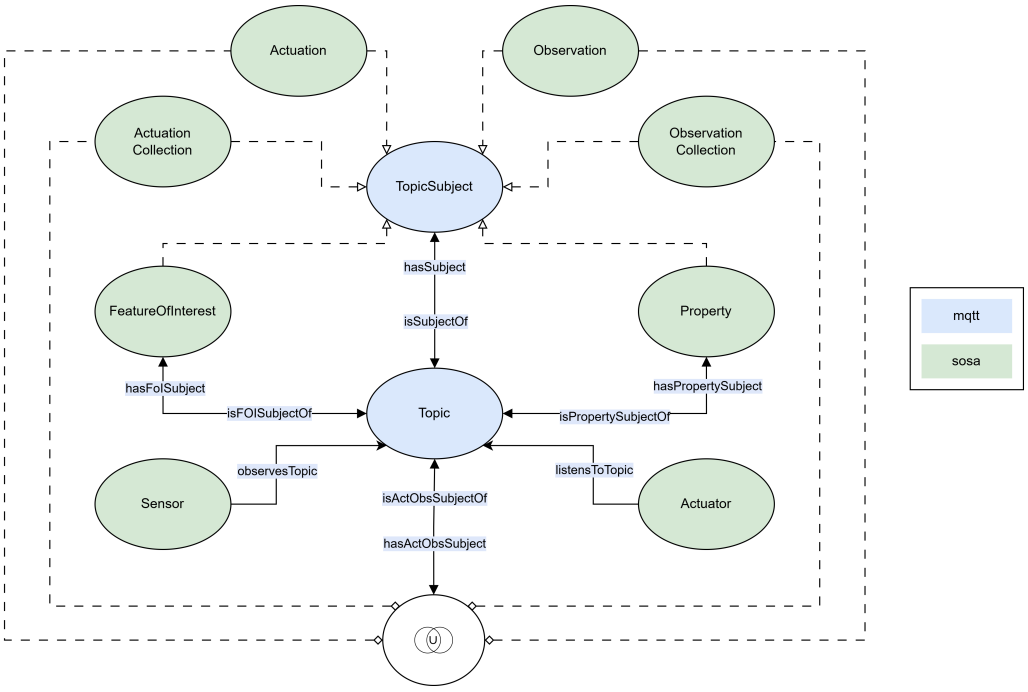

Figure 11. Topic Subject

The topic class serves as the main abstraction for addressing MQTT messages. Topics are modeled as semantically structured entities that can consist of one or multiple instances of the topic subject class. Conversely, a topic subject can be associated with multiple topics, enabling a many-to-many relationship between syntactic topic representations and their semantic meanings.

The topic subject provides a semantic abstraction layer, encompassing the following SSN/SOSA elements: observation, actuation, collections of these, as well as features of interest and observable propertys. Conceptually, a topic can be understood as a combination of semantic elements that describe an activity and its context. Specifically, a topic may encode an Activity, representing the aforementioned SSN/SOSA elements. This arrangement allows for a structured semantic interpretation of MQTT topics beyond their string-based representation.

6. MQTT2RDF Semantic Integration and Analytics Framework

MQTT2RDF is a modular semantic integration framework that enables seamless integration of heterogeneous data from real-time MQTT messages into an RDF triplestore. The framework addresses

 doi:10.20944/preprints202605.1509.v1

dynamic ontology population and serves as a foundation for end-to-end semantic analytics. A shared semantic layer enables automated systems to integrate semantic analytics in a vendor-agnostic, interoperable manner, enabling heterogeneous devices and services to exchange data with machine-interpretable meaning and supporting reliable cross-domain reasoning, event detection, and real-time adaptive control. As part of its semantic core, MQTT2RDF integrates the MQTT4SSN ontology. As shown in Figure 12, the semantic integration framework comprises the following artifacts:

- Containerized deployment environment (deployment layer)
- MQTT broker (edge ingestion layer)
- Node-RED processing flows (stream processing layer)
- RML mapping definitions and execution scripts (semantic transformation layer)
- MQTT4SSN ontology (semantic model layer)
- RDF triplestore (storage layer)
- Semantic Analytics Dashboard using SPARQL queries (analytics layer)

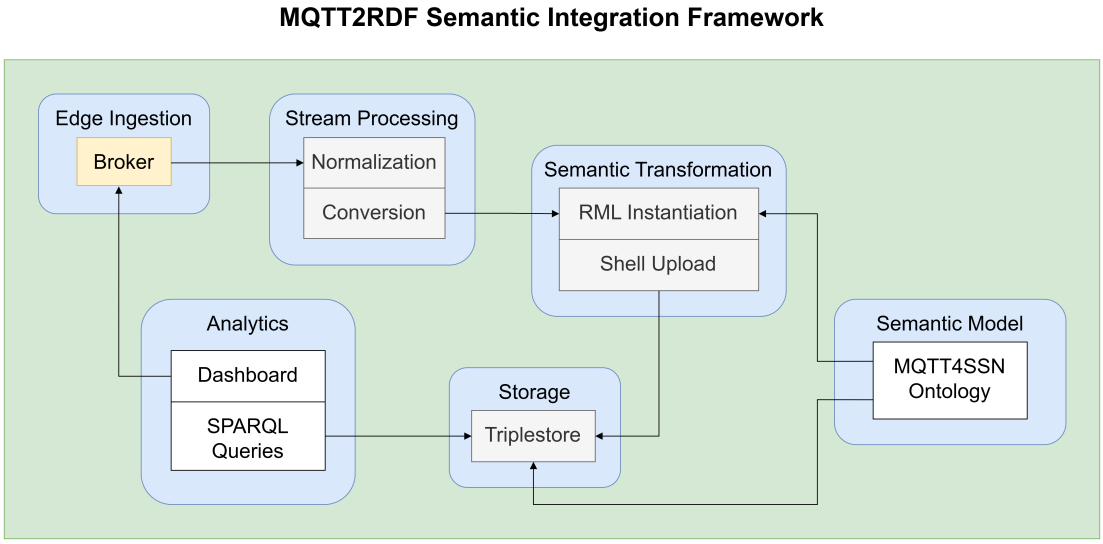

Figure 12. Architecture deployment of the Semantic Integration Framework (MQTT2RDF)

The framework architecture consists of an MQTT broker as the edge ingestion layer, a stream processing layer based on Node-RED [31], a semantic transformation layer using RMLMapper [28], and a triplestore as the storage layer, which incorporates the MQTT4SSN ontology as a semantic model layer. The entire framework is deployed as a Docker [43] environment to ensure reproducibility, simplified deployment, and portability across different environments, thereby supporting FAIR and open science principles.

In our implementation, the EMQX MQTT platform [35] is used as a broker and GraphDB [44] as the RDF triplestore. While EMQX ensures reliable MQTT message ingestion and forwarding, the subsequent layers perform semantic integration. To the best of our knowledge, no open-source MQTT broker transparently exposes the full range of protocol-level metadata (e.g., control packet flags). As a result, the framework relies on the metadata accessible via broker interfaces and webhook mechanisms. An alternative approach would be to capture raw MQTT traffic at the TCP level (e.g., using Wireshark [45] or TShark [46]) to extract complete protocol information. However, this would introduce additional complexity and is not therefore integrated into the current framework. Due to the adaptable and interchangeable nature of the framework layers, other suitable solutions, for example, the broker and the triplestore, could be used.

The semantic integration process is designed as a continuous pipeline: Incoming MQTT messages are forwarded by the broker via webhooks to a receiving endpoint, enabling real-time data exchange. Node-RED subscribes to these event-specific endpoints and processes the incoming messages within

 doi:10.20944/preprints202605.1509.v1

configurable flows. Independent of the original payload format, the data is transformed and normalized into a use-case-specific, uniform JSON schema. This schema combines the raw payload with relevant metadata (e.g., topic, timestamp, and control packet type), thereby preparing the data for semantic lifting. A key feature of the framework is its ability to handle heterogeneous payload formats and character encodings. MQTT messages often contain application-specific and unstructured payloads, which are abstracted through this normalization step. To illustrate the transformed data, Figure 13 shows exemplary normalized payloads in a unified JSON structure using JSONata.

Listing 1. (**a**) SOSA observation collection

```
observationCollection.{
   "FeatureOfInterest": RFID,
   "StartTime": $string(JobStart),
   "EndTime": $string(JobEnd),
   "Sensor": $string(Sensor),
   "Actor": $string(Actuator),
   "Observation": [
     {
       "Property": "GripPoint_X",
       "Result": $string(GripPoint_X)
     },
     ...
   ]
}
```

Listing 2. (**b**) SOSA actuation collection

```
actuationCollection.{
   "FeatureOfInterest": RFID,
   "StartTime": $string(JobStart),
   "EndTime": $string(JobEnd),
   "Sensor": $string(Sensor),
   "Actuator": $string(Actuator),
   "Actuation": [
      {
        "Property": "Correction_X",
        "Result": $string(Correction_X)
      },
      ...
   ]
}
```

Figure 13. Exemplary normalized JSONata transformations in a unified JSON structure: (**a**) SOSA observation collection. (**b**) SOSA actuation collection.

The resulting structures follow the SOSA-based conceptual model of observations and actuations, forming the basis for subsequent semantic lifting. At the end of each processing flow, the normalized data is persisted as JSON files, organized by MQTT control packet type. Changes to these files trigger the RDF mapping process: the RMLMapper instantiates RDF individuals based on mapping definitions expressed in RML and semantically grounded in the MQTT4SSN and SOSA ontologies. The mapping process follows a modular design, with each MQTT control packet type associated with a specific mapping file and its corresponding execution script. The generated RDF triples are continuously streamed into the triplestore, enabling the knowledge graph to be populated almost in real time without batch processing.

Building upon the continuously updated knowledge graph, the semantic analytics dashboard provides an interactive analytics layer for real-time semantic exploration. The dashboard is implemented as an extension of the Node-RED environment and integrates SPARQL-based queries directly into reusable flow components. Each analytical view corresponds to a semantically defined query pattern derived from initially defined CQs and is rendered as structured tables and aggregated summaries.

The dashboard is divided into several analytical perspectives that reflect key usage scenario for semantic IoT systems, identified in section 2. First, *Message Content Analytics* (Figure 14) enables the inspection of heterogeneous payload structures and their semantic enrichment, allowing users to interpret observations and actuations in a unified manner independent of their original encoding.

 doi:10.20944/preprints202605.1509.v1

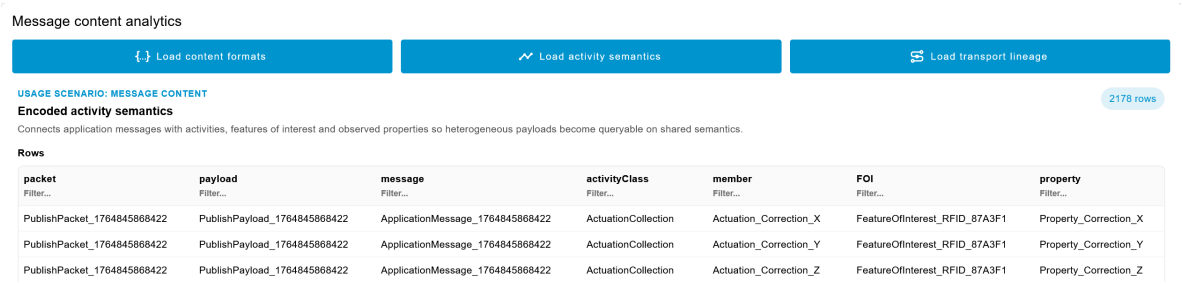

Figure 14. MQTT2RDF Dashboard: Message content analytics

Second, *Semantic Topic Analytics* (Figure 15) reveals the relationship between MQTT topic hierarchies and the underlying sensor and actuator semantics, enabling the discovery of relevant data streams based on observed properties or features of interest.

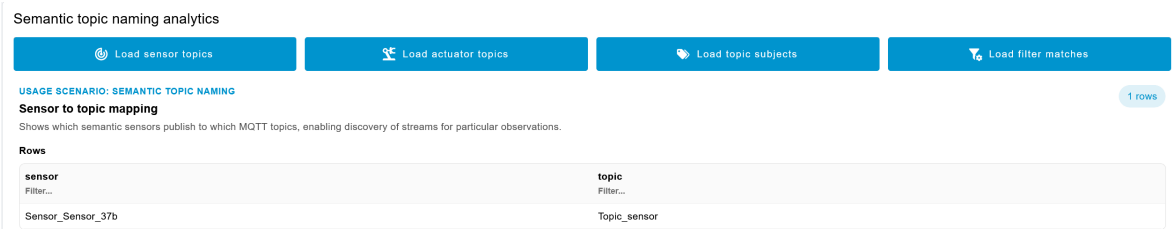

Figure 15. MQTT2RDF Dashboard: Semantic topic naming analytics

Third, *Transport and Provenance Analytics* (Figure 16) supports the investigation of message transmission behavior by correlating control packet types, QoS levels, connection states, and other protocol-level metadata, thereby enabling traceability and debugging of communication failures.

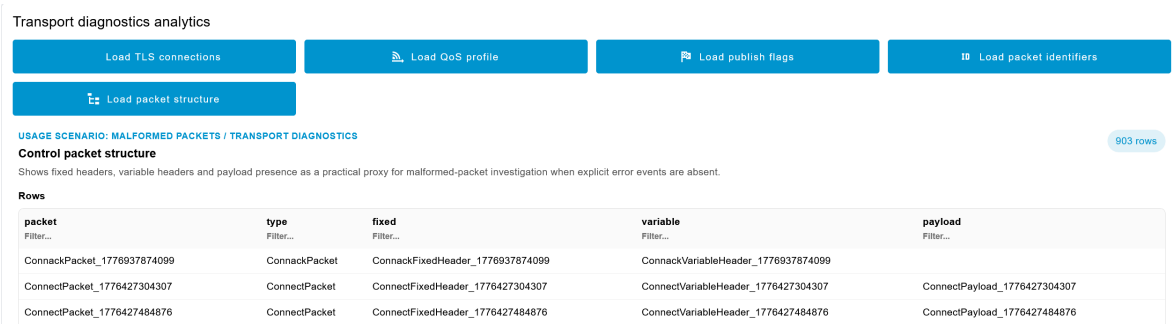

Figure 16. MQTT2RDF Dashboard: Transport diagnostics analytics

By using the explicit semantics encoded in MQTT4SSN and SOSA, the dashboard allows users to correlate information across traditionally separated layers, such as linking sensor observations to network-level transmission characteristics or identifying inconsistencies between expected and observed system behavior. The integration of semantic queries into the dashboard enables domain-independent analytics workflows and supports reproducible, explainable analysis of IoT systems.

The semantic integration framework and semantic analytics dashboard are designed to be adaptable and domain-independent. While MQTT is a prerequisite, making it particularly suitable for industrial IoT scenarios, core components such as the broker or triplestore can be replaced within the Docker environment with minimal adjustments to the stream-processing or semantic-transformation layers.

7. Evaluation

Following the SABiOx methodology [36], we evaluated the ontology through verification 7.2 and validation 7.3. Verification ensures that the ontology is correctly designed and implemented in accordance with our requirements, as determined by the CQs. Validation checks whether the intended

 doi:10.20944/preprints202605.1509.v1

purpose is fulfilled in practical applications, as determined by the industrial use case. For evaluation, we consider a dataset 7.1 from the EKI[2] project, which covers an industrial case scenario. During development and evaluation, we checked the ontology with the OOPS! [38] to prevent modeling errors through potential pitfalls. The final OOPS! Evaluation detects only minor pitfalls, including unconnected ontology elements (P04), missing annotations (P08), and missing inverse relationships (P13). The warnings P04 and P08 refer to imported ontologies, whereas we explicitly modeled the identified relationships in P13 as non-inverse. The full OOPS! evaluation report is available in the ontology documentation[7].

7.1. Dataset

The evaluation is based on an industrial production scenario involving a robotic packaging line, in which a conveyor belt transports packaging pieces. A vision sensor is placed on the robot's arm to measure the grip position in three dimensions (`sosa:Property`). These sensor signals (`sosa:hasSimpleResult`) are sent to an edge device via MQTT in CSV format 3 for further processing.

Listing 3: Comma-separated sensor measurements with column header

```
RFID;JobStart;JobEnd;GripPoint_X;GripPoint_Y;GripPoint_Z;Sensor;Actuator
RFID_87A3F1;20251203091502;20251203091502;102.44;50.15;20.10;VisionSensor_01;RobotArm_01
RFID_87A3F2;20251203091502;20251203091502;101.10;50.05;20.60;VisionSensor_01;RobotArm_01
RFID_87A3F3;20251203091502;20251203091502;100.35;49.70;20.15;VisionSensor_01;RobotArm_01
```

The robot captures the package under a unique RFID number (`sosa:FeatureOfInterest`) and transmits it as metadata alongside its `sosa:Sensor` and `sosa:Actuator` identifiers. The processing system compares the measured position with the nominal target and calculates correction offsets in real time. The edge device sends actuation signals (`sosa:hasSimpleResult`) for the correction points (`sosa:Property`) via MQTT in CSV format 4 to the robot, which adjusts its grip before picking.

Listing 4: Comma-separated actuator commands with column header

```
RFID;JobStart;JobEnd;Correction_X;Correction_Y;Correction_Z;Sensor;Actuator
RFID_87A3F1;20251203091502;20251203091502;-0.44;-0.15;+0.10;VisionSensor_01;RobotArm_01
RFID_87A3F2;20251203091505;20251203091505;-0.10;-0.05;+0.60;VisionSensor_01;RobotArm_01
RFID_87A3F3;20251203091508;20251203091508;+0.65;+0.30;+0.15;VisionSensor_01;RobotArm_01
```

The EMQX MQTT platform [35] was used as a broker, sending event-based webhooks that an HTTP node in Node-RED listens to. The broker limited our framework evaluation because the webhooks could not transmit all control packages and metadata. We found no other open-source broker solutions that were more suitable for our needs. During data normalization in Node-RED, we replaced the missing metadata with dummy data. Additionally, sensor signals were simulated via MQTT using shell scripts in order to evaluate different latencies.

7.2. Ontology Verification

During ontology engineering, we first defined a structured set of CQs in accordance with our requirements, derived from the official MQTT 5.0 Specification [3]. These CQs define the scope of knowledge to be represented in the model design.

Initially, we defined CQs that encompass the components of the MQTT message protocol. Then, we defined CQs that address the gap between the observation, actuation, and sampling semantics and the message protocol. We structured the CQs into four major categories: the Control Packet Hierarchy (CQ1), the Fixed and Variable Headers (CQ2), the Payload with the Application Message (CQ3), the Network Infrastructure (CQ4), and the Topic Subject (CQ5). Each category reflects a distinct conceptual layer of the ontology and targets specific modeling requirements. We defined a total number of 9 general and 37 use-case-specific CQs. The two complementary dimensions, general and use-case-specific CQs, serve both to define the overall scope and key concepts of the ontology and to ensure its practical applicability. This includes usage scenarios such as interpreting heterogeneous message content, structuring semantic topic hierarchies, and analyzing transport errors and provenance

 doi:10.20944/preprints202605.1509.v1

information to support reliable diagnostics and integration. Table 1 shows an excerpt of one CQ per category and dimension. The complete set of CQs, translated into SPARQL queries and prepared into category-related Jupyter notebooks, is available on GitHub.

Table 1. An excerpt of one CQ per category (CQ1-CQ5) and dimension (g = general; u = use-case-specific). The Table shows only a part of the matched Entities and does not cover all possible answers.

CQ	Question	Entities
CQ1.01g	What types of Control Packets exist?	(PublishPacket – subClassOf → ControlPacket)
CQ1.01u	Which MQTT Control Packet has which Fixed Header / Variable Header / Payload?	(ControlPacket – hasFixedHeader → FixedHeader) (ControlPacket – hasVariableHeader → VariableHeader) (ControlPacket – hasPayload → Payload)
CQ2.01g	What types of Fixed Header Flags exists?	(PublishFixedHeader – subPropertyOf → hasFixedHeaderFlag)
CQ2.03u	Are RETAIN/DUP flags set in PUBLISH?	(PublishPacket – hasRetainFlag → boolean) (PublishPacket – hasDUPFlag → boolean)
CQ3.02g	How is the Application Message linked to SOSA?	(ApplicationMessage – applicationMessageEncodesObservation → Observation)
CQ3.04u	Who has sent which Application Messages?	(ApplicationMessage – isApplicationMessageOf → PublishPayload) (PublishPayload – isPublishPayloadOf → PublishPacket) (Client / Broker – sendsControlPacket → PublishPacket)
CQ4.02g	Which relations does the Topic have?	(Topic – isSubjectOf → TopicSubject)
CQ4.01u	Which SOSA sensors publish to which MQTT Topics?	(Sensor – observesTopic → Topic)
CQ5.01g	Which Network Entities does the Network Infrastructure have?	(Client – subClassOf → NetworkEntity)
CQ5.02u	Which MQTT Clients are connected to which Broker?	(Client – isConnectedToBroker → Broker)

CQs related to the *Control Packet Hierarchy* focus on the representation and differentiation of all MQTT control packet types defined in the specification. They include questions about the classification of packets, their structural relationships, and their association with headers and payloads. The *Fixed and Variable Headers* category captures protocol-specific metadata and configuration parameters. The corresponding CQs investigate how header fields, flags, properties, and reason codes are represented within the model. For the *Payload and Application Message*, the CQs focus on the representation of payload structures and their transformation into semantically interpretable application-level messages. In particular, they address how raw payload content can be linked to structured message representations, including content types, encodings, and field structures, as well as how these messages can be connected to SSN/SOSA concepts such as observations or actuations. The *Network Infrastructure* category comprises CQs that address the interaction between MQTT clients and brokers, including message exchange, sessions, and network connections. These questions focus on how participants exchange control packets and how communication provenance can be reconstructed and analyzed. Finally, the *Topic Subject* category includes CQs that target the semantic interpretation of MQTT topics. These questions investigate how topic strings can be decomposed into semantically meaningful components and aligned to SSN/SOSA concepts such as activities, features of interest, and observable properties.

Furthermore, we subdivided these categories into a general and use-case-specific dimension. General CQs cover the intended requirements, while use-case-specific CQs focus more on practical applicability, providing a foundation for later analysis.

 doi:10.20944/preprints202605.1509.v1

The alignment between the designed and implemented ontology with the defined CQs confirms the conceptual completeness of the model. All CQs could be mapped to specific ontology entities, indicating complete coverage of the intended requirements and scope definition.

7.3. Ontology Validation and Framework Evaluation

Validation assesses the operational applicability of the MQTT4SSN ontology by examining whether it can be instantiated with real-world streaming data and support the derivation of meaningful, context-aware insights. In contrast to verification, this step follows an application-based validation approach, in which the ontology is evaluated by integrating it into the MQTT2RDF framework. Consequently, the validation simultaneously reflects the performance and suitability of the framework that operationalizes the ontology. While verification focused on the conceptual coverage and modeling of the defined CQs, the validation evaluates whether these CQs can be answered successfully on instantiated streaming data within a dynamic data-processing environment. The validation pipeline is initiated via a command-line test setup that runs a shell script simulating MQTT clients that generate CSV-formatted messages 7.1 and publish them to a broker. This setup enables controlled experimentation with varying numbers of data volumes. Specifically, we evaluated different configurations by varying a) the number of messages (1, 10, 100) and b) the number of rows per message (1, 10, 100), resulting in a runtime analysis to evaluate the performance characteristics of the processing pipeline. We limited the evaluation to `PUBLISH` control packets due to their application content. The ingestion pipeline includes preprocessing payloads, batching, and executing semantic mappings. During preprocessing, incoming CSV data is parsed and structurally normalized into JSON. In the subsequent queuing and batching phase, messages are temporarily buffered to enable efficient batch processing. The mapping stage then applies the defined RML-based transformation rules to convert the structured data into RDF triples aligned with the MQTT4SSN ontology. The batching mechanism is configured with a maximum batch size of 20 messages and a timeout threshold of 200 ms, ensuring a balance between latency and throughput. The messages are ingested, preprocessed, and transformed into RDF representations in near real-time. This process results in a continuously growing knowledge graph that integrates both semantically enriched message-level metadata and observation data. Table 2 outlines the experimental setup and captures the execution times for preprocessing, queuing, and batching, and semantic mapping across different data configurations.

Table 2. Runtime analysis (in milliseconds) of CSV data transmitted via MQTT for a) varying numbers of messages $m \in \{1, 10, 100\}$ and b) rows per message $r \in \{1, 10, 100\}$. The maximum batch size is limited to 20, with a batch timeout of 200 ms for each runs.

Data ($m \times r$)	Preproc.	Batching	Saving	Node Total	Mapping	Upload	Shell Total	All Total
001 × 001	0,00	221,00	3,00	**224,00**	2.035,00	63,00	**2.130,00**	**2.354,00**
001 × 010	3,00	240,00	3,00	**246,00**	2.853,00	55,00	**2.962,00**	**3.208,00**
001 × 100	5,00	255,00	2,00	**262,00**	2.402,00	80,00	**2.510,00**	**2.772,00**
010 × 001	0,90	221,40	3,30	**225,60**	2.132,00	55,90	**2.212,50**	**2.438,10**
010 × 010	2,30	224,80	3,50	**230,60**	2.345,30	85,60	**2.456,00**	**2.686,60**
010 × 100	6,70	234,20	2,50	**243,40**	2.401,60	98,20	**2.522,60**	**2.766,00**
100 × 001	1,10	224,51	3,61	**229,22**	2.464,79	72,86	**2.565,65**	**2.794,87**
100 × 010	1,62	220,53	3,03	**225,18**	2.184,91	70,76	**3.301,29**	**3.526,47**
100 × 100	6,22	217,62	2,90	**226,74**	2.903,64	256,07	**4.928,84**	**5.155,58**

The evaluation was conducted on a typical notebook equipped with an Intel Core i9-14900HX processor and 64 GB of RAM. This experimental design enables a systematic assessment of how the system behaves under increasing data loads and varying message sizes. In general, the total runtime increases with larger data loads and message sizes. The results indicate that semantic lifting (mapping) and RDF persistence (tripplestore upload) account for the majority of overall processing time, whereas MQTT ingestion, preprocessing, and batching introduce comparatively low overhead. Consequently, scalability is primarily constrained by semantic transformations and triplestore interactions rather than by message transport itself. Although RDF transformation introduces measurable overhead, the overall processing time remains within a range suitable for near-real-time IoT integration scenarios.

 doi:10.20944/preprints202605.1509.v1

To validate the correctness and completeness of the instantiated knowledge graph, the full set of 46 previously defined CQs was executed as SPARQL queries within a Jupyter Notebook environment.

Overall, the validation demonstrates that the MQTT4SSN ontology can be effectively instantiated from streaming MQTT data and supports reliable semantic querying in near real-time. The combination of complete CQ coverage and a scalable ingestion pipeline indicates that the approach is well-suited for deployment in data-intensive IoT environments.

8. Conclusion and Future Work

Industrial technical systems rely on automated, data-driven processes that exchange sensing and actuation signals in real-time through edge devices, frequently over the MQTT messaging protocol. This article introduces MQTT2RDF, a semantic integration framework that provides artifacts for populating ontologies from real-time MQTT traffic. We presented the further developed MQTT4SSN ontology as the semantic core of the framework. The ontology models the semantics of the MQTT protocol and aligns them with SSN/SOSA concepts, enabling end-to-end traceability between sensing systems and message transfer. We have extended our previous proposed ontology version with a detailed solution that covers all MQTT 5.0 control packets, including their complete field contents. The semantic integration framework and ontology provide a foundation for a semantic analytics environment to explore MQTT message transfer, such as message transport provenance. We have set up an industrial production process scenario as a use case that transmits real-time sensor and actuator data with MQTT. The practical applicability of the framework and its semantic core was validated through a set of general and use-case-specific CQs. The use-case scenario achieves full coverage of the modeled requirements.

Although MQTT4SSN was explicitly designed for the MQTT messaging protocol and to address the missing transport-layer component in the otherwise well-established SOSA ontology, parts of the proposed modeling pattern are reusable beyond MQTT, which speaks to the model's generalizability. In particular, the distinction between protocol-level payload representation and application-level semantic interpretation, realized through the payload of the `PUBLISH` packet and application message classes, can serve as a reusable ODP for other messaging and streaming infrastructures. Whereas MQTT control packets and their fields are protocol-specific, the application-level message content, together with its descriptive metadata such as content type, encoding, delimiter, and encoded fields, provides an abstraction that can inspire the modeling of semantically interpretable payloads across different protocol-based communication settings. Likewise, the MQTT-specific part of MQTT4SSN can be reused as an ODP and aligned with SSN/SOSA, as well as with other ontologies for describing sensing systems and related message-based infrastructures.

Future work will explore semantic topic naming by automatically deriving MQTT topic hierarchies directly from observation and actuation semantics. This will reduce manual configuration effort and improve interoperability across deployments. Additionally, we will further develop the semantic analytics environment, addressing automatic transport provenance, to systematically analyze malformed or otherwise faulty MQTT packets. This will enable the automated detection, classification, and root-cause analysis of transport and protocol errors.

Author Contributions: Niklas Doerner: Writing – original draft, Methodology, Investigation, Formal analysis, Conceptualization. Maria Maleshkova: Writing – review & editing, Supervision, Methodology, Investigation, Formal analysis, Conceptualization. All authors have read and agreed to the published version of the manuscript.

Funding: This research, as part of the project „Engineering für die KI-basierte Automation in Produktionsumgebungen“ (EKI)[2], is funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr, which we gratefully acknowledge. dtec.bw is funded by the European Union - NextGenerationEU.

[2] https://dtecbw.de/home/forschung/hsu/projekt-eki/, accessed: May 22, 2026

 doi:10.20944/preprints202605.1509.v1

Data Availability Statement: The MQTT4SSN ontology[3][4] and the full MQTT2RDF artifacts[5][6] are available in separate GitHub repositories and archived on Zenodo with its own DOI [10,47]. The Ontology documentation[7] is available under a persistent identifier. An MQTT2RDF manual is available on GitHub[3] for describing system requirements, deployment steps, and setup instructions. All resources are licensed under Attribution-NonCommercial-ShareAlike 4.0 International[8].

Acknowledgments: During the preparation of this work, the authors used Grammarly to check grammar and spelling. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CQ	Competency Question
ETL	Extract, Transform, Load
FAIR	Findable, Accessible, Interoperable, and Reusable
GDB	Graph Database
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
M2M	machine-to-machine
ODP	Ontology Design Pattern
OOPS!	OntOlogy Pitfall Scanner!
QoS	Quality of Service
QUDT	Quantities, Units, Dimensions, and Data Types
RDB	Relational Database
SSN	Semantic Sensor Network
WOT	Web of Things

References

1. Eren, E. Edge Computing Applications in Industrial IoT: A Literature Review. In Proceedings of the Economics of Grids, Clouds, Systems, and Services (GECON 2022). Springer, 2022, Vol. 13430, *Lecture Notes in Computer Science*, pp. 153–167. https://doi.org/10.1007/978-3-031-29315-3_11.
2. Piccialli, F.; et al. Edge Intelligence for Industrial IoT: Opportunities and Limitations. *Procedia Computer Science* **2024**, *246*, 1–10. https://doi.org/10.1016/j.procs.2024.01.039.
3. Banks, A.; Briggs, E.; Borgendale, K.; Gupta, R. MQTT Version 5.0. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html, 2019. OASIS Standard. Latest version: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.
4. Mohammed, A.; et al. Performance Evaluation Framework of MQTT Client Libraries for IoT Applications. *Internet of Things* **2024**, *26*, 100000. https://doi.org/10.1016/j.iot.2024.100213.
5. Fysarakis, K.; et al. On the Security and Privacy of MQTT in Industrial Applications. In Proceedings of the Proceedings of the Information Security and Cryptography Conference (ISCC), 2017.
6. Janowicz, K.; Haller, A.; Cox, S.J.D.; Le Phuoc, D.; Lefrançois, M. SOSA: A Lightweight Ontology for Sensors, Observations, Samples, and Actuators. *Journal of Web Semantics* **2019**, *56*, 1–10. https://doi.org/10.1016/j.websem.2018.06.003.
7. Haller, A.; Janowicz, K.; Cox, S.J.D.; Le Phuoc, D.; Taylor, K.; Lefrançois, M. Semantic Sensor Network Ontology. W3c recommendation, W3C and OGC, 2017. Accessed May 22, 2026.

[3] https://github.com/doernern/MQTT4SSNOntology, accessed: May 22, 2026
[4] https://doi.org/10.5281/zenodo.16704302, accessed: May 22, 2026
[5] https://github.com/doernern/MQTT2RDF, accessed: May 22, 2026
[6] https://doi.org/10.5281/zenodo.17823700, accessed: May 22, 2026
[7] https://www.w3id.org/MQTT4SSN-Ontology , accessed: May 22, 2026
[8] https://creativecommons.org/licenses/by-nc-sa/4.0/, accessed: May 22, 2026

 doi:10.20944/preprints202605.1509.v1

8. W3C Spatial Data on the Web Interest Group. SSN/SOSA Ontology Draft. https://w3c.github.io/sdw-sosa-ssn/ssn/, 2024. Draft version.
9. Sandeep, M.; Chandavarkar, B.R.; Khatri, S. Heterogeneous Data Format Integration and Conversion (HDFIC) Using Ontology-Based Techniques in IoT. *Evolving Systems* **2024**, *15*, 375–396. https://doi.org/10.1007/s12530-024-09568-7.
10. Doerner, N.; Maleshkova, M. MQTT4SSNOntology: MQTT4SSN v1.0: CEUR Workshop Proceedings release, 2025. https://doi.org/10.5281/zenodo.16704302.
11. Markovic, M.; Corsar, D.; Asif, W.; Edwards, P.; Rajarajan, M. Towards Transparency of IoT Message Brokers. In Proceedings of the Provenance and Annotation of Data and Processes (IPAW 2018). Springer, 2018, Vol. 11017, *Lecture Notes in Computer Science*, pp. 200–203. https://doi.org/10.1007/978-3-319-98379-0_19.
12. Asif, W.; Corsar, D.; Markovic, M.; Edwards, P.; Rajarajan, M. Enhancing Transparency of MQTT Brokers for IoT Applications Through Semantic Provenance. In Proceedings of the Proceedings of the Workshop on Semantic Web for IoT, 2021.
13. Haller, A.; Janowicz, K.; Cox, S.J.D.; Le Phuoc, D.; Taylor, K.; Lefrançois, M. Semantic Sensor Network Ontology. https://www.w3.org/TR/vocab-ssn/, 2017. W3C Recommendation 19 October 2017.
14. Taylor, K.; Haller, A.; Lefrançois, M.; Cox, S.J.D.; Janowicz, K.; Garcia-Castro, R.; Le-Phuoc, D.; Lieberman, J.; Atkinson, R.; Stadler, C. The Semantic Sensor Network Ontology, Revamped. In Proceedings of the Proceedings of the Journal Track co-located with the 18th International Semantic Web Conference (ISWC 2019). CEUR-WS.org, 2019, Vol. 2576, *CEUR Workshop Proceedings*, pp. 1–9.
15. Cox, S.J.D.; Lefrançois, M.; Warren, R.; Atkinson, R.; Moreira de Sousa, L.; Schleidt, K.; Grellet, S.; Haller, A.; Janowicz, K.; Le Phuoc, D.; et al. Semantic Sensor Network Ontology – 2023 Edition. W3C First Public Working Draft WD-vocab-ssn-2023-20250916, World Wide Web Consortium, 2025. W3C First Public Working Draft, 16 September 2025.
16. QUDT.org. Quantities, Units, Dimensions and Types (QUDT) Ontology Version 1.1. https://www.linkedmodel.org/doc/qudt/1.1/, 2011. Accessed May 22, 2026.
17. QUDT.org. QUDT Ontologies Overview. https://qudt.org/pages/QUDToverviewPage.html, 2024. Overview of quantity kinds, units, dimensions, and data types in QUDT. Accessed May 22, 2026.
18. W3C Web of Things. MQTT-to-RDF Ontology and Binding Templates. https://w3c.github.io/wot-binding-templates/bindings/protocols/mqtt/ontology.html, 2024.
19. Doerner, N.; Maleshkova, M. MQTT4SSN: An Ontology for the MQTT Message Protocol. In Proceedings of the Joint Proceedings of the 16th Workshop on Ontology Design and Patterns (WOP 2025) and the 1st Workshop on Bridging Hybrid Intelligence and the Semantic Web (HAIBRIDGE 2025), co-located with the 24th International Semantic Web Conference (ISWC 2025); Novakazi, F.; Dalal, A.S., Eds., Nara, Japan, 2025; Vol. 4093, *CEUR Workshop Proceedings*, pp. 57–70. https://doi.org/10.5281/zenodo.16704302.
20. Blomqvist, E.; Gangemi, A.; Presutti, V. Experiments on pattern-based ontology design. In Proceedings of the Proceedings of the Fifth International Conference on Knowledge Capture, New York, NY, USA, 2009; K-CAP '09, p. 41–48. https://doi.org/10.1145/1597735.1597743.
21. Das, S.; Sundara, S.; Cyganiak, R. R2RML: RDB to RDF Mapping Language. W3c recommendation, World Wide Web Consortium (W3C), 2012.
22. Meester, B.D.; Heyvaert, P.; Delva, T.; Dimou, A.; Sande, M.V. RDF Mapping Language (RML). Unofficial draft, RML.io / Knowledge Graph Construction community, 2024.
23. Michel, F.; Djimenou, L.; Faron-Zucker, C.; Montagnat, J. xR2RML: Relational and Non-Relational Databases to RDF Mapping Language (Version 5). https://webusers.i3s.unice.fr/~fmichel/xr2rml_specification_v5.html, 2017. Specification v5, I3S Laboratory, Université Côte d'Azur, CNRS, Inria.
24. Canim, M.; Cornelio, C.; Farrell, R.; Fokoue, A.; Gao, K.; Gunnels, J.; Iyengar, A.; Musa, R.; Rodriguez-Muro, M.; Uceda-Sosa, R. A knowledge and reasoning toolkit for cognitive applications. In Proceedings of the Proceedings of the fifth ACM/IEEE Workshop on Hot Topics in Web Systems and Technologies, San Jose California, 2017; pp. 1–10. https://doi.org/10.1145/3132465.3132478.
25. Xiao, G.; Ding, L.; Cogrel, B.; Calvanese, D. Virtual Knowledge Graphs: An Overview of Systems and Use Cases. *Data Intelligence* **2019**, *1*, 201–223, https://doi.org/10.1162/dint_a_00011.
26. Xiao, G.; Lanti, D.; Kontchakov, R.; Komla-Ebri, S.; Güzel-Kalaycı, E.; Ding, L.; Corman, J.; Cogrel, B.; Calvanese, D.; Botoeva, E. The Virtual Knowledge Graph System Ontop. In Proceedings of the The Semantic Web – ISWC 2020; Pan, J.Z.; Tamma, V.; d'Amato, C.; Janowicz, K.; Fu, B.; Polleres, A.; Seneviratne, O.; Kagal, L., Eds., Cham, 2020; pp. 259–277.

 doi:10.20944/preprints202605.1509.v1

27. Heyvaert, P.; Dimou, A.; Chaves-Fraga, D. RML Implementation Report. Unofficial draft, RML.io / Knowledge Graph Construction community, 2022.
28. Heyvaert, P.; Van Assche, D.; De Meester, B.; Haesendonck, G.; de Vleeschauwer, E.; Oo, S.M. RMLMapper-JAVA. https://doi.org/10.5281/zenodo.3929132.
29. Ivanovic, P.; Burbach, S.; Maleshkova, M. MontoFlow – A Maritime Ontology Framework for Modeling Ship Sensory Systems. In Proceedings of the The Semantic Web – ISWC 2025; Garijo, D.; Kirrane, S.; Salatino, A.; Shimizu, C.; Acosta, M.; Nuzzolese, A.G.; Ferrada, S.; Soulard, T.; Kozaki, K.; Takeda, H.; et al., Eds., Cham, 2026; pp. 276–294.
30. Listl, F.G.; Fischer, J.; Sohr, A.; Dittler, D.; Jazdi, N.; Weyrich, M. Utilizing ISA-95 in an Industrial Knowledge Graph for Material Flow Simulation - Semantic Model Extensions and Efficient Data Integration. *Procedia CIRP* **2023**, *120*, 1558–1563. https://doi.org/10.1016/j.procir.2023.09.214.
31. Foundation, O.; Contributors. Node-RED.
32. Scrocca, M.; Grassi, M.; Carenini, A.; Calbimonte, J.P.; Anicic, D.; Celino, I. A DataOps Toolbox Enabling Continuous Semantic Integration of Devices for Edge-Cloud AI Applications, 2025. arXiv:2508.02708 [cs], https://doi.org/10.48550/arXiv.2508.02708.
33. Bender, M.; Kirdan, E.; Pahl, M.O.; Carle, G. Open-Source MQTT Evaluation. In Proceedings of the 2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 2021; pp. 1–4. https://doi.org/10.1109/CCNC49032.2021.9369499.
34. Light, R.A. Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software* **2017**, *2*, 265. https://doi.org/10.21105/joss.00265.
35. Inc., E.T. EMQX: The Unified MQTT Platform for AI & IoT Data Streaming — emqx.com. https://www.emqx.com/en. Accessed May 22, 2026.
36. Aguiar, C.Z.; Souza, V.E.S. SABiOx: the extended systematic approach for building ontologies, 2024.
37. Musen, M.A. The Protégé project: A look back and a look forward. *AI Matters* **2015**, *1*. https://doi.org/10.1145/2557001.25757003.
38. Poveda-Villalón, M.; Gómez-Pérez, A.; Suárez-Figueroa, M.C. OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation. *International Journal on Semantic Web and Information Systems (IJSWIS)* **2014**, *10*, 7–34.
39. Lohmann, S.; Link, V.; Marbach, E.; Negru, S. WebVOWL: Web-based Visualization of Ontologies. 04 2015, Vol. 8982, pp. 154–158. https://doi.org/10.1007/978-3-319-17966-7_21.
40. Garijo, D. WIDOCO: a wizard for documenting ontologies. In Proceedings of the International Semantic Web Conference. Springer, Cham, 2017, pp. 94–102. https://doi.org/10.1007/978-3-319-68204-4_9.
41. Wilkinson, M.D.; Dumontier, M.; Aalbersberg, I.J.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.W.; da Silva Santos, L.B.; Bourne, P.E.; et al. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data* **2016**, *3*. https://doi.org/10.1038/sdata.2016.18.
42. Biron, P.V.; Malhotra, A. XML Schema Part 2: Datatypes Second Edition. https://www.w3.org/TR/xmlschema-2/, 2004. W3C Recommendation, 28 October 2004.
43. Merkel, D. Docker: lightweight linux containers for consistent development and deployment. *Linux journal* **2014**, *2014*, 2.
44. GraphDB Downloads and Resources — graphdb.ontotext.com. https://graphdb.ontotext.com/. Accessed May 22, 2026.
45. Wireshark Foundation. Wireshark: Network Protocol Analyzer. https://www.wireshark.org/, 2024. Accessed: May 22, 2026.
46. Wireshark Foundation. TShark: Command-Line Network Protocol Analyzer. https://www.wireshark.org/docs/man-pages/tshark.html, 2024. Accessed: May 22, 2026.
47. Doerner, N.; Maleshkova, M. MQTT2RDF v0.1: Pre-publication release, 2025. https://doi.org/10.5281/zenodo.17823700.