Semantic-based distributed ledger technology for pervasive cyber-physical systems

Politecnico di Bari

Department of Electrical and Information Engineering

Electrical and Information Engineering Ph.D. Program

SSD: IINF-05/A – Information Processing Systems

Final dissertation

Semantic-based Distributed Ledger Technology for Pervasive Cyber-Physical Systems

by Saverio Ieva

Ph.D. program Coordinator:

Prof. Mario Carpentieri

Supervisor:

Prof. Michele Ruta

Course n. 37, 01/11/2021 – 31/10/2024

Politecnico di Bari

Department of Electrical and Information Engineering

Electrical and Information Engineering Ph.D. Program

SSD: IINF-05/A – Information Processing Systems

Final dissertation

Semantic-based Distributed Ledger Technology for Pervasive Cyber-Physical Systems

by Saverio Ieva

Ph.D. program Coordinator:
Prof. Mario Carpentieri

Supervisor:
Prof. Michele Ruta

Referees:
Prof. Massimiliano Rak
Dr. Maria di Summa

Course n. 37, 01/11/2021 – 31/10/2024

To my family

Abstract

The evolution of *Cyber-Physical Systems (CPSs)* and *Distributed Ledger Technologies (DLTs)* marks a significant step forward in building secure and intelligent infrastructures across various domains, including manufacturing, smart cities, and the Internet of Things (IoT). As CPS systems increasingly integrate computational capabilities with physical processes, the need for robust, transparent, and decentralized data management becomes essential. DLTs, particularly blockchain, provide a solution by ensuring data integrity and enabling secure peer-to-peer interactions, making it ideal for supporting the trust and information visibility requirements of CPS environments.

In this context, this dissertation presents a semantic-based framework for integrating DLTs into CPSs, leveraging the *Semantic Web of Things (SWoT)* to enhance decision-making, resource management, and automated reasoning within decentralized systems. The proposed architecture addresses key challenges in scalability, security, and interoperability by combining DLTs with semantic technologies to enable resource-constrained devices to perform complex reasoning tasks locally while preserving data integrity and trust across the system.

The framework's primary contributions include: (1) a knowledge representation layer that supports advanced resource discovery and service selection through automated reasoning; (2) a federated learning model that enables secure, decentralized model training across edge and IoT devices, preserving data privacy and optimizing network efficiency; and (3) a semantic-enhanced blockchain mechanism that utilizes smart contracts and semantic matchmaking to dynamically manage and prioritize resources across the continuum. Extensive experimental validation, including a case study on green mobility for electric vehicles, demonstrates the framework's applicability to real-world CPS scenarios, showcasing improvements in data handling, scalability, and resource utilization.

Contents

Introduction

The evolution of *CPSs* marks a transformative progresse in the integration of computational intelligence with physical processes, laying a foundation for modern industrial systems and IoT-enabled environments. Defined by their close coupling of computational elements with dynamic real-world systems, CPSs plays a crucial role in applications such as *Industry 4.0* and smart manufacturing. These systems embed devices into physical processes, enabling real-time monitoring and control via feedback loops that tightly link physical activities with computations. As CPSs proliferate, driven by advancements in networked devices and embedded systems, challenges emerge concerning scalability, security, and resource management.

DLTs, particularly blockchain, can help address these challenges by introducing a secure, decentralized framework for data management, essential for building trust within heterogeneous CPS infrastructures. Blockchain's capacity for data immutability and peer-to-peer interactions provides a reliable solution for decentralized control, enhancing security by eliminating central points of vulnerability. This property is especially relevant in environments where trust in cooperation and data transactions is critical, such as industrial IoT networks and smart energy grids.

Alongside DLTs, the *SWoT* framework merges IoT capabilities with the Semantic Web, fostering automated reasoning and intelligent decision-making. In SWoT, data from interconnected devices is semantically enriched, allowing devices to autonomously interpret and act on data they exchange. Integrating semantic technologies with DLT not only improves data transparency but also facilitates context-aware decision-making, empowering CPS components to act more autonomously and intelligently.

This dissertation proposes a comprehensive framework combining semantic-based DLTs with CPSs to tackle issues of trust, scalability, and resource efficiency in decentralized environments. The research offers the following key contributions:

- **Semantic Knowledge Representation and Reasoning:** This dissertation introduces a semantic layer for enhanced knowledge representation and automated reasoning within CPS [115], structured across three interrelated planes: Network Plane, Processing Plane, and Information Plane. This layered approach enables comprehensive and scalable resource management, service discovery, and decision-making capabilities in distributed environments.

  In the **Network Plane** [54], the framework leverages Software-Defined Networking (SDN) principles to create a flexible network control layer that integrates Network Function Virtualization (NFV). The network plane decouples the control and data planes, allowing centralized network policy management and real-time adaptation to application requirements. Through SDN and Knowledge-Defined Networking (KDN), [87] a knowledge-driven model is established, where data insights enable continuous refinement and optimization of network operations. This approach provides the dynamic topology adjustments and configurability necessary for scaling resource allocation effectively within CPS.

  The **Processing Plane** focuses on the architecture and computational processes required for automated reasoning. Here, the *Tiny-ME* (Tiny Matchmaking Engine) [115] reasoner provides standard and non-standard inference services on Web Ontology Language (OWL) 2 ontologies, supporting both standard tasks like ontology coherence checks and non-standard tasks such as concept abduction and contraction. Additionally, *Tiny-ME* integrates *Cowl*, a new C parser for OWL 2 Full, released independently under a permissive open-source license to promote reuse [12]. This plane provides essential reasoning capabilities that support service orchestration and autonomous decision-making.

The **Information Plane** [106] is responsible for data transformation and knowledge base construction. By converting raw data into semantically rich Knowledge Bases (KBs) and Knowledge Graphs (KGs), the Information Plane bridges the conceptual gap between data sources and Semantic Web languages, allowing autonomous knowledge discovery and contextual adaptation on pervasive devices. This automated framework supports real-time, context-aware decision-making across distributed CPS.

- **Federated Learning Framework for the Cloud-to-Thing Continuum:** A novel Federated Learning (FL) framework is defined to address the challenges of data privacy and scalability by supporting decentralized model training across edge and IoT devices [78] [79]. This approach utilizes a semantic-based algorithm to aggregate local training data into a global model, preserving privacy while ensuring data efficiency. By enabling both local and cloud-based model training and inference, this FL setup enhances CPS capabilities in scenarios demanding real-time responses.

- **Semantic-Enhanced Blockchain for Distributed Resource Management:** A semantic-enhanced blockchain architecture, named *SeeSaw*, is proposed to handle secure, intelligent resource management in CPSs [118]. Building on *Hyperledger Sawtooth*, SeeSaw integrates semantic annotations into transactions, enabling advanced resource matching and selection through smart contracts. Semantic logic-based inferences embedded within the blockchain allow for dynamic, adaptive interactions, improving both transparency and interoperability.

Extensive experimental campaigns have been conducted to evaluate the proposed approaches, providing quantitative data on performance, efficiency, and applicability across different reference scenarios. In the context of smart manufacturing, the proposed federated architecture has been implemented for monitoring and predictive maintenance, where IoT devices collect and analyze data in real-time through edge nodes. This decentralized resource management setup has demonstrated the effectiveness of edge computing in

supporting operational decisions in dynamic production environments. For health monitoring in assisted settings, the federated learning approach has been applied to allow local processing of collected data, ensuring privacy protection and compliance with regulatory requirements while maintaining the efficiency needed for timely identification of anomalies or critical events. In industrial monitoring, reasoning tasks have been used to autonomously detect risk situations in real-time directly at the edge nodes, demonstrating the system's ability to identify potential hazards, such as fires or explosions, and generate timely alerts.

A specific case study focused on electric mobility was used to validate the SeeSaw framework in a sustainable mobility context. In this scenario, SeeSaw managed resources for Plug-in Electric Vehicles (PEVs) charging in a decentralized manner, leveraging semantic annotations integrated into the blockchain to support optimal selection of available energy resources and transparent management of charging transactions. Experimental results showed that SeeSaw improves the efficiency of distributed resource management and increases trust in transactions between network nodes, demonstrating the practical relevance of the framework in green mobility contexts.

The remainder of this thesis is structured as follows: Chapter 2 provides foundational concepts on CPSs, DLTs, and SWoT, addressing their integration and related challenges. Chapter 3 details the knowledge representation and reasoning methods used to optimize resource orchestration and service discovery in CPSs. Chapter 4 explores the federated learning framework, demonstrating its application to edge-based CPSs. Before conclusions and insights for future research, Chapter 5 focuses on the SeeSaw blockchain framework, along with experimental results.

Chapter 2

Towards secure and intelligent pervasive Cyber-Physical Systems

CPSs represent a innovative approach to integrating computational capabilities with physical processes. This integration aims to enhance system performance, reliability, and efficiency, especially within the manufacturing industry. In particular, CPSs are advanced integrations of computation, networking, and physical processes. In CPSs, embedded devices and networks monitor and control physical processes with feedback loops where physical processes affect computations and vice versa. This integration is crucial in modern manufacturing, known as Industry 4.0, where smart factories utilize CPS technologies to achieve higher levels of efficiency, reliability, and autonomy.

The *Semantic Web* [10], envisioned as the next stage in the World Wide Web (WWW)'s development, is a joint effort directed by the *World Wide Web Consortium (W3C)*. Its goal is to enhance the Web into a global medium to exchange data, information, and knowledge, allowing machines to comprehend the semantics, *i.e.*, the meaning, of the Web's information. This would thus create a network of machine-understandable *Linked Data* [44], making it valuable and practical for automated agents, services and applications.

Knowledge Representation (KR) and *Automated Reasoning* [7] underlie the fulfillment of the Semantic Web vision. KR aims to build a structured model of information where data is not just stored but also interconnected

with other related data, defining their context and mutual relationships. Resource Description Framework (RDF) [121] and OWL are fundamental in this context, as they enable the explicit representation of data semantics, which allows machines to comprehend and reason about the connections between various pieces of information. Automated Reasoning involves computers understanding information and making logical inferences from it. This capability is essential for the Semantic Web, allowing machines to execute complex functions like intelligent searches, data merging, and decision-making based on data semantics. Through automated reasoning, computing agents can deduce new information from existing data, detect inconsistencies, and offer more precise answers to intricate queries.

The SWoT [126] merges the Semantic Web's aim of fostering a more significant and practical Web with the IoT, which links everyday gadgets to the Internet. The core principle is to utilize semantic technologies in smart object networks, thereby enhancing data integration, interpretation, and usability among various interconnected devices.

In the scope of the SWoT, ontology-driven annotations define and harmonize the descriptions of devices, objects, and events. This semantic modeling creates an interoperability framework for intelligent agents to comprehend and interact with their surroundings. Through automated reasoning mechanisms, agents can infer implicit knowledge from the clear descriptions provided, allowing them to function autonomously towards their goals without ongoing human oversight or intervention. This feature of SWoT supports the creation of more responsive, adaptable, and intelligent IoT systems.

A key difference between the SWoT and the traditional Semantic Web is observed in the type and scale of the KBs they use and the queries they process. In the traditional Semantic Web, the emphasis is usually on sophisticated queries spanning extensive KBs with moderate to high expressiveness. These queries are generally executed as resource-intensive batch jobs, requiring considerable processing power and time. In contrast, SWoT is characterized by the need to support various specialized use cases, usually involving smaller KBs with low-to.moderate expressiveness. Inferences in

SWoT are also distinctly different, focusing on on-the-fly queries to adapt to rapidly changing data and context conditions to deliver immediate insights or take autonomous decisions.

This paradigm shift is crucial to making the SWoT both practical and efficient in real-world scenarios. By emphasizing smaller, more agile KBs and swift queries, the SWoT can be seamlessly integrated into various environments, ranging from smart homes and industrial contexts to smart cities and beyond. This strategy facilitates a more dynamic interaction between the semantic layer and the physical realm, enabling IoT devices not only to gather and exchange data but also to comprehend and act upon it in significant ways.

2.1 Cyber-Physical Systems

CPSs represent a innovative approach to integrating computational capabilities with physical processes. This integration aims to enhance system performance, reliability, and efficiency, especially within the manufacturing industry. In particular, CPSs are advanced integrations of computation, networking, and physical processes. In a CPS, embedded devices and networks monitor and control physical processes with feedback loops where physical processes affect computations and vice versa. This integration is crucial in modern smart industry, where factories exploit CPSs to achieve higher levels of efficiency, reliability, and autonomy.

The subsequent sections offer a comprehensive summary of the CPS core framework, describing the various interconnected layers and components that support the fusion of computational and physical aspects. Moreover, it discusses how Edge Intelligence (EI) improves CPS by facilitating data processing and decision-making at the network's edge, thereby lowering latency and enhancing efficiency in real-time scenarios.

2.1.1 Reference architecture of CPSs

The 5C architecture, described in [65], provides a structured model for the deployment and enhancement of CPSs within industrial settings. This architecture supports the automation of complex processes and plays a critical role in Industry 4.0 [21] and Industry 5.0 [67]. The framework consists of five distinct levels, illustrated in Figure 2.1:

- **Connection**: This layer focuses on data acquisition from sensors embedded in physical systems, forming the foundational input for subsequent levels. IoT-enabled devices in modern infrastructures enhance data collection across distributed nodes, enabling detailed monitoring and control of CPS components.
- **Conversion**: Data collected from the Connection layer undergoes processing to produce actionable insights. Techniques such as data filtration, aggregation, and basic analytics refine this information, making it suitable for further analysis at the Cyber level.
- **Cyber**: Advanced analytics, Machine Learning (ML), and Artificial Intelligence (AI) algorithms at this layer interpret data and forecast trends, empowering the CPS to support real-time fault detection, optimization, and decision-making in industrial applications.
- **Cognition**: The CPS adapts to changing conditions by analyzing refined data. Cognitive processes like reasoning and learning enhance system responsiveness to real-time demands, allowing for autonomous adaptation to variable conditions.
- **Configuration**: This level enables CPSs to autonomously adjust operations, allocate resources, and implement corrective measures, ensuring resilience and optimal performance.

Building upon the 5C model, Digital Twin (DT) technology has emerged as a powerful paradigm for optimizing CPSs. Initially used in aerospace and

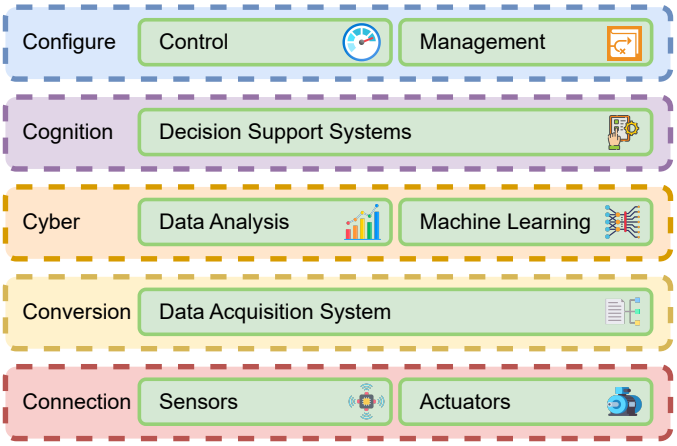


Figure 2.1: The 5C architecture of Cyber-Physical Systems.

manufacturing, DTs now serve as dynamic virtual representations of physical objects, processes, or systems across various industries. By using data from sensors and other sources, DTs offer real-time insights into asset performance and operational conditions, facilitating predictive maintenance and informed decision-making.

DT architectures typically integrate geospatial and simulation models, advanced analytics, and connectivity. The geospatial model provides a 3D visual representation of physical entities, while simulation models create a detailed representation of behaviors under different conditions. This integration of real-time data flow between physical and digital domains ensures high-fidelity reflection of the operational state, supporting lifecycle management and collaborative tasks across stakeholders. Furthermore, the exploitation of KGs within DTs, combined with Large Language Models (LLMs) by means of Retrieval-Augmented Generation (RAG) frameworks, enhances decision support by organizing and processing complex domain-specific information to improve real-time, accurate information retrieval. [52]

Incorporating DTs within CPSs architectures enables a collaborative environment for engineers, operators, and maintenance personnel, enhancing system transparency and usability. This DT-driven approach supports CPSs in achieving greater autonomy, reliability, and operational efficiency, ensuring that infrastructure systems remain adaptable and resilient in dynamic

environments.

2.1.2 Edge intelligence and integration of Digital Twins in CPSs

The adoption of AI in Industry 4.0 has significantly enhanced the capabilities of CPSs. Rapid advancements in AI enable simulations that emulate precise behaviors. These advancements are crucial for CPSs, where *Deep Learning* [40] and other ML models leverage large datasets for training, refining predictive capabilities, and enhancing data interpretation accuracy.

Initially, AI-based systems in IoT environments relied on centralized cloud infrastructures for processing data. However, the rise of robust edge devices has catalyzed a shift toward *Edge Computing* [1]. This approach decentralizes computing tasks, allowing critical computations closer to data sources, which reduces latency, preserves bandwidth, and enhances system responsiveness. Localized processing in edge environments supports applications requiring immediate action and enables quick and reliable responses.

Aligned with this vision, the concept of *EI* [31] integrates AI with Edge Computing, allowing the deployment of AI models on edge devices for real-time data analysis. While resource-heavy model training is handled by cloud data centers, inference tasks are executed locally, enabling CPSs to process data streams in real-time, facilitating localized decision-making. This setup is particularly effective in contexts where EI autonomously manages distributed resources, optimizes operations, and responds to real-time dynamics, enhancing overall system reliability.

A pivotal component of EI within CPSs is the potential use of *DTs*, which create virtual replicas of physical assets or processes to support continuous monitoring, simulation, and real-time control [129]. DTs provide a detailed, virtual representation of system components and operational states, facilitating predictive maintenance and operational optimization. By synchronizing real-world data with digital models, DTs allow operators to analyze

performance across diverse scenarios and implement optimized solutions to emerging challenges.

Further exploring the possibilities, integrating DTs with KGs by means of RAG could enhance decision support and improve human-machine interaction [99]. In this configuration, KGs are used to organize complex, domain-specific knowledge, allowing the system to provide contextually relevant responses by structuring data and relationships. The RAG-based approach enables LLMs within the system to retrieve relevant information from the KG, enriching responses and enabling a conversational virtual assistant to offer detailed, context-aware guidance for operational decisions. This concept is being considered as a potential enhancement in CPSs applications, allowing for even more informed and accessible decision-making.

The flexibility of EI within CPSs is further enhanced by advanced paradigms like *Osmotic Computing (OC)*, which dynamically orchestrates resources across cloud and edge layers [81]. In OC, computational resources are allocated based on workload demands, network conditions, and device availability, allowing CPSs to adapt to fluctuating computational needs. This adaptability is especially valuable in dynamic environments where data flow and processing demands change rapidly. By balancing loads across the cloud-to-edge continuum, OC supports high performance and scalability.

The use of *serverless computing* in CPSs introduces an event-driven model that triggers functions only when necessary. This model can provide higher scalability, cost-efficiency, and responsiveness, allowing CPSs to allocate resources dynamically with minimal infrastructure management. This is particularly beneficial for real-time tasks like anomaly detection and automated resource allocation, where adaptability and efficiency are crucial for maintaining system resilience [79].

Together, EI, DTs, OC, and serverless computing form a comprehensive framework for developing highly distributed CPSs [52]. Exploring the integration of advanced intelligence with responsive Digital Twins, adaptive orchestration, and augmented information retrieval can position next-generation

CPSs to respond in a flexible and resilient way to complex, real-time demands in modern industrial and urban ecosystems, ensuring robustness, efficiency, and scalability across diverse applications.

2.2 Distributed Ledger Technologies

DLTs offer an approach to enhancing trust, security, and transparency within distributed systems, particularly in applications requiring reliable data handling and peer-to-peer interactions. A DLT provides a decentralized data structure and consensus mechanisms, ensuring data immutability and reducing reliance on centralized control. These features are increasingly important in modern CPSs, where trust, scalability, secure data exchange and traceability are key to achieving high-performance, dependable autonomous systems.

The following sections outline the foundational aspects of DLTs, covering structures, consensus protocols, and unique properties, followed by their applications within CPSs.

2.2.1 DLT and blockchain technology

Blockchain refers to a class of data structures and protocols for *trustless* distributed transactional systems. In conventional distributed databases, a trusted intermediary is necessary to ensure irreversibility (i.e., no committed transaction can be reversed or changed) and to prevent censorship (*i.e.*, all valid transactions are committed). Blockchain systems eliminate intermediaries by validating transactions through a distributed *consensus* protocol, ensuring that no single node –or small group of colluding nodes (as a maximum ratio of all participating nodes, the exact value depending on the specific consensus mechanism)– can enforce the addition, removal, or modification of data. Transactions approved within a given timeframe –once more, the duration of this period depends on the specific blockchain system– are collected into *blocks*.

A block fundamentally has two components, as illustrated in Figure 2.2: a *body* and a *header*. A group of transactions and a transaction counter with a value indicating how many transactions exist within the block compose the body. The header includes the following fields:

- The *block hash* is derived from the aggregate data contained within the other header fields;
- the *previous block hash*, consisting of the digest of the preceding block's header;
- the *version*, representing the most recent iteration of the block validation specifications that may be used to verify the block's validity;
- the *timestamp* recording the block's creation time;
- the *Merkle root* encoding the root hash of the *Merkle tree*, as outlined later;
- the *nonce* discovered during the consensus process, adhering to the designated *difficulty target*.

An immutable and reliable DLT can be constructed using the concept of the *Merkle tree*. It is evident from Figure 2.2 that the base of the tree consists of a collection of transactions. The hash digest of each transaction or operation is located at the lowest level (*i.e.*, the leaves of the tree), while each upper layer is created by calculating the hash value of the pairs in the lower level. Ultimately, the root of the Merkle tree has a single hash value that represents the aggregate of all verified transactions. It is important to note that an alteration in one transaction results in a different Merkle root value due to the change at the base level of the Merkle tree.

Leveraging earlier theoretical findings on *Proof-of-Work (PoW)* consensus mechanisms [144], blockchain technology emerged with *Bitcoin*, an open-source platform for digital currency. Bitcoin employs blockchain as a *ledger* to record currency transfer transactions. Following Bitcoin's success, several

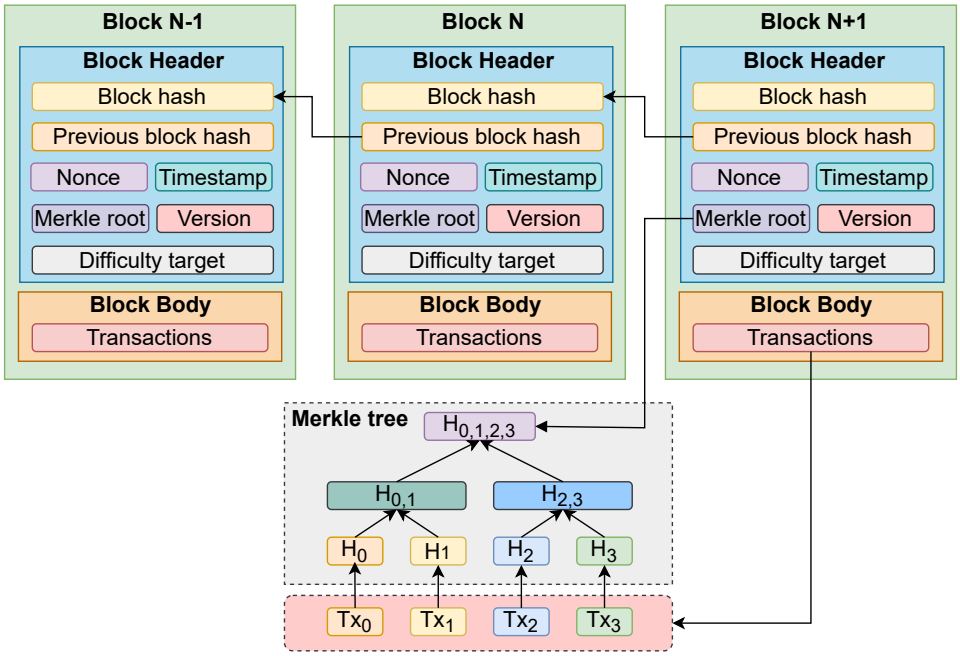


Figure 2.2: Blockchain structure

other blockchain-based electronic currency platforms have been developed. Concurrently, it was recognized that the underlying blockchain technology is essentially a general-purpose distributed database that facilitates trustless collaboration of *Decentralized Autonomous Organizations (DAOs)*. This capability enables the practical realization of the *Smart Contract (SC)* concept [133], *i.e.*,, programs that encode and enforce cooperative processes among multiple parties. Initially, SCs necessitated a trusted mediator, limiting the extensive development of this approach. However, consensus regarding SCs in a blockchain is achieved through parallel replicated execution across the network, effectively transforming every SC-enabled blockchain into a general-purpose application platform based on a distributed Virtual Machine (VM). Numerous proposals have been introduced, including proprietary platforms (with *Ethereum*[1] being perhaps the most well-known) and standardization initiatives such as the *Hyperledger*[2] project stewarded by the Linux Foundation. SC-based blockchains are adopted in various financial

[1]Ethereum Project: `https://www.ethereum.org/`
[2]Hyperledger: `https://www.hyperledger.org/`

and industrial sectors, including IoT among the emerging application areas, as discussed in Section 2.2.2.

Various kinds of blockchain systems are available, determined by the following fundamental design choices:
– **Participation requirements** A blockchain network is called *permissionless* if any node can join – even anonymously – at any time, or *permissioned* if a whitelist of accepted nodes exists and nodes are uniquely identified. This choice significantly influences the blockchain design: permissionless blockchains usually need to provide rewards for the computational efforts of participants, *e.g.*, Bitcoin allows nodes to generate (*mine*) and retain new currency for transaction block validation. In contrast, permissioned blockchains are used in more controlled cooperative environments, where access itself serves as a reward, since it allows the exchange of services or resources.
– **Transaction model.** Blockchain systems facilitate the registration and exchange of *assets*. Each node typically owns a certain quantity of assets at any given time. In the *Unspent Transaction Output (UTXO)* model, a transfer from A to B is represented as *consuming* (*i.e.*, deleting) records for A's spent assets and *creating* (*i.e.*, adding) new ones for B's received assets. Conversely, in the *account-based* model, every node has an account documenting all its assets, updated through transactions. The UTXO model, analogous to a bank statement of account, allows for simpler reconstruction of the current state from a transaction log and is commonly used in e-currency systems. While the account-based model is more versatile, it may slow down transaction processing; however, it is the only feasible option for general-purpose SC-based blockchains.
– **Consensus mechanism.** Permissionless systems necessitate more strict consensus methods, such as Proof-of-Work, to ensure data security unless a substantial fraction of nodes collaborates to compromise the blockchain. Permissioned systems – where each node is identifiable and accountable – can relax consensus requirements to lower computational loads by opting for simpler algorithms; variants of Byzantine Fault Tolerance (BFT) [144] are frequently employed.
– **Smart contract language.** Blockchains can utilize any formalism for

SC definition and execution, such as procedural (imperative) languages or logical (declarative) languages or automata [49]. Industry applications generally adopt computationally complete programming languages, whether pre-existing or specifically created for this purpose (*e.g.*, Ethereum's *Solidity*).

2.2.2 Distributed Ledgers for CPS

Centralized models for managing information, services, and devices are clearly insufficiently scalable for the continually expanding IoT. Alongside cost and performance concerns, they also bring about security and trust issues. According to [147], blockchain technologies enable transparent and trustless peer-to-peer models that are presented as a feasible solution to support the ongoing and future growth of reliable IoT networks and applications. Emerging distributed file systems, billing services, and other blockchain-based tools can be utilized as a machine-to-machine middleware layer, agnostic to applications, for operating IoT resource/service marketplaces with little to no human oversight [26]. Industrial research is exploring various scenarios within Industry 4.0 and Industry 5.0, considered as a significant IoT-based use case for blockchain [36]. Asset tracking and supply chain applications are among the most popular uses, due to the well-known advantages of trustless DAO collaboration [61, 86, 88] and the seamless integration of blockchain solutions into existing industry standards for distributed infrastructures for information sharing, particularly Electronic Product Code Information Services (EPCIS) [58]. The most straightforward methods utilize transactional ledgers for asset transfer, providing high throughput and low costs [26]. Blockchain networks built on SCs offer more adaptable systems, allowing any application logic to be implemented and embedded within the blockchain [26], as well as supporting discoverable, composable, and verifiable multi-step business processes in multi-party Service-Oriented Architectures (SOA) [96]. In this context, the Reference Architecture Model Industry (RAMI) 4.0 [50] specification outlines a SOA for facilitating cross-organizational interoperability and cooperation along the full lifecycle of objects and processes in industrial cyber-physical systems. It can exploit existing results on service discovery

and composition, like those demonstrated by IoT case studies in [11].

There are multiple options to leverage logic-based technologies within blockchains. In [36], a prototype ontology was introduced to annotate transactions with Linked Data [44], which facilitates the exploration of blockchain content for humans via semantic-enabled user agents. The design of ontology-based smart contracts for a prototype blockchain system presented in [59] provided traceability in supply chains. Logical languages can also serve as design principles, as well as tools for the formal specification and execution of smart contracts. Multiple logical frameworks have been utilized for the specification and execution of SC. The study in [49] employed *defeasible reasoning*, a recognized method for formalizing legal regulations and contracts. Conversely, [48] supported the adoption of *Linear Temporal Logic (LTL)*, which is incorporated in various model checking systems. This enables formal verification to ensure that a SC's behavior meets specified conditions. The requirement for formal verification of SCs in business-centric blockchains was also emphasized in [96].

The transparent trustless peer-to-peer models enabled by blockchain technologies are emerging as a viable path for the current and future expansion of IoT. A trend is emerging toward a multi-purpose, blockchain-oriented, machine-to-machine middleware layer for running IoT service marketplaces with minimal or no human intervention [26]. Several blockchain-based proposals, with various levels of maturity, already exist in industrial scenarios such as smart manufacturing, information-intensive marketplaces and smart grid, as well as in smart home and smart city applications [100, 111]. Smart mobility with 5G vehicular networks is one of the most investigated use cases, for both data and energy exchange [146, 73, 57]. All the above works pinpoint information interoperability as a largely open problem, but none of them take in consideration semantic-based structured representations to address the issue.

Furthermore, blockchain has been proposed for trustless secure management of computational resources in Mobile Edge Computing (MEC) architectures [148]. In fact, a further open issue is performance: PoW and SCs still

have a significant impact in terms of transaction throughput [26]. For this reason, one of the explored approaches relies on offloading PoW computation to nearby edge computing nodes, such as in the MEC architecture proposed in [76].

Nevertheless, the wider adoption of traditional blockchain has highlighted some disadvantages of this technology, such as low-throughput, scalability and high transaction fees. This hinders further expansion of service marketplaces in IoT environments, where assets are exchanged through micro-transactions. In fact, a rise in transaction volume during a specific period adversely impacts throughput, as the platform does not have enough time to handle all transactions efficiently. Furthermore, transaction fees can affect the actual value of exchanged assets or services, making micro-payments impractical in an environment where IoT-generated data are to be purchased for small amounts of money. As a result, research is investigating Directed Acyclic Graph (DAG)-based DLTs. Recent examples of the adoption of such infrastructures can also be found in smart grid [108] and smart city [154], proposing infrastructures where a DAG DLT middleware layer is used for the exchange of data generated by individuals through their personal devices. In data marketplaces, IoT devices have the role of producers of data that can be consumed by other parties interacting through a DLT layer. Applications in [108, 62] are based on the IOTA[3] framework and related protocols, such as *IOTA Wallet* for the exchange of tokens. In this scenario, proposed approaches also investigate the integration of further layers on top of the DLT infrastructure for storing personal data off-chain using distributed file systems such as the InterPlanetary File System (IPFS)[4], in order to reduce the required amount of on-chain storage and to enable authentication, access control and deletion of data on demand [153]. In [91], a case study focused on energy trading is used to compare conventional blockchain, DAG-based DLT, and *Holochain*[5]. The research suggests that, while these technologies can potentially resolve scalability challenges, they can also result in increased

[3]IOTA Framework: `https://www.iota.org/`
[4](IPFS: `https://ipfs.tech/`
[5]Holochain Project: `https://www.holochain.org/`

susceptibility to attacks and decreased decentralization in some scenarios. In fact, DAG DLT infrastructures are inadequate in situations that require stronger data immutability guarantees and trustless decentralization, which can take precedence over scalability and performance. This limitation hinders the provision of services that rely on the cooperation of various actors in a peer-to-peer network and secure data structures. Therefore, to implement SOA primitives on a traditional blockchain platform, it is essential to address performance and scalability issues. For example, a significant concern in IoT is the computational resources required for smart contracts, as their resource consumption can be highly unpredictable due to the potential for recursive calls to other SCs. Hence, it is crucial to utilize highly efficient inference tasks and diverse discovery methods to better adapt processing to varying computational demands.

2.3 The Semantic Web of Things

The SWoT framework brings together the capabilities of the Semantic Web and the IoT, enabling intelligent, context-aware interactions among devices. By integrating semantic technologies with connected devices, SWoT allows for enriched data representation and automated reasoning, providing machines with the capacity to understand and process the meaning of exchanged information. This paradigm is particularly valuable for CPSs, where context-aware decision-making enhances operational efficiency and autonomy.

The following sections introduce key SWoT aspects, including the fundamentals of knowledge representation and automated reasoning. These components support intelligent resource discovery, data integration, and real-time response, promoting an ecosystem where CPSs can autonomously adapt to evolving conditions and perform complex tasks with minimal human intervention.

2.3.1 Knowledge representation in the Semantic Web

An *ontology* is a structured model that is machine-readable, representing a domain and including its primary concepts, relationships, and constraints. It provides a standard vocabulary for users and agents, aiding in reasoning about the domain's knowledge [131]. The OWL [102] W3C standard ontology language designed for the Semantic Web is founded on Description Logics (DLs) [7], which are decidable fragments of First Order Logic (FOL) [18, 34], allowing for the formal representation of knowledge via:

- *concepts*, representing sets of objects within the domain;
- *individuals*, which are instances of concepts, i.e., real objects within the domain;
- *roles*, defining relationships between pairs of individuals.

These components can be merged through *constructors* to form DL expressions, with their formal semantics defined by means of an *interpretation* $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ associating each term to a subset of the universe of discourse (the *domain* Δ) by means of an *interpretation function* $\cdot^{\mathcal{I}}$. Concept *conjunction* is interpreted as set intersection: $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$. Concept *disjunction* is interpreted as set union: $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$. The connector $\neg$, is present, it is interpreted as the *complement*. Various other constructs are available, each defining the expressiveness and computational complexity of inference tasks in different languages of the DL family.

In the context of DLs, an ontology (a.k.a. terminology, terminological box, TBox) [131] is composed of two types of *assertions* involving concepts and roles: *inclusion*, which allows the definition of *is-a* relationships between concepts ($A \sqsubseteq D$ where A and D are concept expressions); *equivalence*, which allows naming expressions, or specifying that two expressions represent the same set of instances ($A \equiv D$). Individuals, and relationships between them, make up the so-called *assertion box* (ABox). A Knowledge Base (KB) consists of a $\langle$TBox, ABox$\rangle$ pair. With respect to OWL, the term "ontology"

is often used informally to refer to a KB as whole, rather than just its terminological part.

The W3C issued formal Recommendations for two versions of the OWL in 2009 and late 2012. OWL 2 enhanced the foundation of its predecessor by increasing the expressiveness of the language and expanding support for datatypes and annotations. Entities are fundamental components of OWL 2 ontologies, forming the basis for defining the terminology of an ontology through named terms. Each entity is uniquely identified by an Internationalized Resource Identifier (IRI) [35], which extends the concept Universal Resource Identifier (URI) [9] by allowing a broader range of characters. Entities fall into the following categories:

- **Classes**: sets of individuals, corresponding to a DL concept. For instance, the class *Sensor* might include individuals like *TempSensor01* and *HumiditySensor02*. Classes can have hierarchies, where one class is a subclass of another. For example, *Sensor* can be a subclass of *Device*, meaning every sensor is also a device.
- **Individuals**: specific instances or objects that belong to classes. For instance, *Room101* can be an individual of the class *Location*.
- **Object Properties**: relationships between individuals, corresponding to DL roles. For instance, the property *locatedIn* can relate the individual *TempSensor01* to the individual *Room101*, indicating that TempSensor01 is located in Room101.
- **Datatype Properties**: assign data values to individuals, corresponding to DL functional roles on concrete domains. For instance, a *temperatureValue* property might assign the value *23.5* to the individual *TempSensor01*.
- **Annotation Properties**: encode information about parts of the ontology itself, rather than the domain of interest. For example, they might provide metadata about when an axiom was added or who the author of a particular part of the ontology is.

Additionally, OWL allows defining *anonymous individuals*, which are useful for representing information about something without specifying a unique identifier. For instance, if one wants to state that there exists a sensor that is monitoring the temperature in a room but does not want to specify which sensor it is, they might use an anonymous individual to represent that sensor. Just as in DLs, OWL entities can be combined in more complex *expressions* by using logical *constructors*, characterizing sets of individuals by precisely outlining criteria related to their properties. Individuals that fulfill these criteria are deemed instances of the corresponding class expressions. For instance, atomic classes like *TemperatureSensor* and *InternetConnectedDevice* can be conjunctively combined to describe the class of Internet-connected temperature sensors.

Every OWL 2 ontology is a collection of *axioms*, basic statements that assert what is true in the domain of interest by combining entities and expressions. OWL allows for a variety of axiom types, allowing the composition of entities and expressions into many types of logical assertions.

The variety of logical constructors supported by OWL makes it highly expressive, allowing the specification of intricate domain knowledge, but also posing challenges for the computability and implementation of inference tasks. To address this, OWL 2 introduced several *profiles*, streamlined versions of the language tailored to meet specific industrially relevant use cases, such as enhanced reasoning scalability and efficient query answering:

- *OWL 2 EL* is optimized for ontologies with large numbers of properties or classes. It is particularly suited for fields like life sciences, due to its support for extensive hierarchies.
- *OWL 2 QL* is designed for applications requiring efficient access to large amounts of instance data. It facilitates easy integration with relational databases, making it ideal for applications where ontology-based data access is critical.
- *OWL 2 RL* is targeted at applications that require scalable reasoning without sacrificing too much expressiveness. It is particularly useful

in scenarios where reasoning can be implemented using rule-based systems.

Each profile achieves a balance between expressiveness and efficiency, allowing practitioners to select the most suitable subset for their specific needs, thereby ensuring that OWL remains adaptable and useful across various domains. Importantly, knowledge-based systems are not obligated to strictly adhere to any particular OWL profile: the supported OWL constructs can be selectively restricted to align with DLs known for their favorable computational complexity and practical performance, while still being expressive enough for meaningful applications. One such DL is the *Attributive Language with unqualified Number restrictions* ($\mathcal{ALN}$), which offers moderate expressiveness while maintaining polynomial space and time complexity for inferences [32]. Given that $\mathcal{ALN}$ will be the primary reference for expressiveness throughout most of this dissertation, its constructors are detailed below, and Table 2.1 provides an overview of the syntax and semantics of its constructors and assertions.

- $\top$, *universal concept*. All the objects in the domain.
- $\bot$, *bottom concept*. The empty set.
- A, *atomic concepts*. All the objects belonging to the set A.
- $\neg A$, *atomic negation*. All the objects not belonging to the set A.
- $C \sqsubseteq D$, *intersection*. The objects belonging to both C and D.
- $\forall R.C$, *universal restriction*. The objects x such that if x is related to y by the relation R, then y belongs to the set C.
- $\exists R$, *unqualified existential restriction*. The objects that are related to at least one object by the relation R.
- $\geq nR$, $\leq nR$, $= nR$, *unqualified number restrictions*[6]. The objects that are related to at least, at most, or exactly n objects by the relation R.

[6]It is useful to notice that $\exists R$ is equivalent to $\geq 1R$ and that $= nR$ is a shortcut for $\geq nR \sqcap \leq nR$.

Table 2.1: Syntax and semantics of $\mathcal{ALN}$

Name	Syntax	Semantics
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Atomic negation	$\neg A$	$\Delta^{\mathcal{I}} \backslash A^{\mathcal{I}}$
Universal quantification	$\forall R.C$	$\{d_1 \mid \forall d_2 : (d_1, d_2) \in R^{\mathcal{I}} \rightarrow \ d_2 \in C^{\mathcal{I}}\}$
Number restrictions	$\geq nR$	$\{d_1 \mid \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \geq n\}$
	$\leq nR$	$\{d_1 \mid \sharp\{d_2 \mid (d_1, d_2) \in R^{\mathcal{I}}\} \leq n\}$
Inclusion	$A \sqsubseteq D$	$A^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Equivalence	$A \equiv D$	$A^{\mathcal{I}} = D^{\mathcal{I}}$

In practical applications, specific syntax is necessary to store and exchange ontology documents between different tools and platforms. Ontologies can be serialized in any RDF-based triple syntax, such as *RDF/XML* [121] (RDF embedded in *eXtensible Markup Language*) or *Turtle* [109]. Other OWL-specific syntaxes, like *Functional* [102] and *OWL/XML* [92], directly encode OWL axioms without converting them into RDF triples. Lastly, the *Manchester syntax* [46] is intended to be more user-friendly for those without a logics background. Various tools exist for translating between different syntaxes, which provides flexibility and adaptability for different requirements and preferences.

2.3.2 Automated reasoning in the Semantic Web

The close relationship with DLs allows OWL ontologies to capture complex domain knowledge, supporting automated reasoning to infer new knowledge based on explicitly defined facts and relationships. As a result, tools and systems built on OWL can make sophisticated deductions and answer complex queries by leveraging the foundational principles of DLs. Automated reasoning algorithms usually fall in one of the following categories:

- *Structural algorithms* focus on the form of logical expressions. They operate by examining and modifying the composition of these expressions, often converting them into more straightforward or standardized versions. These algorithms are particularly advantageous in scenarios where the logical structure is complex or plays a crucial role in the reasoning process and are generally very efficient (polynomial time). However, their drawback is that they are incomplete, *i.e.*, they can't identify all possible relationships for highly expressive DLs.

- *Tableaux-based algorithms* function by building a *tableau*, a tree-like configuration that represents different interpretations of the ontology. By expanding this tableau using specific rules and checking for inconsistencies, these algorithms can determine, for instance, whether an ontology is consistent or if a certain concept expression is satisfiable. This group of algorithms can manage much more expressive DL, though at a much higher (exponential, and sometimes double exponential) time and space complexity.

- *Rule-based algorithms* work by utilizing sets of predefined rules, typically in the *antecedent-consequent* format, applied to a set of asserted facts, thereby generating new (inferred) facts. In *forward chaining*, rules are applied iteratively to the expanding set of facts until a specific fact is deduced (*e.g.*, if the system is queried) or until no further facts can be derived. Conversely, *backward chaining* starts with the query and works backward by applying rules to determine which facts must hold true to satisfy the query. Rule-based algorithms are usually efficient, running in polynomial time relative to the input size, but they are quite rigid and do not support *non-monotonic* reasoning, *i.e.*, conclusions that can be retracted in light of new contradictory facts.

Regarding the $\mathcal{ALN}$ DL, comprehensive structural inference algorithms are widely established for both standard ([7], 2.3.1) and non-standard, non-monotonic inferences [113]. These algorithms rely on concept unfolding and Conjunctive Normal Form (CNF) normalization preprocessing steps. Essentially, **concept unfolding** involves the recursive expansion of terminological

axioms in the TBox within concept expressions, eliminating the need for the TBox in subsequent inferences. To guarantee finite unfoldings and ensure polynomial time and space complexity of inference procedures, $\mathcal{ALN}$ TBoxes must adhere to the following constraints [113]:

- the left-hand side (LHS) of inclusion ($\sqsubseteq$) and definition ($\equiv$) axioms must be atomic, *i.e.*, *general concept inclusions* are not allowed;
- if an atomic concept A is the LHS of a definition axiom, then it cannot be the LHS of any other inclusion or definition axiom;
- TBoxes must be *acyclic*, *i.e.*, the concept at the LHS of inclusion and definition axioms must not appear in the unfolding of the right-hand side (RHS) of the same axiom. This requirement can be partly relaxed, by allowing *told subsumption cycles* [139].

CNF normalization translates the unfolded concept expression in a canonical form that preserves its semantics. In $\mathcal{ALN}$ CNF, every concept expression is either $\bot$ or the conjunction ($\sqcap$) of:

- (possibly negated) atomic concepts;
- greater-than ($\geq$) and less-than ($\leq$) number restrictions, no more than one per type per role;
- universal restrictions ($\forall$), no more than one per role, with fillers recursively in CNF.

Given a DL ontology $\mathcal{T}$ and S, R two concepts in $\mathcal{T}$, the *satisfiability* and *subsumption* standard inference services provided by DL-based systems [7] can be formalized as follows:

- **Subsumption**: checks if S is more specific than R w.r.t. the ontology $\mathcal{T}$, *i.e.*, $\mathcal{T} \models S \sqsubseteq R$. In this case, all instances of S are also instances of R.

- **Satisfiability**: checks if S can have instances w.r.t. the ontology $\mathcal{T}$, *i.e.*, $\mathcal{T} \not\models S \sqsubseteq \bot$. An unsatisfiable class contains a contradiction, which implies that it cannot have any instance.

Once $\mathcal{ALN}$ concept expressions have been unfolded and normalized, subsumption and satisfiability can be carried out by looking at the structure of the expressions, comparing them as if they are sets of primitive atoms representing the $\mathcal{ALN}$ constructs in Table 2.1.

General knowledge-based applications usually require additional, more complex inference services over ontologies, such as *ontology coherence*, *consistency*, and *classification*:

- **Ontology Coherence** involves checking that all named concepts in the TBox are satisfiable [90]. If the ontology has an unsatisfiable class, then it is incoherent, though useful inferences can still be drawn from it. As such, incoherent ontologies can be and are often used in applications.
- **Ontology Consistency** determines whether it is possible to interpret the axioms in the ontology such that there is at least one class which has an instance. If the ontology is inconsistent, every class is interpreted as the empty set, and as such no useful conclusions can be drawn. This is generally regarded as a severe error in ontology modeling, and most reasoners just abort inferences when faced with this condition.
- **Ontology Classification** computes the overall concept taxonomy induced by the subsumption relation, from $\top$ to $\bot$. Essentially, classification is logically equivalent to computing all the subsumption relations between all pairs of concepts in the TBox. As such, it is a rather complex reasoning task, which requires careful optimization in order to be completed in reasonable time and space over large ontologies.

Chapter 3

Knowledge Representation and Reasoning for Cyber-Physical Systems

3.1 Network plane: semantic-based object network management

The rapid expansion of cloud computing, particularly in hybrid cloud configurations, and the deployment of IoT are transforming the conventional management of enterprise networks by introducing new challenges related to configurability, dynamism, and scalability. To address these issues, it is necessary to finely control network topology and communication capabilities in order to efficiently schedule the resource allocations required for on-demand services. The SDN paradigm, which is built on NFV, seeks to address these challenges: it implements core network components as software-based services that can be virtualized without the need for specialized hardware. This permits precise control and scaling of network components to meet user demands. SDN separates the control plane from the data plane, offering a logically centralized view of the network that simplifies the implementation of global network policies. Furthermore, AI can enhance existing capabilities and tools when used alongside low-level standard protocols for network

modeling and configuration.

To this aim, KDN is a novel paradigm that merges SDN, Network Analytics (NA), and AI to improve network operation, optimization, and management [87]. The concept of KDN derives from the idea of the Knowledge Plane (KP), which envisions a network control framework using ML and cognitive methods to automate network decision-making [27]. KDN overcomes complexity of distributed network systems by utilizing the centralized control of SDN and the in-depth data insights provided by advanced NA. This approach is crucial for KDN, as it provides the KP with a complete understanding of the network's state, enabling more effective learning and decision-making processes. At the same time, NA collect and analyze real-time data from various network elements, offering granular insights into traffic patterns, network performance, and potential issues. By combining these elements, KDN creates a feedback loop where the KP continuously refines its knowledge base and optimizes network operations based on real-time data and historical trends.

The KP within KDN acts as the intelligent core of the network, utilizing ML algorithms to process the vast amounts of data gathered by the network analytics. It transforms this data into actionable knowledge, which can then be used to automate network management tasks, optimize configurations, and predict future network behavior. The KP operates in close conjunction with the SDN controller, using an intent-driven language to translate high-level decisions into specific control directives that are executed across the network. This approach not only enhances the automation capabilities of the network but also allows for a more dynamic and adaptive response to changing network conditions, thereby improving overall network performance and reliability.

Integrating CPS and DLT in this context can provide significant advantages. CPS involves the integration of computer systems with physical processes, enabling real-time data collection and system optimization. When combined with DLT, which offers a decentralized and immutable record-keeping mechanism, the management of enterprise networks can achieve

higher levels of transparency, security, and reliability. This approach ensures that every transaction or change within the network is accurately recorded and easily auditable, fostering trust and enhancing compliance with regulatory standards.

In this context, a Decision Support System (DSS) has been proposed for network management and orchestration, employing a novel approach that integrates Semantic Web technologies with automated reasoning. The framework uses a two-level ontology, where a low-level ontology is generated from network information models expressed in Yet Another Next Generation (YANG) [16]. This is followed by mapping the low-level descriptions—annotated with data collected from network nodes—to high-level entities. The final KB is enriched with Semantic Web Rule Language (SWRL) [47] rules to perform specific actions, such as detecting requirement violations and suggesting corrective measures.

In Section 3.1.1, the ontology designed to represent the network's contextual characteristics is examined, along with the method for automatically generating annotations for network model instances, exploiting the classes and roles derived from the YANG information model. Subsequently, the semantic-driven DSS for network management and orchestration is introduced in Section 3.1.2.

3.1.1 Knowledge Representation for Software Defined Networks

The adopted methodology for automatic translation from network information model to domain ontology is described in what follows.

The data modelling language YANG [16] was initially created to define configurations and schemas managed by Network Configuration Protocol (NETCONF) [37]. According to the YANG specification, schema definitions consist of *modules* and *submodules*. **Modules** are independent documents that expose data definitions to other modules. **Submodules** contain

reusable data definitions that are accessible from other modules, but they cannot independently describe objects.

Each module includes a collection of declarations that specify domain entities (`container`, `list`), attributes (`leaf`, `leaf-list`), and reusable schemas or data node definitions (`grouping`, `typedef`) that can be imported into various documents. YANG is more expressive and flexible compared to other schema-definition languages, such as Extensible Markup Language (XML), as it allows for the creation of reusable modules, complex communication patterns, and extensions of existing modules (using the `augment` keyword). Moreover, YANG is extensively used in the network control industry for defining communication schemas between endpoints. Specifically, the YANG model is employed in various network standards, including the European Telecommunications Standards Institute (ETSI) Network Functions Virtualisation Solutions (NFV-SOL) protocol [60], the Open Network Foundation (ONF) Transport API (TAPI) [80], and in technical documentation from Cisco Systems, Internet Engineering Task Force (IETF), and Internet Assigned Numbers Authority (IANA). The method described focuses on transforming the YANG information model into an equivalent OWL ontology representation [103], facilitating the modeling of domain knowledge. This conversion is executed by *yang2OWL*, a specially designed tool whose architecture is illustrated in Figure 3.1.

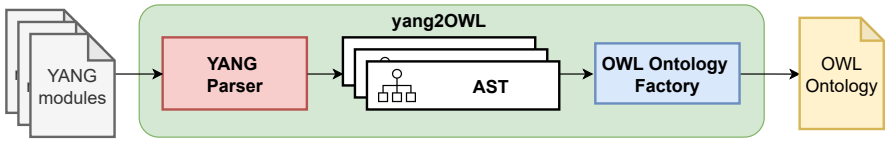


Figure 3.1: yang2OWL architecture

It consists of the following components:

- **YANG Parser**: it processes the input YANG modules and constructs an intermediate Abstract Syntax Tree (AST) to capture the syntactic structure of the input files. Each AST node represents a YANG statement and contains the corresponding *keyword*, *argument* and optional metadata.

- **OWL Ontology Factory**: it uses the intermediate data structures to produce the OWL ontology. In particular, each node representing a schema entity is translated into a OWL class, while the leaf attributes are converted to data properties.

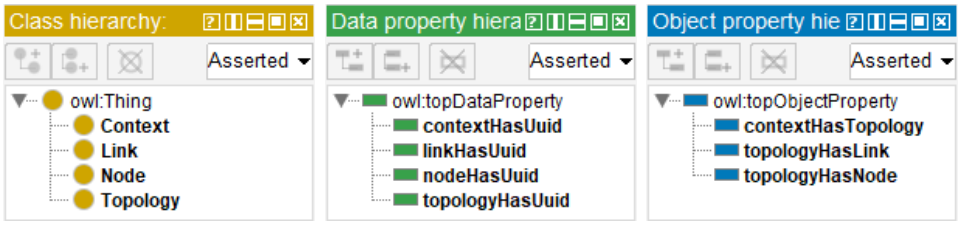
As an example, the YANG schema tree [17] shown in Figure 3.2a is transformed into the classes (Figure 3.2b) and properties (Figure 3.2c and Figure 3.2d).

1. The YANG module snippet defines a `context` container, which includes a list of `topology` items, each of which comprises two lists of `node` and `link` elements, respectively. Each entity includes a `uuid` basic attribute.
2. Every YANG object (`context`, `topology`, `node`, and `link`) is converted into the corresponding OWL class as displayed in Figure 3.2b.
3. The leaf attributes are linked to the data properties in Figure 3.2c.
4. If a YANG element contains children objects, the OWL Ontology Factory creates corresponding object properties to link the parent container with each child class. In the example, since `topology` contains a list of `nodes`, the `topologyHasNode` property is added (Figure 3.2d).

The proposed *yang2OWL* tool allows generating ontologies starting from different information models. For example, the tool is used to generate the ontology corresponding to the ONF TAPI YANG models. While the tool automatically produces the OWL ontology from YANG schema definitions, it also supports ABox (*i.e.*, Assertion Box, containing individuals and their relationships) generation from configuration instances related to the initial YANG modules. Typically, these configurations are encoded in a lightweight data interchange format, such as JavaScript Object Notation (JSON)[68]. To achieve this, a `SemanticAnnotator` has been developed in the software prototype mentioned in Section 3.1.2, which can enhance the KB with individuals derived from JSON documents obtained from the SDN controller interface.

```
module: tapi-common
  +--rw context!
     +--rw uuid?  uuid
     +--ro tapi-topology:topology* [uuid]
     |  +--ro tapi-topology:node* [uuid]
     |  |  |  +--ro tapi-topology:uuid    uuid
     |  +--ro tapi-topology:link* [uuid]
     |  |  |  +--ro tapi-topology:uuid    uuid
     |  +--ro tapi-topology:uuid    uuid
     ...
```

(a) YANG module snippet



(b) Classes **(c)** Data properties **(d)** Object properties

Figure 3.2: Example of the OWL ontology generated from YANG module

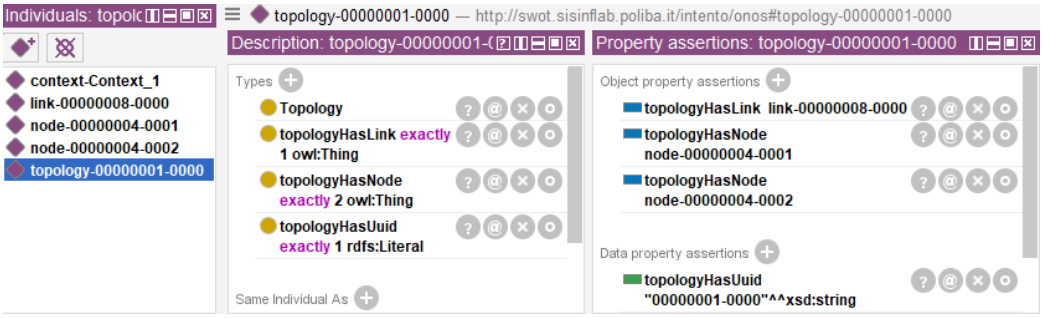
The Figure 3.3 illustrates an example of ABox generation for a JSON document, which details a network context with a single topology consisting of two nodes and a link (Figure 3.3a). The following translation rules are used.

1. Each entry represents either an atomic attribute (*e.g.*, `"uuid": "00000001-0000"`), a JSON object or a list of those (*e.g.*, `"link": [...]`).
2. Each object is translated as an OWL individual, with its key identifying the reference OWL class, and primitive properties are represented with data property assertions on the parent individual.
3. The root JSON Object is assumed to be of the same type as the root object specified in the YANG module (`Context`).

Following these guidelines, the translation of the previous JSON object

```
{
    "uuid": "Context_1",
    "topology": [{
        "uuid": "00000001-0000",
        "node": [
            {"uuid": "00000004-0001"},
            {"uuid": "00000004-0002"}
        ],
        "link": [{"uuid": "00000008-0000"}]
      }]
}
```

(a) JSON configuration



(b) Individuals generated from JSON document

Figure 3.3: Example of ABox generation from JSON document

results in five individuals: one of type `Context`, two `Nodes`, one `Link` and one `Topology`, as depicted in Figure 3.3b. The leaf `uuid` values are adapted as data properties, and object properties reflect the JSON structure, hence the `Topology` individual is related to the two `Node` individuals and the `Link` through the `topologyHasNode` and `topologyHasLink` properties, respectively.

The tool *yang2OWL* can obtain a detailed network description from the YANG information model and the contextual data provided by the SDN controller, which exposes a northbound TAPI RESTful connector. The generated ontology, however, lacks several logic relationships between the domain classes, since they cannot be modelled through YANG constructs, *e.g.*, both link `AvailableCapacity` and `CostCharacteristic` in TAPI definition

can be modelled as subclasses of a generic Key Performance Indicator (KPI) superclass which is useful in this scenario, but this is not explicitly stated in YANG modules.

To overcome this limitation, the proposed approach follows a two-level ontology design pattern [120]: relevant entities of the **TAPI KB** –containing all individuals generated through the `SemanticAnnotator` w.r.t. the ontology produced by *yang2OWL* (named **TAPI Ontology**)– are mapped to higher-level **DSS KB** entities, which embed the description of the contextual relationships between network components and are expressed w.r.t. a **DSS Ontology**, structured as shown in Figure 3.4.

- `NetworkEntity` and `NetworkServiceLevel` (`NSL`) respectively represent a generic network entity and a network quantitative characteristic (*e.g.*, bandwidth, latency, jitter, hops count, energy consumption).
- Each network entity can be further classified as a `Topology`, a `Node`, a `Link` or connectivity service (`Service`).
- Nodes having connectivity service endpoints are classified as `Host`s, otherwise they are treated as `Switche`s.
- `NSL` has three key properties: `hasTarget`, indicating the network entity associated with the measurement or constraint; `hasValue`, containing the reference value; `hasType`, indicating the KPI measure.
- `KPI` specializes into `NSLV` (Network Service Level Value class), which represents the actual measurements, and `NSLA` (Network Service Level Agreement) class, which models the *requirements* inferred from the TAPI Ontology or enforced by the user.
- Requirements are further classified either as `MaximumNSL` or `MinimumNSL` if they represent an upper or lower bound on the KPI, respectively. Instances of this type can be classified as `Violation` if they conflicts with one or more `NSLA`.

As an example, to represent the energy consumption of a `Node` individual, the system adds a `NSLV` individual to the KB, having `hasType` property set to "Energy", the `hasTarget` object property in relationship with the `Node` individual and the `hasValue` property set to the numeric value of energy consumption; `NSLA` individuals are defined in a similar way. If the DSS detects a violation between a `NSLA` and `NSLV`, the latter is reclassified as a `Violation`.

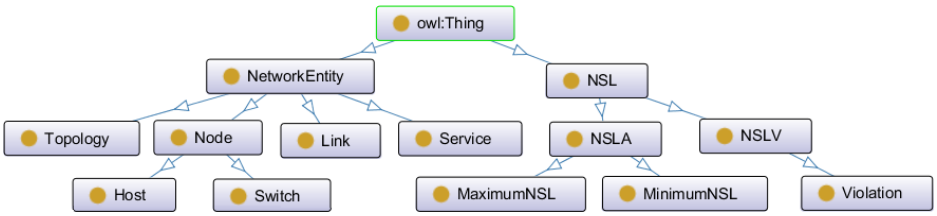


Figure 3.4: Semantic network management system - DSS Ontology

Following this approach, individuals in the DSS KB can be collected at runtime by mapping each relevant entity in the TAPI KB representing a node, link, topology, connectivity service, network service level requirement, or resource consumption to a new individual described w.r.t. the new DSS Ontology. Since OWL cannot easily represent "if-then-else" clauses, which are convenient in network orchestration scenarios, SWRL [47] is adopted to extend OWL capabilities by means of Horn clauses.

Essentially, SWRL is exploited to perform three tasks, which are described in the following.

1. **Infer additional DSS individual facts from TAPI ABox assertions**. An example is reported hereafter in a Protégé-like syntax [94], in which SWRL is used to retrieve the maximum link capacity for a `MaximumNSL` individual from the `TotalPotentialCapacity` individual in the TAPI ontology:

```
Link(?link), MaximumNSL(?nsl),
mappedFrom(?link, ?t_link), hasType(?nsl, "Capacity"),
linkHasTotalPotentialCapacity(?t_link, ?t_capacity),
availableCapacityHasTotalSize(?t_capacity, ?t_size),
```

```
totalSizeHasValue(?t_size, ?value) => hasValue(?nsl, ?value)
```

2. **Detect `NSLA` violations**. In the following example, the SWRL states that a `NSLV` greater than a corresponding agreement upper threshold implies a violation:

```
MaximumNSL(?nsla), NSLV(?nslv), NetworkEntity(?ne),
hasTarget(?nsla, ?ne), hasTarget(?nslv, ?ne),
hasType(?nsla, ?type), hasType(?nslv, ?type),
hasValue(?nsla, ?req), hasValue(?nslv, ?value),
greaterThan(?value, ?req) => Violation(?nslv)
```

3. **Execute actions on individuals when specific conditions are verified**. As shown in the following rule, when an energy constraint violation occurs for a particular network topology, topology rearrangement must be executed:

```
Violation(?v), Topology(?t), hasType(?v, "Energy"),
hasTarget(?v, ?t) => execute(recompute_nodes, ?t)
```

3.1.2 Resource orchestration: a semantic-based approach

The proposed DSS is able to automatically detect network Quality of Service (QoS) violations and to suggest the appropriate corrective actions. Figure 3.5 shows the overall architecture. It integrates reasoning capabilities by means of *Owlready2* [64], a Python module which allows mixing object-oriented and ontology-based programming patterns, as it maps OWL classes, instances and properties to the corresponding built-in Python language constructs.

An abstract `OntologyManager` class is defined to include several common methods. For example, the `update` method is used to generate the ABox individuals at runtime. In the proposed system, this abstract class is specialized as `TAPIOntologyManager`, to handle the ontology generated from YANG modules, and as `DSSOntologyManager`, to cope with the high level DSS ontology. Furthermore, the `OntologyManager` allows extending system

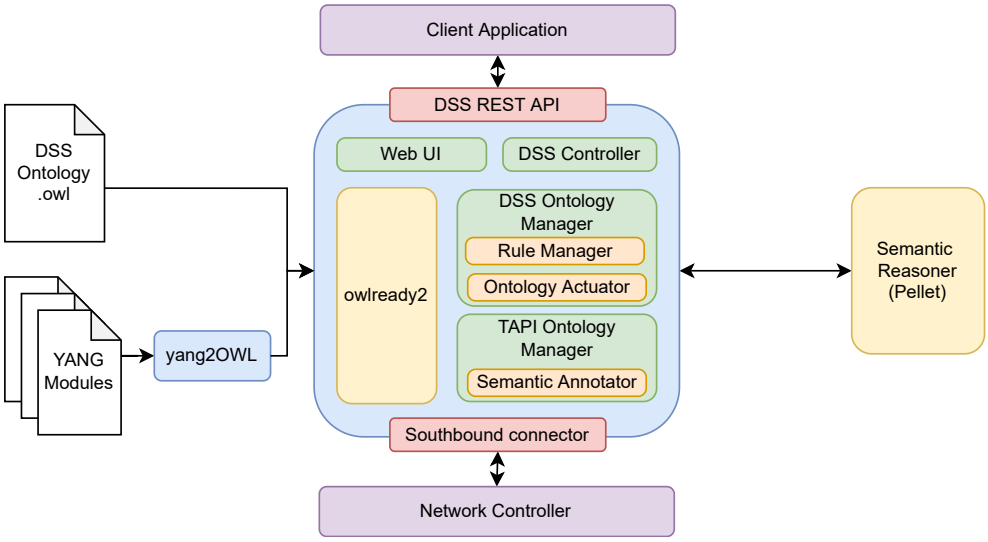


Figure 3.5: Semantic network management architecture

functionalities by means of the plugin design pattern. The following plugins have been provided:

- **SemanticAnnotator**, previously introduced in section Section 3.1.1, allows `TAPIOntologyManager` to translate the network descriptions in JSON format to OWL Individuals;
- **RuleManager** provides Create, Read, Update and Delete (CRUD) operations to edit SWRL rules at runtime;
- **OntologyActuator** automates the execution of tasks. The component exploits the `execute` object property, which can be specified as the tail of a SWRL rule, in the form of `execute(?f, ?t)` atom, where `?f` represents a Python function and `?t` is an individual, so that whenever the head of the rule is verified, OntologyActuator invokes the Python `?f` function passing `?t` as an argument.

When the system starts, it loads both the DSS ontology and the TAPI ontology, using Owlready2. The sequence diagram in Figure 3.6 shows how

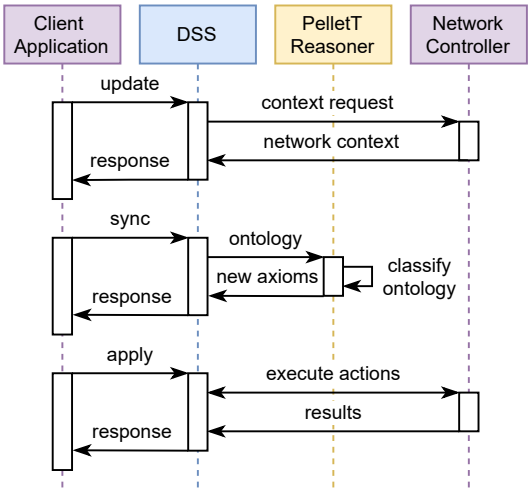


Figure 3.6: Semantic network management system - Sequence diagram

the components interact when a specific operation is executed. The system can execute the following three basic operations:

- **Update**: invokes the `update` method for each `OntologyManager` to collect ABox individuals. In particular, the `TAPIOntologyManager` queries the SDN controller to retrieve the network context and exploits the `SemanticAnnotator` to generate the OWL individuals from the response. Subsequently, the `DSSOntologyManager` updates the high-level contextual individuals w.r.t. the newly collected TAPI individuals;
- **Sync**: leverages the reasoning capabilities of the *Pellet* [130] reasoner to infer implicit knowledge from the KB and apply the SWRL rules, in order to perform the tasks described in Section 3.1.1;
- **Apply**: exploits the functions exposed by the `OntologyActuator` to execute the suggested corrective actions.

The system is intended to be controlled by third-party client applications through the `DSS REpresentational State Transfer (REST) Application Programming Interface (API)`. A minimal Web glsUI –relying on the REST API– is also provided, which summarizes the DSS performance metrics and

the semantic-based outcomes (like the active `NSLAs` and the suggested corrective actions) and interacts with the `DSSController` to define new `NSLAs` or `NSLVs`, or to execute one of the aforementioned `update`, `sync` and `apply` actions.

To preliminarily validate the functionality of the proposed framework, the prototype system was evaluated in a small test environment, featuring a basic network of four nodes interconnected by four links, overseen by a network controller with a northbound TAPI interface. The context specification includes Network Service Level (NSL) details, such as capacity and the number of hops. Initially, the DSS Web UI displays only generic ontology data, as it lacks information about the network status. Therefore, `update` and `sync` commands were executed to collect factual knowledge about the underlying topology. Because the service permits third-party applications to define `NSLVs`, both TAPI-specific data (including hop count and node capacity) and external metrics (e.g., energy consumption) were taken into account. Following these operations, the Web UI accurately reflects the updated information.

Figure 3.7 reports two dashboard screenshots, showing different system states. Initially, Figure 3.7a shows the `NSLA` and `NSLV` values, as well as the network topology with the connectivity services already enabled. Since no requirement violation has been detected, the system does not suggest any corrective action. Figure 3.7b shows the system status after the insertion of a new NSL constraint by the user, through the REST Connector API.

This new requirement is identified with the name *user.energy.max.001* and has a value < 10 units. Measurement *energy.00000001-0000* violates the new constraint, since its value is 14. The system therefore suggests to execute path computation of the connectivity service. After applying the suggested corrective action, the system recomputes the path for each connectivity service, suggesting to turn off the nodes *00000003-0001* and *00000004-0001*, since they are not needed to keep the connectivity service alive.

Inferred Action	Target Instance

NSLA	Target	Metric	Value
capacity.max.00000004-0002	mapped_node-00000004-0002	Capacity	< 10
capacity.max.00000003-0002	mapped_node-00000003-0002	Capacity	< 10

NSLV	Target	Metric	Value	Violation
capacity.00000004-0002	mapped_node-00000004-0002	Capacity	10	False
energy.00000001-0000	mapped_topology-00000001-0000	Energy	14.0	False
hops.00000023-0000	mapped_link-00000023-0000	Hops	10	False

Service	EndPoints
connectivity-service-00000001-0000	["node-00000003-0002", "node-00000004-0002"]

Topology

(a) System requirements - No violation

Inferred Action	Target Instance
recompute_nodes	mapped_topology-00000001-0000

NSLA	Target	Metric	Value
capacity.max.00000004-0002	mapped_node-00000004-0002	Capacity	< 10
capacity.max.00000003-0002	mapped_node-00000003-0002	Capacity	< 10
user.energy.max.001	mapped_topology.00000001-0000	Energy	< 10

NSLV	Target	Metric	Value	Violation
capacity.00000004-0002	mapped_node-00000004-0002	Capacity	10	False
energy.00000001-0000	mapped_topology-00000001-0000	Energy	14.0	True
hops.00000023-0000	mapped_link-00000023-0000	Hops	10	False

Service	EndPoints
connectivity-service-00000001-0000	["node-00000003-0002", "node-00000004-0002"]

Topology

(b) User requirement - Violation

Figure 3.7: DSS Dashboard

3.2 Processing plane: multiplatform reasoning architecture

As mentioned in Section 2.1.1, within the context of CPS architecture, it is essential to manage the integration of computational and physical components effectively. DLT provides a decentralized solution that enhances the transparency and security of interactions among distributed CPS components. Particularly, the Cognitive Layer of CPS, which is in charge of sophisticated reasoning and decision-making, can benefit from the use of semantic technologies and smart contracts.

For CPS operating over DLT, the Cognitive Layer can exploit automated

reasoning capabilities to manage and coordinate distributed components effectively. Tiny-ME, the Tiny Matchmaking Engine, provides a multiplatform architecture with a core implemented in C, ensuring both portability and efficient implementation of KB management and reasoning services. Tiny-ME supports smart contracts by facilitating reasoning and matchmaking in the Semantic Web of Everything (SWoE) context, enabling intelligent interactions and decision-making processes.

Tiny-ME [115] offers both standard (Ontology Classification, Ontology Coherence, Concept Subsumption, and Concept Satisfiability checks) and non-standard (Concept Abduction, Contraction, Bonus, Difference, and Covering) inference services for moderately expressive KBs in an OWL 2 subset corresponding to the *Attributive Language with unqualified Number restrictions* ($\mathcal{ALN}$) DL. This choice enables efficient execution of inference algorithms, suitable for resource-constrained devices typically used in CPS.

Tiny-ME is interoperable with major desktop and mobile operating systems (OSes) and can be deployed in *Docker*[1] containers, facilitating its use in microservice architectures for Web and Cloud applications. The standard *OWLlink* [72] protocol has been extended to include Tiny-ME's non-standard inference services for remote invocation. Furthermore, Tiny-ME has been ported to the *Apache NuttX*[2] real-time operating system Real-Time Operating System (RTOS) for the *Pixhawk*[3] open standard platform for Unmanned Aerial Vehicle (UAV) autopilots. This case study demonstrates Tiny-ME's adaptability to embedded systems with stringent constraints and its potential for CPS over DLT.

The core features of Tiny-ME [115] are reported as follows:

- The Tiny-ME SWoE-oriented reasoning and matchmaking engine, designed to support smart contracts and automated reasoning within the Cognitive Layer of CPS operating over DLT, providing both standard

[1]Docker home: `https://www.docker.com/`
[2]Apache NuttX home: `https://nuttx.apache.org/`
[3]Pixhawk home: `https://pixhawk.org/`

and non-standard inference services in a moderately expressive OWL 2 fragment corresponding to the $\mathcal{ALN}$ DL;

- A new multiplatform architecture that supports various operating systems including Windows, Linux, macOS, Docker containers, Android, and iOS, reducing porting efforts to other platforms;
- Re-engineered inference procedures within SWoT/ SWoE frameworks, which can be embedded in CPS contexts to enhance Cognitive Layer's capabilities, *e.g.*, implementing DSS on edge nodes;
- Cowl, a new C parser for OWL 2 Full, embedded in Tiny-ME and released independently with a permissive open source license, promoting reuse [12, 13];
- Extension of the OWLlink interface for HyperText Transfer Protocol (HTTP)-based application-reasoner interaction, supporting non-standard inference services;
- A case study on the Pixhawk UAV autopilot demonstrating (i) the ease of porting to the Apache NuttX RTOS and (ii) the feasibility of implementing a useful application given the hardware constraints of embedded devices.

Elements like Cowl, the OWLlink extension, and the multiplatform architectural model can also be adapted for other ontology-based software systems. The remainder of this section is organized as follows: Section 3.2.1 details Tiny-ME's inference services and high-level architecture. Section 3.2.2 presents the Pixhawk case study, followed by early performance results.

3.2.1 Reasoning in the Semantic Web of Everything

The combination of DLTs and CPSs has pioneered novel methods for integrating reasoning services. These services can be performed seamlessly within CPS systems, allowing for enhanced capabilities and functionalities.

Table 3.1: $\mathcal{ALN}$ constructors supported by Tiny-ME

Name	DL syntax	Manchester syntax
Top	$\top$	`owl:Thing`
Bottom	$\bot$	`owl:Nothing`
Concept	C	*C*
Role	R	*R*
Conjunction	$C \sqcap D$	*C* `and` *D*
Atomic negation	$\neg A$	`not` *A*
Universal restriction	$\forall R.C$	`R only C`
Unqualified number restrictions	$\geq n\ R$ $\leq n\ R$	*R* `min n` *R* `max n`
Definition axiom	$A \equiv C$	`Class:` *A* `EquivalentTo:` *C*
Inclusion axiom	$A \sqsubseteq C$	`Class:` *A* `SubClassOf:` *C*

In SWoT contexts, performance constraints of computing devices are strict and require careful software design choices. Adding new constructors makes DL languages more expressive, but leads to an increase in computational complexity of inference services [19]. Hence a tradeoff is needed, the proposed system employs structural algorithms with polynomial complexity on $\mathcal{ALN}$ concept expressions (see Table 3.1 for the list of supported constructs), utilizing the techniques of *unfolding* and *CNF* normalization as preprocessing steps [127].

Structural algorithms for standard reasoning tasks are well-known (see [7], 2.3.1). In $\mathcal{ALN}$ DL, once concept expressions have been unfolded and normalized, comparisons between them basically come down to set operations. As an example, let us consider the following $\mathcal{ALN}$ axioms (named TB_{ex}):

$$A \equiv \forall P.D \sqcap (\geq 3P)$$
$$B \sqsubseteq \forall P.D$$
$$C \sqsubseteq B \sqcap (\geq 2P)$$

By applying unfolding and CNF normalization, we get the following con-

cept expressions (CEs):

$$A \rightarrow \forall P.D \sqcap (\geq 3P)$$
$$B \rightarrow B \sqcap \forall P.D$$
$$C \rightarrow B \sqcap C \sqcap \forall P.D \sqcap (\geq 2P)$$

It can be noticed B and C appear among the conjuncts of their own unfolded concept expressions, while A does not: as explained in [7, §9.2.3], in $\mathcal{ALN}$ DL an atomic concept must be included in its own unfolding iff it is the LHS of an inclusion axiom, while it is omitted if it is the LHS of an equivalence one.

Let us suppose we need to check whether $A \sqsubseteq C$ holds: after unfolding and normalizing, we just need the CEs of A and C, *i.e.*, the whole TBox is not required anymore. In this case, $A \sqsubseteq C$ holds because $\forall P.D$ is in both the CEs of A and C, and $(\geq 3P)$ is more specific than $(\geq 2P)$.

Tiny-ME provides *standard* checks for Subsumption and Satisfiability in concept expressions. **Satisfiability** can be effortlessly verified by performing CNF normalization [127]: as mentioned earlier, a CNF-normalized concept expression A is either $\bot$, or a conjunction of various supported constructs. To determine if A is satisfiable, it is enough to confirm that it is not $\bot$. **Subsumption** uses the classic structural algorithm from [7]. The procedure is detailed in algorithm 1.

Algorithm 1: Subsumption Check

Input : Two CNF-normalized concept expressions R and S

Output: Boolean result for Subsumption $R \sqsubseteq S$

```
1  if R ≡ ⊥ then
2      return R ⊑ S = True ;                          // R subsumes everything
3  foreach atomic concept A in S do
4      if A is not in R then
5          return R ⊑ S = False ;                     // Atomic concept not found in R
6  foreach negated atomic concept ¬A in S do
7      if ¬A is not in R then
8          return R ⊑ S = False ;                     // Negated atomic concept not found in
                                                          R
9  foreach role P such that ≤ xP is in S do
10     if ≤ yP with x < y is in R then
11         return R ⊑ S = False ;                     // Role cardinality not met in R
12 foreach role P such that ≥ xP is in S do
13     if ≥ yP with x > y is in R then
14         return R ⊑ S = False ;                     // Role cardinality not met in R
15 foreach role P such that ∀P.E is in S do
16     if ∀P.F with F ⋤ E is in R then
17         return R ⊑ S = False ;                       // Universal role not met in R
18 return R ⊑ S = True ;                              // All checks passed; R subsumes S
```

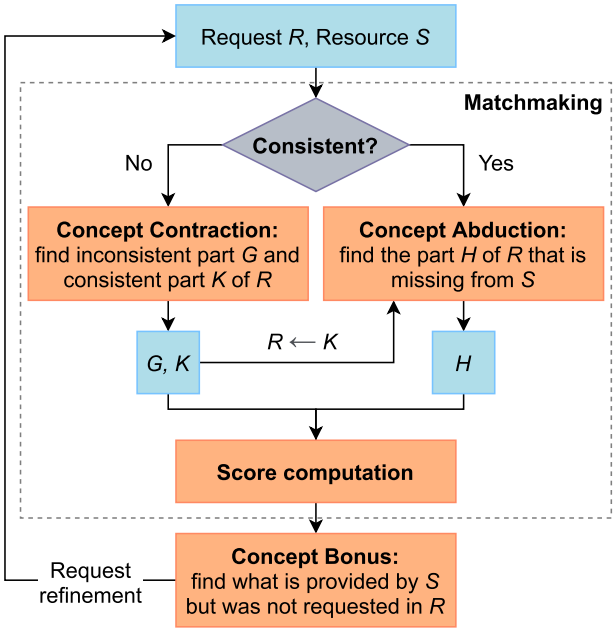


Figure 3.8: Semantic matchmaking framework

Standard inference services are not enough in scenarios requiring more than just a Boolean answer, such as in matchmaking or negotiation. In those cases, *non-standard* Concept Abduction and Concept Contraction [113] are more useful, as they provide justifications for (missed) subsumption and (un)satisfiability, in the Open World Assumption (OWA). Tiny-ME supports the *semantic matchmaking* framework sketched in the flow chart of Figure 3.8 and outlined hereafter [115].

Consider a set of axioms $\mathcal{T}$ in $\mathcal{ALN}$, and let R and S be two concepts in $\mathcal{L}$ that represent a *request* and a *resource*, respectively, and are both satisfiable in $\mathcal{T}$. Initially, a **Consistency Check** is conducted to determine whether $R \sqcap S$ is satisfiable with respect to $\mathcal{T}$. Formally, this is expressed as $\mathcal{T} \models R \sqcap S \not\sqsubseteq \bot$. If this check does not succeed, the resource is a *partial match* for the request, and **Concept Contraction (CC)** can be performed.

The result of CC is a pair of concepts $\langle G, K \rangle$ such that $\mathcal{T} \models R \equiv G \sqcap K$ and $\mathcal{T} \models K \sqcap S \not\sqsubseteq \bot$. In essence, Contraction identifies the portion of R that conflicts with S. By discarding only the conflicting requirements G (for *Give up*) from R, what remains is the concept K (for *Keep*), representing a

revised version of the initial request. The solution G arising from Contraction clarifies the reason why the conjunction of R and S is unsatisfiable, thus enabling a transition from a partial match to a *potential match* scenario. When R and S are expanded and expressed in CNF in $\mathcal{ALN}$, $CC(R, S)$ can be determined using the algorithm algorithm 2. For example, referring to the above TB_{ex}, $\langle G, K\rangle = CC(A, \leq 2\ P) = \langle \geq 3\ P, \geq 2\ P\rangle$.

Algorithm 2: Concept Contraction $CC(R, S)$

Input : Two CNF-normalized concept expressions R and S in $\mathcal{ALN}$

Output: A pair of concepts $\langle G, K\rangle$ where G is the conflicting part, and K is the revised request

Initialization:
Set $K := R$ and $G := \top$; `// Initialize` K `to` R `and` G `to` $\top$ `(no conflict yet)`
foreach *atomic concept* A *in* K **do**
    if $\neg A$ *is in* S **then**
        Move A from K to G ; `// Conflict detected, move` A `to` G
foreach *negated atomic concept* $\neg A$ *in* K **do**
    if A *is in* S **then**
        Move $\neg A$ from K to G ; `// Conflict detected, move` $\neg A$ `to` G
foreach *role* P *such that* $\geq xP$ *is in* K *and* $\leq yP$ *is in* S *with* $y < x$ **do**
    Replace $\geq xP$ in K with $\geq yP$;
    Conjoin $\geq xP$ with the concept expression for G ; `// Adjust` K `and` G
foreach *role* P *such that* $\leq xP$ *is in* K *and* $\geq yP$ *is in* S *with* $y > x$ **do**
    Replace $\leq xP$ in K with $\leq yP$;
    Conjoin $\leq xP$ with the concept expression for G ; `// Adjust` K `and` G
foreach *role* P *such that* $\forall P.E$ *is in* K *and* $\forall P.F$ *is in* S **do**
    if $\exists \geq xP$ *with* $x > 0$ *is either in* K *or* S **then**
        Compute $\langle G', K'\rangle = CC(E, F)$ recursively;
        Conjoin $\forall P.G'$ with the concept expression for G;
        Replace $\forall P.E$ with $\forall P.K'$ in K ; `// Resolve conflicts`
return $\langle G, K\rangle$; `// Return the conflicting` G `and the revised request` K

Concept Abduction (CA) can be computed in case of potential match, which occurs if S does not clash with R but is not subsumed by it, *i.e.*, $\mathcal{T} \models R \sqcap S \not\sqsubseteq \bot$ and $\mathcal{T} \models S \not\sqsubseteq R$. The output of CA consists of a concept $H \in \mathcal{L}$ such that $\mathcal{T} \models S \sqcap H \sqsubseteq R$ and $S \sqcap H$ is satisfiable in $\mathcal{T}$. It is important to highlight that CA along with other inference services for semantic matchmaking in Tiny-ME [115] are based on the OWA [113], meaning that missing details in a concept expression do not imply negation, but rather signify an unspecified constraint, possibly due to unknown factors or considered negligible.

The solution H (representing *Hypothesis*) can be seen as the part of R that is required and not found in S, offering a rationale for the absent Subsumption and a method to transition from a *potential* to a *full match* (a.k.a. *subsume match* [69, 101]), which is the intended result of the matchmaking system and occurs when $S \sqsubseteq R$, implying that all requested features are supplied by the resource[4]. With R and S unfolded and CNF-normalized concept expressions in $\mathcal{ALN}$, $CA(R, S)$ is elaborated in the subsequent structural algorithm 3. For example, referring to the above TB_{ex}, $H = CA(A, B) = \geq 3\ P$.

Concept Bonus (CB) is useful in matchmaking scenarios, as a resource S might include attributes not specified in R –either because the requester was unaware or indifferent– which can be utilized in a query enhancement process. CB identifies and provides a *Bonus* concept B from S, indicating what the resource offers even if it was not requested. The method for determining the Bonus B of S w.r.t. R is identical to the CA problem where R and S roles are reversed, *i.e.*, R is considered the resource and S the request.

[4] *Exact match*, where $S \equiv R$, is the optimal outcome, but a full match is equally favorable from the requester's perspective as all their preferences are satisfied.

Algorithm 3: Concept Abduction $CA(R, S)$

Input : Two CNF-normalized concept expressions R and S in $\mathcal{ALN}$

Output: A concept H (Hypothesis) such that $\mathcal{T} \models S \sqcap H \sqsubseteq R$ and $S \sqcap H$ is satisfiable

Initialization:
Set $H := \top$; `// Initialize` H `as the most general concept`
foreach *(possibly negated) atomic concept* A *in* R **do**
    if $\nexists B$ *in* S *such that* $B \sqsubseteq A$ **then**
        Conjoin A with the concept expression for H ; `// Add` A `to` H `if not subsumed by any concept in` S
foreach *role* P *such that* $\geq xP$ *is in* R **do**
    if $\geq yP$ *is not in* S ***or*** $\geq yP$ *is in* S *with* $y < x$ **then**
        Conjoin $\geq xP$ with the concept expression for H ; `// Add` P `with cardinality restriction to` H `if not satisfied by` S
foreach *role* P *such that* $\leq xP$ *is in* R **do**
    if $\leq yP$ *is not in* S ***or*** $\leq yP$ *is in* S *with* $y > x$ **then**
        Conjoin $\leq xP$ with the concept expression for H ; `// Add` P `with cardinality restriction to` H `if not satisfied by` S
foreach *role* P *such that* $\forall P.E$ *is in* R *and* $\forall P.F$ *is in* S **do**
    Compute $H' = CA(E, F)$ recursively;
    Conjoin $\forall P.H'$ with the concept expression for H ; `// Resolve abduction within role fillers recursively`
return H ; `// Return the hypothesis` H `that explains the potential match`

The $\mathcal{ALN}$ CNF for concept expressions establishes a metric space outlined by a *norm* operator $\|\cdot\|$. In the Tiny-ME matchmaking framework [115], the CNF norm of G and H denotes a semantic distance **penalty** for CC and CA, respectively, helping to classify resources pertinent to a specified request. Likewise, $\|B\|$ functions as a relevance metric for the Bonus. The algorithm 4 calculates the penalty values proposed in [113].

Algorithm 4: Penalty Calculation for Concept Expressions

Input : CNF-normalized concept expressions G and H in $\mathcal{ALN}$
Output: Penalty value for G and H

Initialization:
Set $P := 0$; `// Initialize penalty to 0`
foreach *(possibly negated) atomic concept* A *in* G **do**
    if A *is not in* H **then**
        $P := P + 1$; `// Each atomic concept counts as 1`
foreach *number restriction* $\geq xP$ *in* G **do**
    if *number restriction* $\geq yP$ *is in* H **then**
        $r := \frac{|x-y|}{x}$; `// Ratio of the difference of cardinality`
    else
        $r := 1$; `// Penalty is 1 if` H `lacks this number restriction`
    $P := P + r$
foreach *number restriction* $\leq xP$ *in* G **do**
    if *number restriction* $\leq yP$ *is in* H **then**
        $r := \frac{|x-y|}{x}$; `// Ratio of the difference of cardinality`
    else
        $r := 1$; `// Penalty is 1 if` H `lacks this number restriction`
    $P := P + r$
foreach *universal restriction* $\forall P.E$ *in* G **do**
    if $\forall P.F$ *is in* H **then**
        Compute penalty recursively for fillers: $r := Penalty(E, F)$
    else
        $r :=$ Penalty for missing universal restriction ; `// Defined as needed`
    $P := P + r$
return P ; `// Return the total penalty`

For CA, CB, and CC, the summarized algorithms aim to a minimality criterion, since one usually wants to hypothesize or give up as little as possible. Conversely, a maximality criterion is adopted for the **Concept Contraction (CD)**, which defines a way to subtract information in a concept description from another one: in the original definition by Teege [135], if $\mathcal{T} \models R \sqsubseteq S$, then the output of difference $D = CD(R, S)$ is a concept $D \in \mathcal{L}$ such that $R \equiv S \sqcap D$. In that case, one typically wants to subtract as much as possible. In Tiny-ME, the applicability of CD has been extended from the full match (*i.e.*, subsumption) case to potential and partial matches as well, by exploiting CC and CB [115]. In detail, given two unfolded and CNF-normalized $\mathcal{ALN}$ concept expressions R and S, $CD(R, S)$ is computed structurally as illustrated in algorithm 5.

Algorithm 5: Concept Difference $CD(R, S)$

Input : CNF-normalized concept expressions R and S
Output: Concept difference D

Initialization:
if $R \sqcap S \not\sqsubseteq \bot$ **then**
  $K := S$; `// If` R `and` S `are consistent, initialize` K `to` S
else
  $\langle G, K \rangle := CC(R, S)$; `// Perform concept contraction to obtain` K
$D := CB(R, K)$; `// Calculate` D `using Concept Bonus`
return D ; `// Return the concept difference` D

For example, referring again to TB_{ex}, $CD(B, A) = B$ and $CD(A, B) =\geq 3\ P$.

While CA, CB, CC, and CD are useful in one-to-one discovery, matchmaking and negotiation scenarios, Tiny-ME also includes the **Concept Covering (CCov)** non-standard inference for many-to-one composition of a set of elementary instances to answer complex requests [127]. Basically, CCov takes a set of resources and a request and aims to (i) cover (*i.e.*, satisfy) features expressed in the request as much as possible through the conjunction of resources, and (ii) provide an explanation of the possibly uncovered part. In formulae, given a concept expression R (request) and a set of concept

expressions $\mathcal{S} = \{S_1, S_2, ..., S_n\}$ in a KB (available resources), where R and $S_1, S_2, ..., S_n$ are satisfiable in the reference ontology $\mathcal{T}$, the output of CCov is a pair $\langle \mathcal{S}_c, H \rangle$ where $\mathcal{S}_c \subseteq \mathcal{S}$ contains concepts in S forming a (possibly incomplete) covering of R w.r.t. $\mathcal{T}$ and concept H is the (possible) part of R not covered by elements in $\mathcal{S}_c$. The detailed workflow of CCov is formalized in algorithm 6

Algorithm 6: Concept Covering $CCov(R, \mathcal{S})$

Input : Concept expression R (request) and a set of concept expressions $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ (available resources) unfolded and CNF-normalized

Output: A pair $\langle \mathcal{S}_c, H \rangle$ where $\mathcal{S}_c \subseteq \mathcal{S}$ and H is the uncovered part of R

Initialization:
Set $\mathcal{S}_c := \emptyset$; // Set of concepts covering R
Set $H := R$; // Initial uncovered part of R
while $S_{max} \not\equiv \top$ **do**
    Set $r_{min} := \|H\|$; // Initialize minimum penalty to the norm of H
    Set $S_{max} := \top$; // Initialize the maximum covering resource
    Set $H_{max} := H$; // Initialize the best covering result
    foreach $S_i \in \mathcal{S}$ **do**
        if $S_i \sqcap R \not\sqsubseteq \bot$ ***and*** $CD(S_i, H) \not\equiv \top$ **then**
            Compute $H_i, r_i := CA(H, S_i)$; // Perform Concept Abduction
            if $r_i < r_{min}$ **then**
                Update $r_{min} := r_i$;
                Update $S_{max} := S_i$;
                Update $H_{max} := H_i$;
    if $S_{max} \not\equiv \top$ **then**
        Add S_{max} to $\mathcal{S}_c$; // Include S_{max} in the covering set
        Remove S_{max} from $\mathcal{S}$; // Remove S_{max} from the available resources
        Set $H := H_{max}$; // Update the uncovered part of R
return $\mathcal{S}_c, H$; // Return the covering set and the uncovered part of R

Each iteration of the internal loop (line 8) computes H_i and r_i respectively as the CA hypothesis and penalty w.r.t. the remaining uncovered part H. The resource with minimal penalty, *i.e.*, with maximal covering of H, is added to the set $\mathcal{S}_c$, until no resources further increase the covering.

Furthermore, the following inference services are executed using Tiny-ME [115], which employs optimized structured algorithms, making them suitable for deployment in embedded systems, particularly in CPS scenarios involving DLT technologies.

- **Ontology Coherence**: it entails verifying that all specified concepts in the TBox are satisfiable [90]. According to [116], instead of determining this by directly checking the satisfiability of every Terminological Box (TBox) concept, Tiny-ME utilizes a modified Classification algorithm that terminates upon identifying any unsatisfiable concept. This strategy has been adopted over the naive method as the majority of time consumed by structural inference algorithms is taken by unfolding and CNF normalization. By minimizing Subsumption checks (and consequently unfolding and normalization), Classification typically yields substantially better performance.
- **Ontology Classification**: it computes the overall concept taxonomy induced by the subsumption relation, from $\top$ (*Top* a.k.a. *Thing*) to $\bot$. The system adopts a variant of the *enhanced traversal* algorithm in [8], with a number of optimizations, such as caching of subsumption check results, exploitation of told subsumers and disjoints [139], and caching of unfolded and normalized concept expressions [116].

High-level architecture. The comprehensive high-level architecture is illustrated in Figure 3.9 and elaborated below:

- *Core layer:* This layer is developed in standard C11 without any compiler extensions or platform-specific API calls. It includes a highly optimized implementation of both standard and non-standard inference

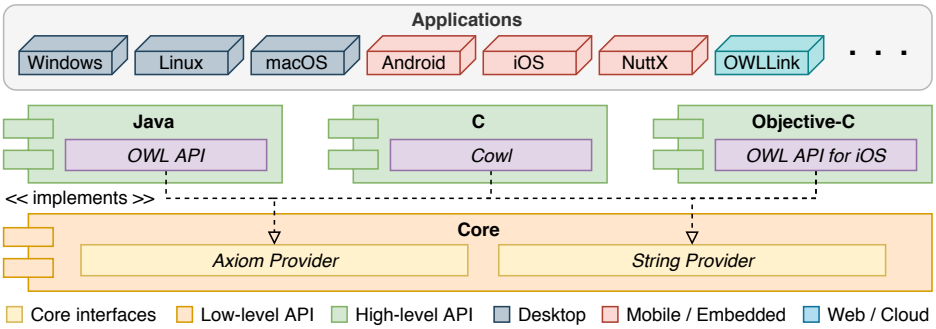


Figure 3.9: High-level architecture

algorithms and their supporting data structures. This layer is designed to be completely independent of knowledge representation or storage methods: $\mathcal{ALN}$ OWL entities (*i.e.*, named constructs like classes, object properties, and named individuals) are converted into numerical identifiers termed *entity pointers*, and their string form is never needed for reasoning. This capability is utilized by the *Axiom provider* interface, which the reasoner consults to obtain structured representations of KB axioms. Additionally, the Core layer offers an optional *logging* API that queries the *String provider* interface for the string forms of entity pointers. In a distributed SWoE setup, certain devices might not require string representation features or may lack the memory to handle them; in such cases, they can implement only the Axiom provider API and still perform reasoning tasks on (unlabeled) entities.

- *Platform-specific APIs:* The architecture's modularity supports the implementation of various APIs in different programming languages. Generally, using C11 for the reasoning core allows for a broad range of possible higher-level APIs, as most programming languages' runtimes provide at least basic compatibility with C code.

3.2.2 Case study and experiments

To evaluate the portability of the proposed Tiny-ME reasoner and its suitability for use on SWoT resource-constrained devices, an experimental setup

was carried out using the *3DR IRIS+*[5] UAV (also known as a drone). The UAV was equipped with the following components:

- ***Pixhawk 1***[6]: a reliable and open-source autopilot platform that functions as the flight controller. It handles the stabilization of the drone and oversees flight operations like altitude, orientation, and navigation.
- ***STM32F427***[7]: a system-on-chip (SoC) that includes a 180 MHz ARM Cortex M4 CPU with 256 kB of SRAM. This microcontroller provides sufficient computational power for real-time data processing and control in embedded systems like UAVs, while maintaining energy efficiency.
- ***PX4***[8]: the flight control firmware used in the experiment, specifically the FMUv2 version. PX4 runs on top of the Apache NuttX RTOS (Real-Time Operating System), which allows developers to create and integrate custom applications and modules tailored to specific tasks or operational needs. The modular design of PX4 makes it particularly suitable for adding custom software to extend the UAV's capabilities[9].

This setup provides an ideal platform to assess the performance of the reasoner under the constraints typically found in SWoT-enabled environments, including limited processing power, memory, and real-time requirements.

Basically, development for the Pixhawk system requires configuring the *CMake*[10] build system on the computer used for cross-compilation. This involves specifying the location of source files, as well as setting up instructions for the build and deployment processes. CMake is used to build the custom firmware and deploy it to the Pixhawk device, which is connected

[5]IRIS+ home: `https://3dr.com/support/articles/iris/`
[6]Pixhawk 1 home: `https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk.html`
[7]STM32F427 home: `https://www.st.com/en/microcontrollers-microprocessors/stm32f427-437.html`
[8]PX4 home: `https://px4.io`
[9]PX4 developer documentation: `https://dev.px4.io`
[10]CMake home: `https://cmake.org`

via USB. The setup process is illustrated in Figure 3.10, where the testbed environment for deployment is depicted.

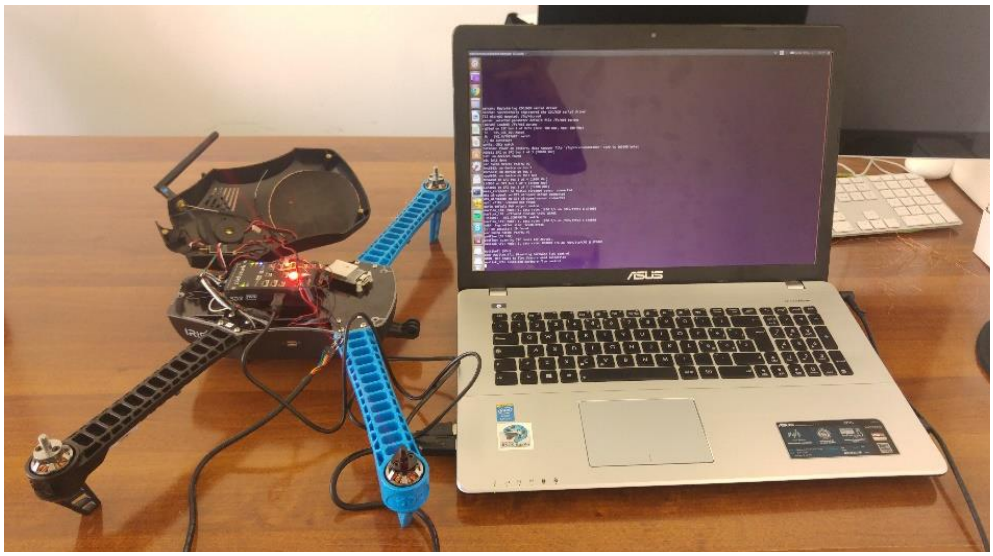


Figure 3.10: Testbed setup for deployment on Pixhawk

In this setup, the C interface of Tiny-ME has been integrated into a command-line application that can be executed within the NuttX shell on the Pixhawk. The porting process required only configuration of the build system. Remarkably, no changes to the Tiny-ME source code were necessary to compile the reasoner for the target system. Once the firmware was deployed to Pixhawk, runtime tests demonstrated the correct operation of the reasoner without any functional issues.

The ontology structure is shown in Figure 3.11. The KB models the relevant concepts and relationships needed for the detection task, including fire and explosion risk conditions based on gas and vapor concentrations.

To evaluate the feasibility and usefulness of on-board reasoning, a case study was conducted on a UAV-based system for detecting fire and explosion risks from gas or vapor. This study followed the guidelines of the European Union (EU) Directive 2014/34/UE[11], which identifies a risk if the following

[11]Directive 2014/34/UE: `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32014L0034`

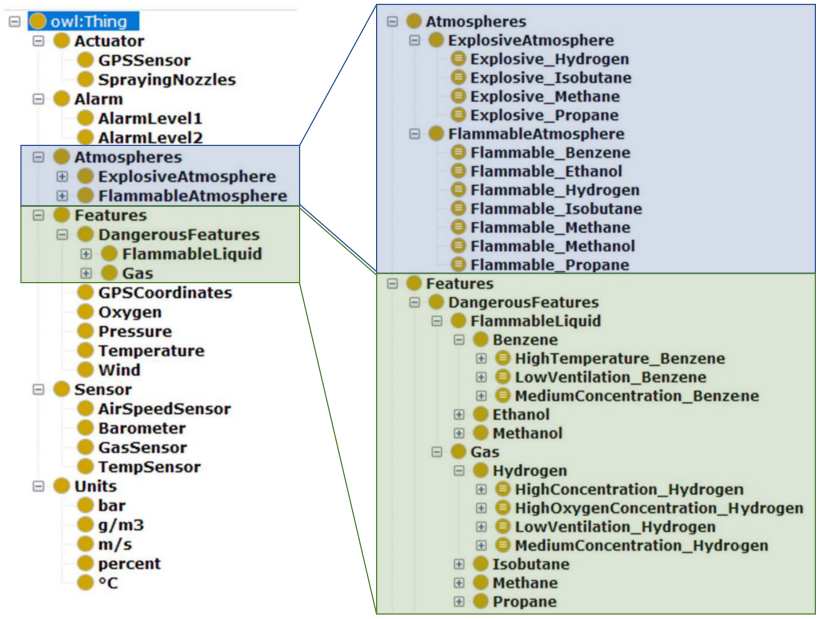


Figure 3.11: Classes in the KB modeled for the case study

conditions are met:

- The gas or vapor concentration is higher than the substance-specific *Lower Explosion Limit (LEL)* (Lower Explosive Limit), defined as the minimum concentration that can cause combustion in the presence of an ignition source.
- For a gas, the oxygen concentration is higher than the *Limiting Oxygen Concentration (LOC)* (Limiting Oxygen Concentration), which is the threshold below which combustion cannot occur.
- For a vapor, the air temperature exceeds the substance-specific *flash-point*, which is the temperature at which the substance can ignite in air.

Figure 3.11 provides an overview of the upper-level classes of the OWL $\mathcal{ALN}$ KB[12] used in the case study. The KB consists of 292 axioms, 73

[12]KB available at `http://swot.sisinflab.poliba.it/onto/drone_risk_detection`

classes, 13 object properties, and 27 individuals. Each substance monitored by the system (such as hydrogen, methane, propane, isobutane, benzene, ethanol, and methanol) is represented by individuals that model its basic characteristics and associated fire and explosion risk conditions.

The UAV is equipped with a global navigation satellite system (GNSS) antenna and several sensors, including those for temperature, atmospheric pressure, wind speed, oxygen concentration, and the concentration of the substances being monitored. A periodic task was scheduled on the UAV to:

1. Collect sensor data and preprocess it on board.
2. Use Tiny-ME to create a class expression describing the relevant parameters for each monitored substance.
3. Perform semantic matchmaking with respect to the KB individuals that represent risk conditions.

Essentially, the main thread gathers data from the sensors, which are preprocessed on the basis of predefined thresholds and filters. For each data point, Tiny-ME is exploited to instantiate the annotation that refers to the relevant class in the ontology. Subsequently, a conjunctive expression is composed, and reasoning is invoked to perform semantic matchmaking. This allows the system to identify risk conditions in real-time, ensuring timely alerts in hazardous scenarios.

In Figure 3.11, classes of explosive or flammable atmosphere conditions for the considered substances are highlighted in blue, while environmental features which influence risk levels are highlighted in green. For example, let us consider a 6 g/m^3 methane concentration, 1 m/s wind speed, and 15% oxygen concentration: based on class definitions in the ontology as derived from the EU directive, this produces the following annotation (reported in Manchester syntax [46]):

R: `MediumConcentration_Methane` `and` `HighOxygenConcentration_Methane` `and` `LowVentilation_Methane`.

As previously mentioned, by utilizing a set of predetermined thresholds, Tiny-ME creates a conjunct for each monitored substance, each corresponding to an environmental feature class within the ontology. Consequently, the classes shown in R are described within the ontology as follows:

```
MediumConcentration_Methane ≡ Methane and (hasConcentration only g/m3)
and (hasConcentration min 6 owl:Thing) and (hasConcentration max 11 owl:Thing).

HighOxygenConcentration_Methane ≡ Methane and (withOxygenConcentration
some owl:Thing) and (withOxygenConcentration only ((hasOxygenConcentration
only percent) and (hasOxygenConcentration min 14 owl:Thing))).

LowVentilation_Methane ≡ Methane and (withWindSpeed some owl:Thing) and
(withWindSpeed only ((hasSpeed only m/s) and (hasSpeed max1 owl:Thing))).
```

As it can be observed, Tiny-ME does not handle datatype properties because it relies on the $\mathcal{ALN}$ DL. Consequently, the meaning of numerical data properties is approximated by unqualified number restrictions, which are processed in reasoning algorithms as outlined in Section 3.2.1. Following this semantic labeling, the UAV matches R with every KB individual that belongs to the *Explosive_Atmosphere* and *Flammable_Atmosphere* subclasses, such as *Explosive_methane* and *Flammable_methane*. For each relevant substance, this phase helps determine if the conditions for fire or explosion are satisfied, based on the monitored and modeled environmental parameters in R.

For each individual, first of all the UAV uses Tiny-ME to check if it is consistent with R, *i.e.*, if their conjunction is satisfiable. If the check fails, then surely the current situation does not pose a risk. Otherwise, Concept Abduction is invoked to compute the semantic distance d between the monitored situation and the risk model represented by the specific individual.

It is useful to recall that each KB individual is considered as a request in the matchmaking framework described in Section 3.2.1, while R is treated as a resource. This entails that the semantic distance d represents "how much"

is missing from R to fully satisfy risk conditions. The threshold $d = 4.0$ has been determined for the case study, based on the algorithm for penalty computation in Section 3.2.1 and the adopted KB modeling patterns. Below this threshold, a risk is recognized and the UAV issues an alert.

In particular, explosion risk has been tested for all substances before fire risk, as the former requires raising a higher-severity alert. Following up the above example, the KB contains `Explosive_methane` and `Flammable_methane` risk profiles for methane. The compatibility check between R and *Explosive_methane* fails, as class `HighConcentration_Methane` is defined as having a LEL of 12 g/m^3, while `MediumConcentration_Methane` has a value between 6 g/m^3 and 11 g/m^3. The corresponding classes are described within the ontology as follows:

```
Explosive_methane: HighConcentration_Methane and HighOxygenConcentration_Methane and LowVentilation_Methane
```

```
Flammable_methane: LowVentilation_Methane and MediumConcentration_Methane and HighOxygenConcentration_Methane
```

```
HighConcentration_Methane ≡ Methane and (hasConcentration only g/m3) and (hasConcentration min 12 owl:Thing)
```

The descriptions of the two classes have number restrictions with disjoint intervals on the same property, therefore they are disjoint; this implies that explosion risk is lacking. Conversely, *Flammable_methane* $\sqsubseteq R$ and CA detects a full match ($d = 0$), *i.e.*, semantic distance below the given threshold. As a consequence, the UAV will raise a fire alert related to the methane substance.

Loading the case study KB onto the Pixhawk 1 required addressing the device's strict memory limitations. To achieve this, two key actions were taken: (i) non-essential modules, such as debugging and audio control, were disabled from the default FMUv2 firmware configuration, and (ii) the KB was encoded into the compact *Protocol Buffers* binary format[13] using the

[13]Protocol buffers home: `https://developers.google.com/protocol-buffers`

nanopb library[14]. This approach reduced the size of the KB significantly, from approximately 85 kB (in OWL 2 functional syntax) to a mere 2.1 kB, while also freeing up memory by eliminating the need for the Cowl parser.

Early quantitative tests were performed, with each test repeated five times to ensure reliability. Average results are presented in Table 3.2, and several important metrics were evaluated.

- The KB loading time averaged 117.2 ± 7.22 ms, which demonstrates a relatively fast initialization, especially given the compact nature of the KB encoding.
- The annotation and matchmaking time for a single substance was measured at 10.2 ± 1.12 ms, showing that the system can quickly process and match data from the KB.
- For a more demanding scenario involving the annotation and matchmaking of all eight substances in the KB, the time increased to 55.6 ± 2.58 ms, which is still within a reasonable range for a real-time embedded system.
- Memory usage after loading both the reasoner and the KB was 151.5 ± 0.21 kB, a value that underscores the efficiency of the memory optimization techniques employed.
- At the end of one detection loop iteration, memory usage increased slightly to 176.0 ± 0.25 kB. This shows a modest memory overhead introduced by the runtime operations, indicating that the system maintains a low memory

Overall, these results (as shown in Table 3.2) demonstrate that the proposed tool is computationally sustainable even on a highly resource-constrained device like the Pixhawk 1. The low memory usage and quick execution times suggest that it can operate effectively in typical ubiquitous computing scenarios, making it a suitable candidate for SWoT environments.

[14]Nanopb repository: `https://github.com/nanopb/nanopb`

Time Metric	Result
KB loading time	$117.2 \pm 7.22\ ms$
Annotation and matchmaking time (1 substance)	$10.2 \pm 1.12\ ms$
Annotation and matchmaking time (8 substances)	$55.6 \pm 2.58\ ms$

Memory Metric	Result
Memory usage (after reasoner and KB loading)	$151.5 \pm 0.21\ kB$
Memory usage (end of one detection loop iteration)	$176.0 \pm 0.25\ kB$

Table 3.2: Quantitative results of the KB case study testing on Pixhawk 1.

3.3 Information plane: automated Knowledge Base generation

The SWoE vision aims to bring interoperable technologies for knowledge representation and reasoning to all scales of computing, from the WWW to nanodevices with strict processing, memory, and energy constraints. Distributed infrastructures for data stream gathering, analysis and interpretation can greatly benefit from knowledge-based methods and techniques, as commonly adopted ML methods have optimal performance only for very large and well-curated datasets.

Dataset transformation into a usable KB or KG requires overcoming the *conceptual gap* of the RDF[121] and OWL[103] data models with respect to most data sources [89]. Data governance and curation are complex problems for the majority of real-world infrastructures [134] based on wireless sensor networks, embedded devices and wearables, which generate high-volume and high-velocity streams of noisy and uncertain data.

When starting from raw observation data sets, the currently prevalent approaches to the generation of a KB or KG rely on cloud-based *Data Lakes*. Though flexible, this approach is too complex and cumbersome for SWoE scenarios, not only because it requires huge storage resources, but also because it prevents agents running on pervasive devices from discovering and

detecting relevant information autonomously. A wide body of research concerns mapping tabular data to KBs and KGs, which can cover many practical use cases: several methods, systems, and evaluation benchmarks exist [74]. They allow flexible mapping definitions beyond the simple "entity-per-row" assumption, in order to meet the requirements of advanced data management applications. For this reason, they often exhibit a steep learning curve and require significant expertise with Semantic Web technologies.

In many SWoE contexts, however, observations represent events gathered from heterogeneous independent sources, such as sensors, IoT devices and objects populating a smart environment. Every event –either periodic or triggered by a condition– is an individual entity, with a specific value for each one of its attributes. As data models are kept relatively simple and regular, these contexts can benefit from leaner data processing frameworks for KB/ KG construction.

The RDF Mapping Language (RML) [33] has extended the *R2RML* [28] W3C recommendation for a declarative mapping language from tabular data to RDF. RML adds support for the integration of multiple data sources and for various structured data formats in addition to relational databases. Relational database-to-RDF (R2RML) and RML are still among the most widely adopted approaches for KG construction. RML has been further extended by *RML-star* [30] in order to support the *RDF-star* [122] language for annotating RDF statements with other RDF statements. These approaches, however, require writing a configuration document manually to specify mapping definitions. *Morph-KGC* [5] is an R2RML and RML interpreter focusing on performance and scalability: it groups rules in the input mapping documents, in order to guarantee the generation of disjoint sets of RDF triples by each group.

Several systems use popular Linked Open Data (LOD) sources to annotate tabular data automatically. The architecture of *JenTab* [2] consists in a pool of modular processing tasks, which are combined in different pipelines for each type of table semantic annotation problem; it uses *Wikidata* [141] for entity resolution. In addition to Wikidata, *DAGOBAH* [75] queries four

other services, including DBpedia [66] and Wikipedia API. Machine learning combined with probabilistic [143] and constraint programming [29] methods have also been exploited to assign a semantic model to tabular data sources automatically.

Whereas the above approaches produce RDF output, systems targeting OWL include *Mapping Master* [97] and *BootOX* [56]. Mapping Master provides a language to define mappings of complex spreadsheets to OWL ontologies. BootOX interprets R2RML mappings by encoding relational database features to OWL 2 axioms: the three phases of the *bootstrapping* problem as formulated in BootOX –vocabulary and ontology generation, mapping generation, importing– are conceptually similar to the processing phases of the approach presented in this paper, although the latter focuses on data-driven applications.

This section introduces a framework for automatic generation of OWL 2 KBs from observation data sets. The method consists in three phases: (i) data preparation and modeling of an upper ontology including classes for each type of observation (*i.e.*, relevant event in the problem domain) and for each feature; (ii) automatic TBox generation; (iii) automatic Assertion Box (ABox) generation, creating an OWL individual for each observation instance. By keeping logical expressiveness relatively simple, the KB generation approach is amenable to event classification and detection problems based on semantic matchmaking, [119] which can be executed on pervasive devices by means of optimized reasoning engines [115]. Preliminary performance tests have evaluated the sustainability of the proposal. Section 3.3.1 describes in detail the framework architecture and processing steps, while Section 3.3.2 reports on performance results.

3.3.1 Framework architecture and processing steps

The main goal of the proposed framework is to automate the construction of a KB starting from a reference dataset consisting of observation data. This process plays a critical role in the *Cyber* layer of the 5C CPS architecture,

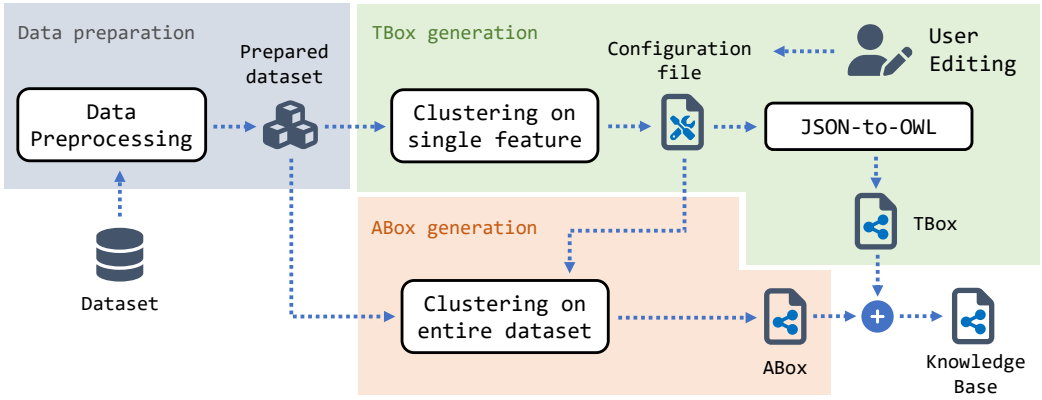


Figure 3.12: Workflow of the proposed framework

where data acquired at the *Connection* and *Conversion* levels is transformed into actionable knowledge using ML and AI techniques. The *Cyber* layer enables the extraction of insights and prediction of future trends by employing advanced analytics on filtered and aggregated data.

In this way, a dataset can be easily mapped into a conceptual model related to a specific domain, such as UAVs or Industrial IoT, where observations from physical devices can be annotated and used for decision-making. Additionally, this process can be extended to support semantic-enhanced smart contracts in DLTs. Smart contracts, when integrated with semantic reasoning, can facilitate more autonomous, trustworthy decision-making, enabling secure and automated execution of agreements based on data-driven knowledge.

As shown in Figure 3.12, the process of KB generation consists of three sub-tasks: (a) data preparation and upper ontology modeling; (b) TBox generation; and (c) ABox generation. This enables seamless knowledge generation and system adaptation in various CPS scenarios, improving efficiency, adaptability, and reactivity in real-time systems and extending to smart contract validation in DLT-based environments.

The first step towards the definition of the KB is the dataset analysis, aiming to identify the following information:

- List of *features* of observations, denoted as $\mathcal{F} = \{F_1, F_2, \ldots, F_N\}$; each feature has a (numerical or categorical) domain.
- List of *tuples*, denoted as $\mathcal{R} = \{R_1, R_2, \ldots, R_M\}$. Each tuple R_i is a set of attribute values $\langle v_{i,1}, v_{i,2}, \ldots, v_{i,N} \rangle$ representing an observation.
- List of *events* of interest, denoted as $\mathcal{E} = \{E_1, E_2, \ldots, E_P\}$ and representing the set of class labels associated with the observations. Each tuple R_i can be associated with one or more events.

The upper layers of the reference ontology $\mathcal{T}$ should model the domain conceptualization along the specific patterns detailed hereinafter, in order to support semantic-based data annotation and interpretation. $\mathcal{T}$ is assumed as acyclic and expressed in the moderately complex $\mathcal{ALN}$ DL [7]. This is required to be compliant with nonstandard, nonmonotonic inferences provided by reasoning engines designed for SWoE applications (*e.g.*, *Tiny-ME* [123]).

For each feature in $\mathcal{F}$, $\mathcal{T}$ must include a hierarchy of concepts derived from a reference class, selected by the user typically by referring to a well-known upper ontology, forming a partonomy of the topmost concept. For example, in Figure 3.13, the *FeatureOfInterest* class defined in the *SOSA (Sensor, Observation, Sample, and Actuator)* ontology [55] is used as an upper layer for the TBox section related to the features. This sub-phase must deal with any *impedance mismatch* between the dataset model and (the available expressiveness of) the selected OWL sublanguage [105].

In this way, each dataset attribute F_i is represented by means of a class/-subclass taxonomy featuring all significant value ranges and configurations it can take in the domain of interest. The breadth of each sublevel will be determined automatically during the TBox generation task described below. In that step, each subclass $F_{i,j}$ will also be associated with contextual parameters by means of specific *OWL Annotation Properties*. Similarly, the events in $\mathcal{E}$ are modeled as subclasses of an output concept identified by the user (*e.g.*, the *Observation* class in Figure 3.13).

The next step of the framework consists in processing the prepared dataset

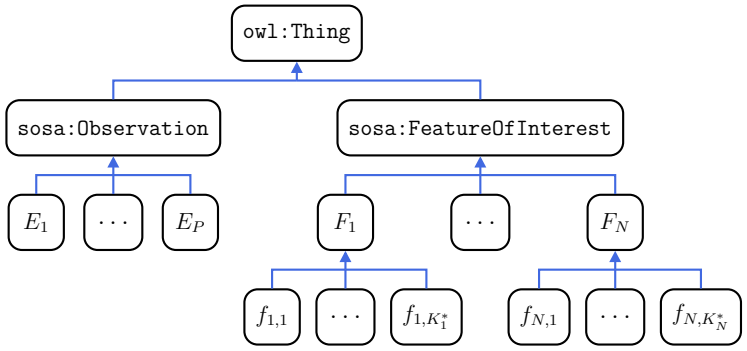


Figure 3.13: TBox hierarchy

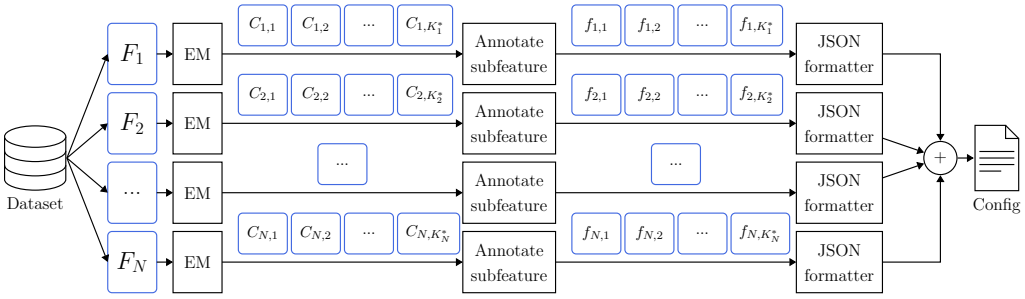


Figure 3.14: Generation of the configuration file

to generate the TBox. This task is composed of two distinct sub-tasks:

1. Generation of a configuration file, storing the ontology metadata and all features parameters required to create and characterize the concept hierarchy;

2. Serialization of the TBox, according to one of the available OWL 2 syntaxes, guided by the configuration file.

The main advantage of this two-step approach lies in the fact that the user can customize the configuration file (*e.g.*, to include further contextual information) before proceeding with the generation of the TBox. In this way, the proposed workflow can be adapted and reused for the generation of ontologies in a wide variety of domains of interest, including SCs operating over DLT-based systems.

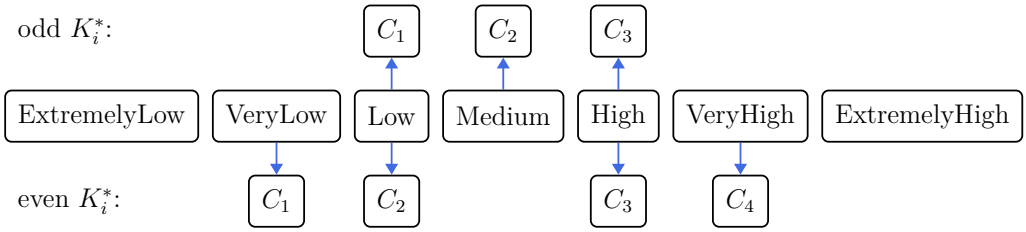


Figure 3.15: Example of prefix assignment

Given a properly prepared dataset as input, the process of building the configuration file is illustrated in Figure 3.14. The prepared dataset is processed via the *K-means* clustering algorithm, chosen for its computational suitability for SWoE contexts [4]. For each feature $F_i \in \mathcal{F}$, the Elbow Method (EM) [137] computes the optimal number K_i^* of clusters to partition the values contained in the dataset for each feature.

Denoting as $\mathcal{C}_i = \{C_{i,1}, C_{i,2}, \ldots, C_{i,K_i^*}\}$ the set of clusters obtained from the feature F_i, for each cluster $C_{i,j} \in \mathcal{C}$ a new subclass $f_{i,j}$ (for $i = 1..N$ and $j = 1..K_i^*$) is added to the TBox. The name of each subclass of a given feature is obtained by combining a cluster *prefix* with the feature *name*, *e.g.*, *Low* + *Temperature* = *LowTemperature*. Up to 7 different prefixes are currently supported to partition a feature: *ExtremelyLow*, *VeryLow*, *Low*, *Medium*, *High*, *VeryHigh*, *ExtremelyHigh*.

As shown in Figure 3.15, if the number of clusters K_i^* associated with F_i is odd, the prefix *Medium* will be assigned to the subclass $f_{i,m}$ with $m = (K_i^* + 1)/2$. On the contrary, if K_i^* is even then the prefix *Medium* will not be assigned to any subclass.

Each subclass $f_{i,j}$ is also characterized by the *centroid* ($c_{i,j}$) and the *range* of values associated with the $C_{i,j}$ cluster computed via K-means. The *kbg:centroid* annotation property associates the centroid to the subclass, while minimum and maximum values are specified as a pair of annotation properties — borrowed from the *Schema.org* vocabulary [43] — named *schema:minValue* and *schema:maxValue*, respectively. This facilitates associating the features of individual dataset observations to specific clusters and

Metadata	Annotation Property IRI	Short IRI	Range
Title	http://purl.org/dc/terms/title	dcterms:title	xsd:string
Description	http://purl.org/dc/terms/description	dcterms:description	xsd:string
Creator	http://purl.org/dc/terms/creator	dcterms:creator	xsd:string
Version	http://www.w3.org/2002/07/owl#versionIRI	owl:versionIRI	xsd:string
Centroid	http://sisinflab.poliba.it/kbgen/centroid	kbg:centroid	xsd:float
Min Value	http://schema.org/minValue	schema:minValue	xsd:float
Max Value	http://schema.org/maxValue	schema:maxValue	xsd:float

Table 3.3: List of supported OWL annotation properties

mapping each of them to the related subclass.

The final configuration is created by including the definition of all computed OWL classes and their annotated values. Basic ontology metadata (detailed in Table Table 3.3) is also specified by the user to further characterize the reference KB. Finally, the configuration file is generated according to JSON syntax [20], representing a self-describing and easy-to-parse data format.

In the second step, domain experts can modify the configuration file both to refine results obtained by the clustering procedure and introduce additional elements (*e.g.*, classes, properties) not included in the original dataset. The refined configuration is parsed by a dedicated software module implemented in Java in order to generate the corresponding ontology. The OWL API (version 3.4.10) library [45] is used as a reference implementation providing several functionalities for creating, manipulating, and serializing OWL ontologies.

The ABox generation process is shown in Figure 3.16. The same dataset used to generate the TBox is annotated to generate OWL individuals. In particular, an instance of the ABox can represent:

- A single tuple of the dataset whose values are mapped to elements of the TBox. In this way, a generic data corpus can be translated to an OWL KB, where each record corresponds to an instance;
- An aggregate event description derived from a clustering procedure con-

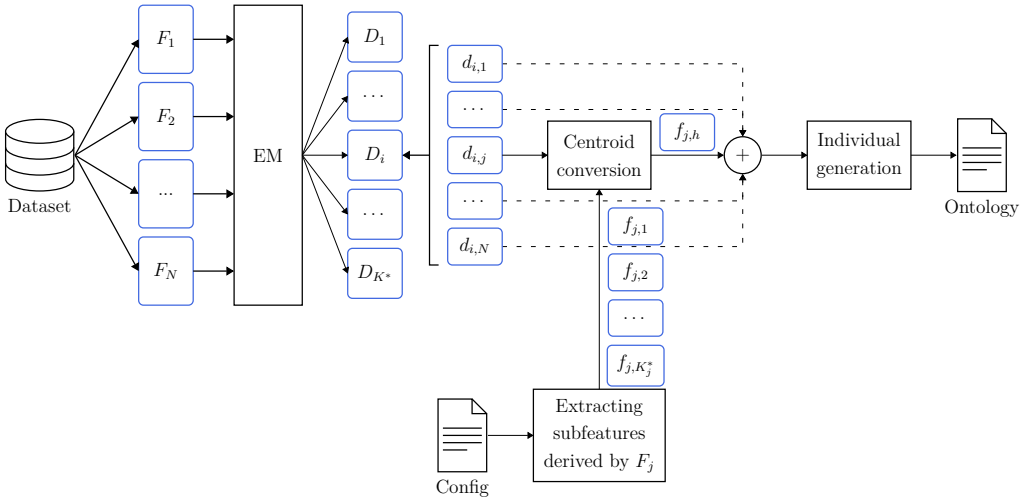


Figure 3.16: ABox generation process

sidering all attributes simultaneously rather than individual features.

In the latter case, K-means algorithm takes the whole dataset in input, in order to identify different partitions of collected observations into K clusters. Each cluster represents an OWL individual to be modeled within the KB with a reference class expression, which can be used to label new observations. The EM is used also in this case for defining the optimal number K^* of clusters to partition the whole dataset. After mapping all values $d_{i,j} \in D_i$ into OWL classes defined in the TBox, an OWL named individual will be created as a conjunctive expression of ontology axioms.

3.3.2 Performance evaluation

As a preliminary feasibility assessment of the proposed approach, computational performance has been evaluated exploiting a reference dataset [25] consisting of 10000 observations, each characterized by 48 features (specifically, 22 Boolean, 3 continuous numeric, 16 discrete numeric and 7 categorial features). Tests have been carried out on a desktop PC equipped with Intel Core i7-4790 quad-core CPU at 3.6 GHz, 16 GB DDR3 RAM at 1600 MT/s, 2 TB SATA storage memory at 7200 RPM, Windows 10 Home 64-bit.

Performed tests regard the turnaround time of the KB generation procedure, which is composed of the following sub-tasks: (a) dataset preprocessing; (b) features clustering; (c) configuration file creation; (d) TBox generation; (e) dataset clustering; (f) ABox generation (atomic concepts conjunction); (g) ABox generation (atomic concepts with object properties).

Tests have been repeated five times and average values have been reported. As shown in Figure 3.17, steps (b) and (e) require on average a higher processing time due to the clustering procedures.

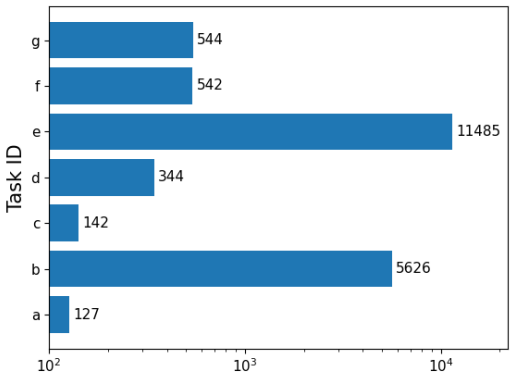


Figure 3.17: Processing time (ms)

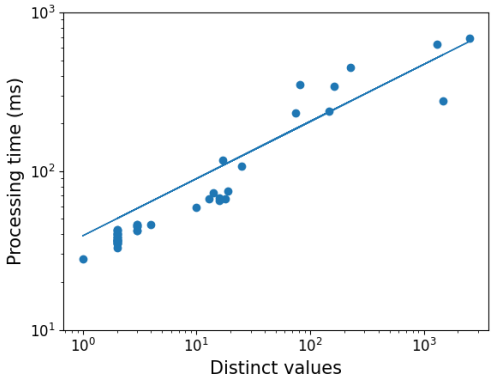


Figure 3.18: Clustering time - Task *b*

In particular, the execution time of task (b) strongly correlates with on the number of distinct values collected for each feature (Figure 3.18). The two modeling approaches (f) and (g) for the generation of the ABox require similar processing time. Users can select the most suitable approach according to the tasks that will be performed on the generated output without worrying about particular performance issues.

Chapter 4

Cloud-to-thing Cyber-Physical System infrastructure for Federated Learning

The rapid expansion of real-time data streams generated by IoT and CPS is placing unprecedented demands on centralized cloud-based data collection and processing systems, leading to challenges in scalability, availability, and performance. The *edge computing* paradigm enables a shift toward processing data closer to its source, allowing for faster data analysis, reduced latency, and improved bandwidth efficiency, all while enhancing data privacy and resilience to infrastructure failures and security vulnerabilities.

By combining edge and cloud computing, a flexible data management infrastructure is created, allowing data to be processed dynamically based on the applications' needs, network status, and resource availability. Research indicates that a majority of organizations are implementing hybrid edge-cloud strategies [77], with recent advancements like *OC* [142] establishing a seamless *cloud-to-edge continuum* [151]. In this continuum, application components, such as microservices, can migrate dynamically across edge and cloud layers [81] to balance efficiency, latency, and data privacy.

ML and AI applications are leading use cases for cloud-to-edge architectures, traditionally with centralized model training in the cloud and decentralized inference at the edge. However, with the rising computational

capabilities of edge devices, model training and inference are now possible across both layers, utilizing low-power coprocessors at the edge to optimize latency and power consumption [150, 81]. In particular, this capability enables *FL* [152], where edge nodes collaborate to solve ML problems using locally generated data, under the coordination of a cloud-based aggregator.

As cloud-based and edge computing paradigms advance, *serverless computing* [128] represents the next step, with stateless functions, or *lambdas*, providing a scalable, cost-effective, and fault-tolerant approach. Serverless frameworks abstract infrastructure management, enabling dynamic, on-demand allocation of functions based on application needs. This elasticity is particularly valuable for large-scale CPS and IoT deployments, minimizing resource overhead while maximizing availability and response time.

Building on these paradigms, the growth in IoT devices capable of executing higher-level programming languages opens new possibilities for extending the cloud-to-edge continuum toward the periphery of the network, directly at data-generation points. Distributing computation this extensively reduces latency and bandwidth requirements, offering a resilient, scalable model that is well-suited for ML and AI applications, where data streams from multiple sources need to be filtered, analyzed, and aggregated. By utilizing a cloud-to-edge-to-thing approach, serverless functions can process workloads across all three layers based on factors like availability, data privacy, and latency.

The *SWoE* [123] concept further envisions advanced ML and AI capabilities on IoT and embedded devices, yet comprehensive frameworks that unify these paradigms into a seamless *cloud-to-thing continuum* [83] are still emerging.

In response to this need, this thesis proposes a novel semantic-based DLT framework for extending the cloud-to-edge continuum into IoT ecosystems. The main contributions are summarized as follows:

- The architecture seamlessly integrates cloud, edge, and IoT layers, providing a transparent, cohesive system that enables nodes to process

data locally or send it to cloud endpoints as required. Leveraging DLT as a decentralized trust layer, this framework guarantees data integrity and security across the continuum. Through semantic annotations, data and interactions are enriched, allowing nodes to make context-aware decisions while maintaining consistency across the network.

- To validate the framework, a federated learning scenario was adopted, demonstrating how semantic-enhanced data aggregation and local ML training can be achieved across IoT devices and edge nodes, with the cloud acting as a coordinator. This setup allows for resource-constrained devices to conduct computations that would traditionally require centralized processing, thus preserving data privacy by reducing the need for data transmission to the cloud.

- The architectural design emphasizes data privacy and security, critical aspects in sensitive IoT applications such as healthcare or vehicular networks. Security measures are implemented via robust authentication and encrypted communications, while privacy is preserved through federated learning techniques, which allow IoT devices to train ML models locally or at the edge, without needing to transmit sensitive data over the Internet.

- A prototype was developed using Commercial Of-The-Shelf (COTS) technologies, including Amazon Web Services (AWS) for cloud infrastructure and serverless function deployment, with edge nodes implemented on Raspberry Pi and STM-32 microcontrollers. This prototype confirms the system's portability and feasibility, illustrating the ease of implementation across various devices and environments.

To showcase the framework's capabilities, a case study was conducted on the *MotionSense* dataset [84], focused on activity recognition using data from wearable devices. The federated learning approach was tested across cloud, edge, and IoT layers, demonstrating efficient activity recognition with minimal latency, while preserving data privacy and reducing communication overhead.

The proposed cloud-to-edge-to-thing federated learning framework holds wide-ranging applications. In telemedicine, wearable devices can perform local data processing for individual users and share anonymized data summaries with edge or cloud layers, enabling timely, accurate health monitoring. In industrial IoT scenarios, the framework supports predictive maintenance and quality control across multiple production facilities. Furthermore, in the *Internet of Drones* [3], individual drones can process ML and AI tasks locally, sharing only high-level model data with peers and ground control, thereby optimizing latency, bandwidth, and energy consumption for missions such as environmental monitoring and precision agriculture.

The remaining chapters of this dissertation are organized as follows: Section 4.1 provides background on the chosen ML algorithm and relevant literature; Section 4.2 outlines the proposed framework's architecture and workflow; Section 4.3 presents a case study for activity recognition using federated learning; and Section 4.4 discusses early experiments, followed by conclusions.

4.1 Background

This section briefly recalls the reference ML algorithm which is exploited in a novel way to enable federated learning in the cloud-to-thing continuum. Relevant related work is also discussed.

4.1.1 Matchmaking Features for (federated) mAchine Learning Data Analysis

This work leverages *MAtchmaking Features for mAchine Learning Data Analysis (MAFALDA)* [119], an IoT-oriented semantic-enhanced ML algorithm. MAFALDA supports the typical ML pipeline for classification problems: data collection and preparation, feature extraction and selection, model training and refinement with hyperparameter optimization, model evaluation and de-

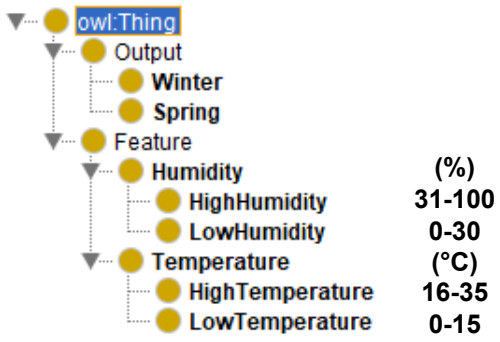


Figure 4.1: Example ontology

ployment to generate predictions. Nevertheless, the semantic-enhanced approach changes the way each step is performed w.r.t. purely stochastic methods. The proposed workflow of MAFALDA for FL is depicted in Figure 4.2 and described in detail in what follows.

Data Collection (DC) and **Data Modeling (DM).** After training set gathering and cleaning, feature selection is supported by an ontology which models the domain conceptualization along properly defined patterns, in order to allow semantic-based data annotation and interpretation. The ontology is expressed in a restriction of the *glsOWL* version 2 [103], corresponding to the $\mathcal{ALN}$ DL. Specifically, for each feature f_q the ontology must include a corresponding class A_q and a partonomy of it, *i.e.*, a set of subclasses $A_{q,1}, A_{q,2}, \ldots A_{q,p_q}$ with $\forall j = 1, \ldots p_q : A_{q,j} \sqsubseteq A_i$, representing all meaningful configurations or value ranges the feature can take; p_q is different for each q as partonomies for different features are not constrained to have the same number of classes. As a minimal example, consider the ontology in Figure 4.1, which models room temperature and humidity features.

Build Training Matrix (BTM). The subsequent training step builds an intermediate aggregate data structure named *Training Matrix (TM)*, which is then used to generate the *model* as a set semantic annotations, one denoting each possible output class in the training set. Training is executed on a set S of n training samples, each with (at most) m features, and w distinct output classes. Processing the i-th sample, its q-th feature value is mapped to a *concept component* $C_{i,q}$ constructed over A_{q,j_q} in the reference

Table 4.1: Example training set

Temperature	**Humidity**	**Output**
17	40	Spring
21	60	Spring
3	20	Winter
8	50	Winter

Table 4.2: Example Training Matrix

HighHum.	**LowHum.**	**HighTemp.**	**LowTemp.**	**O**
1	1	2	0	Spring
1	1	0	2	Winter

ontology, which is the subclass in the partonomy for A_q containing the particular feature value. For the sake of conciseness, the definition of concept components and details for constructing them in [119] are not recalled here. Overall, the i-th sample $\forall\ i = 1, \ldots, n$ is composed of: (a) up to m *concept components* $C_{i,1}, \ldots, C_{i,m}$ annotating its features; (b) an observed output O_i labeled with a class name in the ontology. Samples are processed sequentially in order to build the TM, which is a $(w+1) \times (k+1)$ matrix. All the different output classes are in the first column while the k distinct concept components occurring in the training set are in the first row. Each element of the TM represents the number of occurrences of the column header concept component in the samples having the row header output. Basically, the construction algorithm (detailed in [119]) takes the i-th training sample and first checks its output class O_i: if no previous sample has been associated to that class, it appends a row to the TM setting its values to zeros. Analogously, for each concept component $C_{i,j}$, if no previous sample includes it, then the algorithm appends a column to the TM and sets its values to zeros. Finally, the algorithm increases by 1 the value of the cell corresponding to O_i and $C_{i,j}$. Continuing the above example, suppose two output classes exist, named *Spring* and *Winter*, and the training dataset contains the four samples shown in Table 4.1. Then the TM is computed as reported in Table 4.2.

Build OWL Model (BOM). Once the TM is built, the model can be

generated to be used for prediction tasks by means of semantic matchmaking, as explained in [119]. Basically the model consists in an $\mathcal{ALN}$ expression E_i for each output class O_i, given from the logical conjunction of concept components $C_{i,j}$ appearing in the TM row of O_i. In order to improve model accuracy, MAFALDAdefines a set of dynamic thresholds over each row and each column of the TM in order to exclude from the expression the concept components which occur too infrequently [119]. These thresholds are the subject of hyperparameter optimization for model refinement, in which typical techniques can be applied [149]. Anyway, the final model is just a set of high-level formal OWL 2 expressions, summarizing even large data sets in a compact and meaningful way. In the running example, supposing for the sake of simplicity a fixed threshold $\theta = 0.7$ is adopted for all rows and columns of the TM, the *HighHumidity* and *LowHumidity* concept components are discarded for both output classes, as they have a frequency of $\frac{1}{1+1} = 0.5 < \theta$. Therefore the final trained model consists in the following pair of OWL class expressions:

- $Spring \equiv HighTemperature$
- $Winter \equiv LowTemperature$

Aggregation for federated learning. As recalled above, the TM is an intermediate structure which summarizes –and anonymizes– input data exploiting a reference domain ontology. This work leverages one of its fundamental properties: if a training set is partitioned in two or more subsets and the corresponding TMs are generated, they can be aggregated simply by summing the values in cells corresponding to the same concept component and the same output class, *e.g.*, in our running example, the cells for *Spring* row and *LowHumidity* column can be summed across multiple TMs. The result will be identical to the generation of a single TM from the whole training set. Based on this property, the algorithm supports federated learning by aggregating TMs computed locally by different nodes and summing them, without exchanging training data, as pictured in Figure 4.2. Additionally, the fact that the trained model consists in a set of OWL 2 individuals for the various

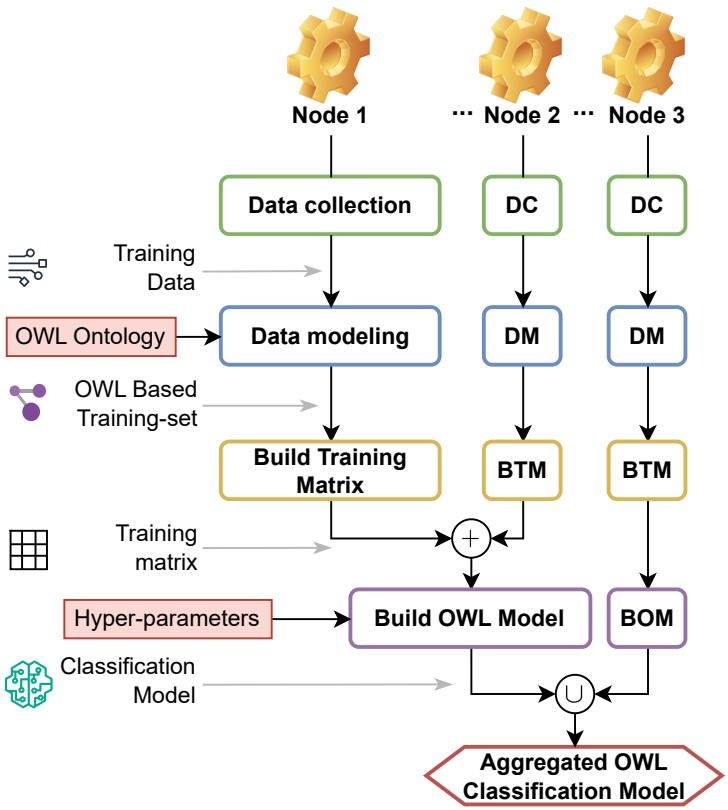


Figure 4.2: MAFALDA workflow for federated learning

output classes allows a second (optional) level of aggregation in federated learning, basically given by the conjunction of sets of individuals generated by independent nodes but referring to the same domain ontology. The latter method, also shown in Figure 4.2, grants even smaller data exchanges, by establishing a unified knowledge base from the collective knowledge of the various nodes.

Overall, the adopted MAFALDA algorithm enables:

- *incremental learning*, in which training samples can be processed in batches: for each batch the same TM is updated, and at the end of each batch a new model is generated from the TM;
- *federated learning*, where individual nodes process different yet homogeneous data sets (*i.e.*, referring to the same ontology) and construct

independent TMs. Each TM is then used in two ways: (i) define a model for carrying out predictions locally; (ii) send the TM to an upper-level aggregator, which will combine multiple TMs into a single one and generate a (presumably more accurate) model. This approach can be applied recursively with two or more levels of aggregation, and both the TM and the model generated at an upper level can be fed back to lower-level nodes, in a continuous loop aimed to improve performance as well as follow possible drifts and long-term dynamics of the monitored phenomenons.

While the work in [119] hinted at the possibility of federated learning for MAFALDA but did not formalize it, this work exploits the original algorithm by constructing on top of it a serverless federated learning framework, which expands the cloud-to-edge continuum to include IoT field devices as computational nodes capable of running both training and inference tasks.

4.1.2 Serverless computing in the cloud-to-edge continuum

Cloud-edge [22] computing has emerged as an innovative paradigm aiming to address the diverse and evolving demands of modern applications and services. The *cloud-to-edge continuum* [15] framework extends from localized edge devices to centralized cloud infrastructures, enabling seamless data and task flow across the cloud-edge interface. For instance, [140] describes a unified resource orchestration strategy for effectively managing cloud-edge resources, treating them as a single abstracted entity for executing distributed services. At the network edge, nodes process data in real-time to minimize latency and network bandwidth usage. Conversely, centralized cloud infrastructures offer extensive computational power and storage for demanding tasks. Bridging these extremes into a continuum enhances performance, by tailoring architectural designs to the specific requirements of different applications, distributing data and processing tasks across the network.

Recent research has focused on the integration of edge and cloud technologies in several areas like cyber-physical systems [23, 81], healthcare [132] and intelligent transportation systems [6]. Nevertheless, cloud-edge computing architectures pose several challenges: (i) reducing *latency* is a primary goal, but achieving ultra-low latency between edge and cloud in real-world scenarios can be challenging due to varying network conditions and processing demands; (ii) as the number of edge devices and applications increases, managing *scalability* becomes a significant challenge, as edge infrastructures must be able to handle the dynamic growth in data processing requirements while maintaining performance and reliability; (iii) edge devices generate large volumes of information requiring an efficient *data management*, *i.e.*, deciding what data to process locally, what to transmit to the cloud, and how to store, retrieve, synchronize and orchestrate data effectively.

Combining compute continuum architectures with serverless frameworks represents one of the more promising research areas. Compared with traditional cloud computing approaches, *serverless computing* [128] aims to create dynamic environments where both infrastructure and platforms in which the services are running are hidden from customers. In this way, users of cloud services can invoke the desired functionality of their application only paying for the resources they actually use. The invocation of the functions is delegated to one of the available computation nodes (*e.g.*, cloud containers, decentralized edge environments or specific IoT devices) and the obtained results are sent back to the user. The fusion of serverless architecture with the cloud-to-edge continuum holds a great potential in the field of IoT-based federated learning scenarios [70]. By deploying serverless functions strategically across the continuum, edge devices can perform initial model training, leveraging their proximity to data sources. On the contrary, resource-intensive tasks like model aggregation and global updates can be executed in the cloud. This approach optimizes the federated learning workflow, while reducing the burden on individual edge devices and ensuring low-latency data processing thanks to the virtually unlimited computational resources available in the cloud.

4.1.3 IoT-oriented federated machine learning frameworks

Emerging application scenarios based on the cloud-to-edge continuum are increasingly relevant in the field of distributed intelligence, driven by the rapid expansion of IoT infrastructures and the need for low-latency, fault-tolerant, and secure processing capabilities [110]. This evolution has spurred the development of advanced edge analytics services [112] that address these demands by efficiently managing computational tasks across cloud, edge, and IoT layers.

A foundational architecture for orchestrating containerized microservices and deploying cloud-edge intelligence was proposed in [81]. This model leverages OC principles [142] to support data mining with predictive ML models, trained and deployed across edge and cloud environments. By utilizing computational resources opportunistically, it aims to maximize prediction accuracy. However, while effective, this architecture does not fully integrate IoT devices for either training or inference tasks. Additionally, federated learning methods were not employed, as data was centralized for processing rather than distributed.

Further advancements in cloud-edge AI frameworks have targeted enhancements in ML efficiency, aiming to reduce transmission latency and bandwidth use. In particular, one framework [138] employs container orchestration to manage task allocation and data processing, integrating a BranchyNet Deep Neural Network (DNN) model with early-exit branches for rapid inference on edge devices. This model significantly optimizes response times and system load; however, within this architecture, IoT devices serve only as data sources, lacking the capability to independently handle training or inference.

FLoX [63], a federated learning framework, was designed to train and deploy neural network models across heterogeneous, distributed resources. Built on the *funcX* [24] federated serverless platform, it decouples FL model

training and inference from infrastructure management, allowing flexible deployment across diverse network devices. While similar in approach to the goals of this thesis, it does not support training or inference tasks on IoT devices.

Another framework, *Rural AI* [104], leverages funcX to implement a federated Function-as-a-Service (FaaS) architecture suited to rural precision agriculture. This architecture addresses the unique demands of rural environments, which are often characterized by limited and unstable network infrastructures, thus demonstrating how serverless computing can enhance traditional FL in constrained conditions.

Other notable efforts include *FedLess* [41], a serverless framework for training and deploying DNN models across heterogeneous FaaS platforms, and the Serverless Hierarchical Federated Learning (SHFL) framework [95], which organizes a two-layer FL architecture. In SHFL, nodes are grouped into clusters managed by cluster heads, which share model parameters among a localized worker network.

While these frameworks offer numerous advantages, they each present specific limitations in comparison to the approach explored in this thesis, as summarized in Table 4.3. The framework under development uniquely integrates the following key features:

- A serverless Functions-as-a-Service architecture.
- Full support for federated learning across cloud, edge, and IoT environments.
- Capabilities for AI training and inference on IoT nodes, as well as on cloud and edge.
- Implementation flexibility facilitated through commercial off-the-shelf (COTS) software tools.

This combined architecture is designed to harness the full potential of

cloud-edge-IoT synergy, extending intelligence to the very edges of the network and aligning with the demands of real-world distributed intelligence applications.

Table 4.3: Related works comparison

Reference	COTS Tools	FaaS	FL	Cloud AI	Edge AI	IoT AI
[81]	✓	✗	✗	✓	✓	✗
[138]	✓	✗	✓	✓	✓	✗
[63]	✓	✓	✓	✓	✓	✗
[41]	✓	✓	✓	✓	✓	✗
[95]	✗	✓	✓	✓	✓	✗
This proposal	✓	✓	✓	✓	✓	✓

4.2 Proposed framework

The proposed approach relies on serverless computing to define a cloud-to-thing framework for data collection, ML model training, and inference. The proposal aims to achieve three key properties:

1. **Federated learning flexibility:** the federated learning workflows extend to IoT devices as nodes for ML training and inference, harnessing the Mafalda algorithm recalled in Section 4.1.1.

2. **Unified execution:** functions can seamlessly run in the cloud, on on-premise edge devices and on IoT devices with minimal to no difference in code.

3. **User and device transparency:** the infrastructure operates transparently for users and devices. This means that nodes have the capability to collect and dispatch data to either local edge devices or the

cloud endpoint, with the response remaining consistent, regardless of the processing device.

The proposed approach relies on serverless computing to define a cloud-to-thing framework for data collection, ML model training, and inference. The proposal aims to achieve three key properties:

1. **Federated learning flexibility:** the federated learning workflows extend to IoT devices as nodes for ML training and inference, harnessing the MAFALDA algorithm recalled in Section 4.1.1.
2. **Unified execution:** functions can seamlessly run in the cloud, on on-premise edge devices and on IoT devices with minimal to no difference in code.
3. **User and device transparency:** the infrastructure operates transparently for users and devices. This means that nodes have the capability to collect and dispatch data to either local edge devices or the cloud endpoint, with the response remaining consistent, regardless of the processing device.

Further details on the proposed framework are discussed in the following sections. While description details refer to federated learning scenarios for the purpose of clarity and accuracy, the core architecture presented in Section 4.2.1 is basically general-purpose, as it can support distributed serverless functions for any type of application involving IoT, edge and cloud layers.

4.2.1 Architecture

Figure 4.3 depicts a high-level conceptual representation of the architecture, which encompasses the cloud infrastructure, edge devices and IoT field nodes. Specifically, it includes the following components:

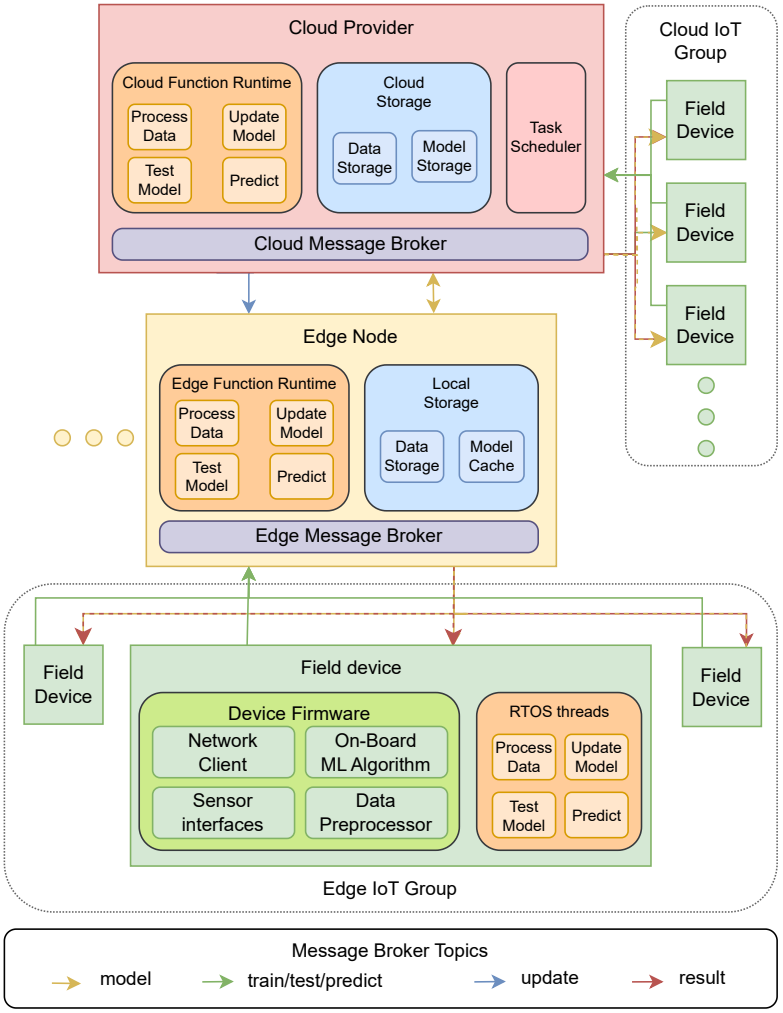


Figure 4.3: Reference framework architecture

- **Cloud Provider**: has the responsibility of efficient provisioning and elastic scaling of the underlying cloud infrastructure.
- **Edge Node**: scaled-down counterpart to traditional cloud data center, delivering computation, communication, and storage capabilities.
- **Field Device**: IoT node characterized by a lightweight RTOS as well as limited computational and energy resources.
- **Cloud/Edge IoT Group**: defines a set of field devices, connected either to the global cloud provider or to a local edge Node. Mutual

authentication and authorization mechanisms are enforced, using certificates to manage roles and permissions for publishing and reading messages on available topics.

- **Function Runtime**: executes the stateless tasks in the Cloud Provider or in the Edge Node for the federated learning workflow in an event-driven and distributed manner. In Field Devices the same functions map to RTOS threads directly, for more efficient resource usage.

- **Storage**: stores the collected data and the machine learning models used for federated learning tasks. In federated learning scenarios, data is often partitioned into smaller, manageable batches to optimize communication efficiency. In particular, **Cloud storage** holds the latest and most complete version of the federated learning model in persistent memory, whereas **Local Storage** acts as a cache: if an Edge Node must carry out a prediction task but lacks a local model, it retrieves the one from Cloud storage.

- **Message Broker**: orchestrates event-driven message transmission and reception over the *Message Queuing Telemetry Transport (MQTT)* standard protocol (`https://mqtt.org/`), adopting the publish/subscribe paradigm. Nodes publish packets to specific topics, and the message broker routes received messages to subscribed services.

- **Task Scheduler**: it can invoke serverless functions in accordance with user-configured policies or timers. The main role in the proposed architecture is to invoke the “Update Model" function periodically, in order to decouple updates to the aggregated data for federated learning (in MAFALDA’s case, TMs from the task of training a new version of the ML model. Other triggers can be configured to activate the function, such as events related to the connection/disconnection of nodes in the architecture, requests from Field Devices or Edge Nodes, or when a certain amount of data is uploaded to the Data Store.

A noteworthy feature is the capability to perform machine learning tasks not only on edge nodes but also on IoT devices, without depending on the

cloud infrastructure. This flexibility ensures that outcomes remain consistent, with variations primarily in response times. The following functions are executed in the Function Runtime:

- **Process Data**: the primary task is to read the data published by Field Devices. This function plays a pivotal role in the framework's data processing pipeline, ensuring that incoming data is efficiently archived in the Data Storage component for subsequent model updates and inference. The function is invoked for each incoming message from the `train` MQTT topic;
- **Update Model**: enhances predictive capabilities over time through incremental learning from newly available data batches. It retrieves locally stored data batches from the Data Storage, and performs an incremental model update, as described in Section 4.1.1. By adopting a *mini-batches* approach for incremental updates, the model can be trained in short, lightweight bursts of computation, complying with the execution time and memory constraints of serverless functions. The updated model are saved into the Model Storage. This function is started periodically by the Task Scheduler service.
- **Test Model**: the function responds to MQTT `test` messages containing labeled samples for prediction. It utilizes this data to calculate a confusion matrix and evaluation metrics, including precision, recall, F1-score, and overall accuracy for the most up-to-date model available on the node.
- **Predict**: responds to MQTT `predict` messages, which contain a series of unclassified samples, by delivering the classification results based on the most up-to-date model available on the node.

In order to validate the proposed serverless architecture, an off-the-shelf infrastructure based on AWS technologies has been employed for implementing all the components in a complete prototype, as shown in Figure 4.4:

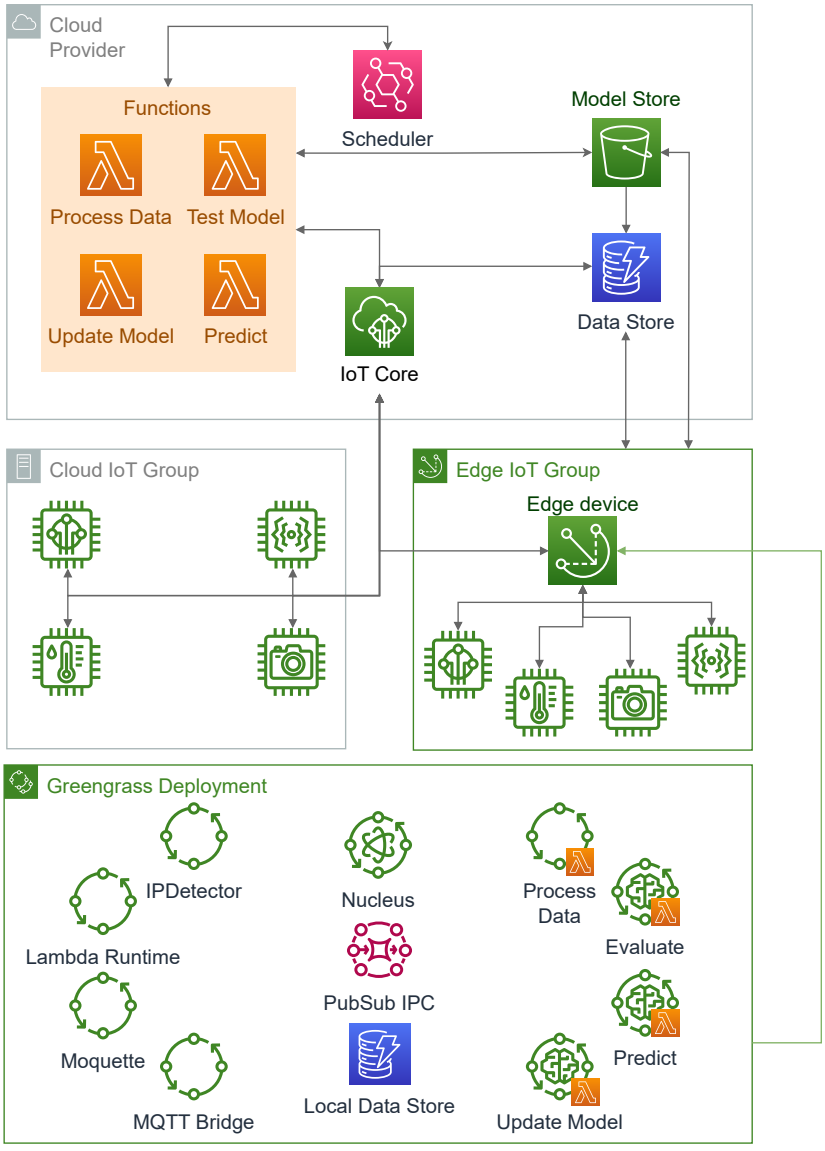


Figure 4.4: Proposed AWS-based prototype architecture

- **IoT Core**: as the central component of the architecture, the *AWS IoT Core* (`https://aws.amazon.com/it/iot-core/`) service is responsible for managing MQTT communications, device provisioning, and authentication within the network. It can interface with and manage both *on-premise* IoT device groups and edge nodes, which in turn can manage other IoT device groups.

- **Model Storage**: the *Amazon S3* (`https://aws.amazon.com/s3/`) object storage service is used to store and manage ML models. When an updated model is loaded, it replaces the previous version.

- **Data Storage**: the *DynamoDB* (`https://aws.amazon.com/dynamodb/`) scalable NoSQL key-value database handles data storage in the form of batches. A configurable batch ageing policy is established to limit the amount of relevant data stored for model training and update.

- **Lambda Functions**: a set of *AWS Lambda* [1] functions has been defined to map the aforementioned four tasks: process data, update model, test model, and predict. These lambdas are invoked and executed in the same way both in the cloud and on edge nodes. Moreover, the same source code implementing the MAFALDA algorithm is exploited to run functions locally in Field Devices on data collected from available sensors. Further implementation details are provided in Section 4.3.1.

- **Scheduler**: the *AWS EventBridge* (`https://aws.amazon.com/eventbridge/`) service enables real-time data change notifications from AWS services, personal applications, and Software as a Service (SaaS) applications without coding. In the architecture, AWS EventBridge serves as the Task Scheduler component, orchestrating and triggering lambda functions automatically, based on a pre-configured timer.

- **Edge device**: a sufficiently capable edge device, such as a single-board computer or a PC, acts as *AWS IoT Greengrass* (`https://aws.`

[1] `https://aws.amazon.com/lambda/`

`amazon.com/greengrass/`) *CoreDevice* to run Greengrass services and manage its IoT device group.

- **Greengrass Deployment**: A group of components executed on AWS CoreDevices, including:
  - **Nucleus**: manages the device's lifecycle and control communications with the cloud.
  - **PubSub IPC**: enables event-driven distributed Inter-Process Communication (IPC) among the internal components within the Greengrass node.
  - **Moquette**: a local MQTT broker allocated for the subnet.
  - **MQTT Bridge**: serves as an intermediary for MQTT messages among Moquette, PubSub IPC and AWS IoT Core, enhancing communication within the Greengrass ecosystem and connecting it to the broader AWS infrastructure.
  - **Process Data**, **Update Model**, **Test** and **Predict**: the corresponding Lambdas imported into the device that react to messages received on the local broker.
  - **Local Data Store**: managed through a local instance of DynamoDB, serves the dual purpose of storing the data batches received from the `train` topic and the updated models received from the cloud.
  - **IPDetector**: A component that manages the Cloud Discovery procedure.

Details on how these components interact in the training and testing/prediction phases of serverless federated learning are explained in the following two sections.

4.2.2 Training

This section describes the sequence of operations outlined by the training task, managed through the *Process Data* lambda function within the framework described in Section 4.2.1. Figure 4.5 sketches the sequence diagram of operations and interactions among cloud/edge/field components: numbered steps are outlined in what follows.

1. The Field Device (FD) locally performs preprocessing on sensor data within a temporal window. During this phase, the intermediate aggregated data structure is generated, *i.e.*, the TM in case of MAFALDA, or a local classification model is trained as described in Section 4.1.1.

2. The FD publishes a message serialized in binary format on the `train` MQTT topic. Depending on application-specific concerns about privacy and bandwidth availability, the federated learning framework is configured so that the message contains either the training data batches, the TM or the local classification model, as shown by the two alternative sequences in Figure 4.5.

3. The message triggers the execution of the *Process Data* function on the Edge Node or Cloud Provider managing the IoT group of the FD, which compresses and stores the received data on the relevant data/model store. The architecture does not constrain the way FDs are associated: for configuration simplicity, in the current AWS-based prototype the association is static either to an Edge Node in the local network or to the cloud, but proper dynamic criteria can be considered for future revisions.

4. Upon completion, the *Process Data* function publishes an acknowledgment message, notifying the FD of the task conclusion.

Independently from the data upload phase, there exists a model update phase, which allows updating the ML model to achieve a more accurate

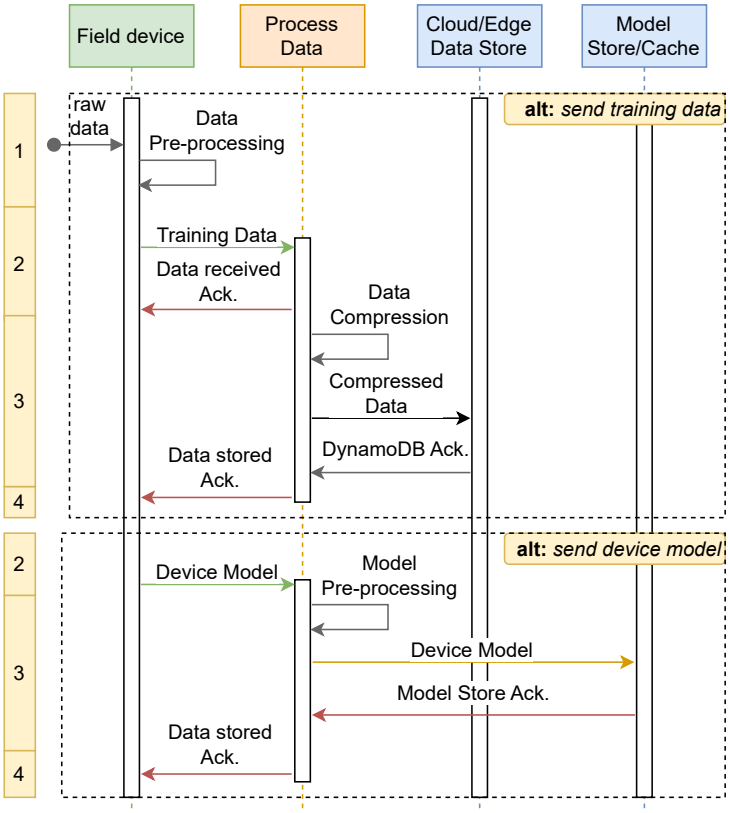


Figure 4.5: Model training sequence diagram

version based on the whole dataset available in the Data Store. Figure 4.6 illustrates the sequence diagram of the interactions among the components.

1. The invocation of the *Update Model* serverless function is managed by the *AWS EventBridge* (`https://aws.amazon.com/eventbridge/`) component, a serverless event router which periodically schedules an event to trigger the function in the cloud.

2. The function retrieves the most recent TMs and models from the Model Store (managed through an S3 instance), along with the available data from the Data Store (managed through DynamoDB). TMs sent by IoT or edge devices are also retrieved from the Model Store at that time, in order to perform TM aggregation. The aggregated TM is used to generate the overall updated model, which is aggregated with Device

Models and sent back to the Model Store. This aggregated model will be used for future prediction and training tasks.

3. Upon conclusion, cloud *Update model* function sends a message on the `update` topic.

4. On the Edge Node, the same *Update Model* function is invoked upon the arrival of a message on the `update` topic, to update the reference model with the data available on Edge Data and Model Store. Initiating simultaneous updates from multiple devices can lead to write conflicts to the Model Store. To reduce the likelihood of conflicts, the function waits for a random interval before execution. Additionally, an Optimistic Concurrency Control strategy is adopted: the model is overwritten only if the object's version number has not changed from the initial read to the moment of writing the updated model. In case of a conflict, the training must be repeated on the new version.

5. The updated model is stored in the Local Model Storage of the Edge Node, which, as previously described in Section 4.2.1, plays the role of caching objects locally to reduce frequent access to the Cloud storage. Finally, the function sends the updated model to the cloud-based Model Store.

Serverless runtimes are typically configured to execute functions for short periods of time. In AWS Lambda execution time is capped at 15 minutes. To address this limitation, it is possible to allocate larger amounts of resources to more intensive tasks such as the *Update Model* function.

4.2.3 Testing and prediction

This section provides a detailed overview of the workflow for carrying out ML inferences on distributed models, such as performing classification tasks to predict events based on sensor data connected to FDs. The sequence diagram in Figure 4.7 shows the workflow in the case of cloud execution.

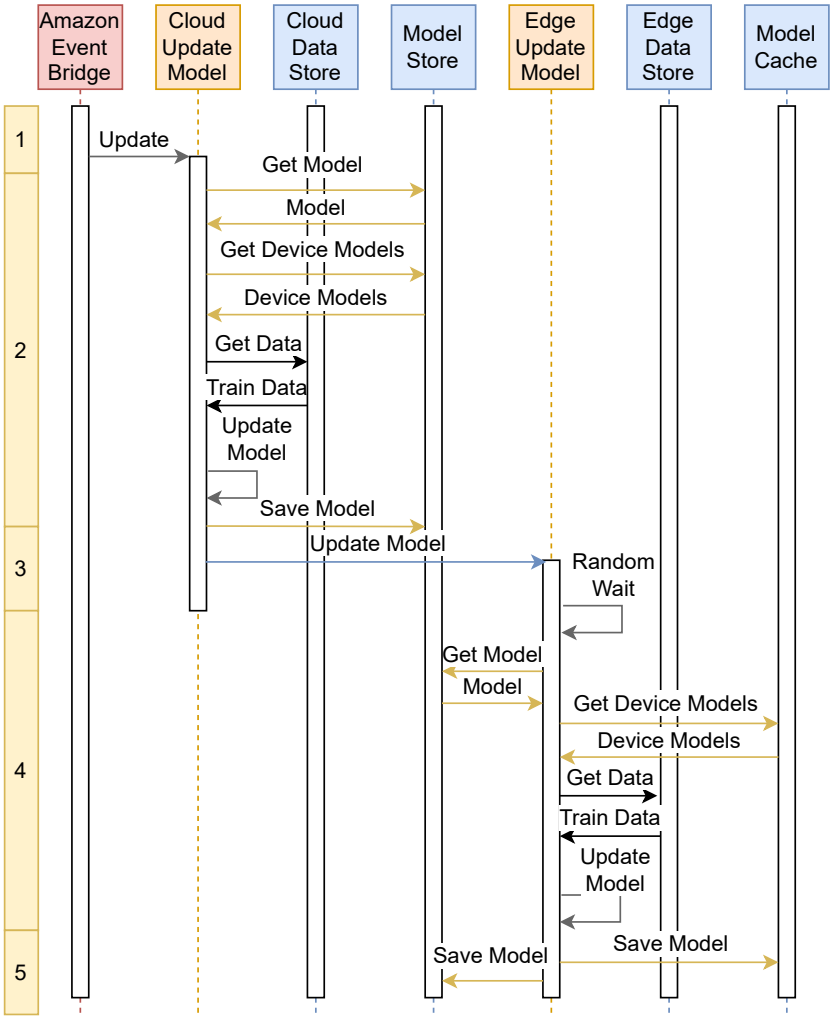


Figure 4.6: Model update sequence diagram

1. The Field Device, following data processing like in Section 4.2.2, publishes a message on the `predict` (respectively, `test`) topic.

2. The corresponding function is initiated in the cloud. The *Predict* (resp. *Test Model*) function retrieves the latest trained model instance from the Model Store, carries out classification (resp. evaluation), and dispatches the outcome as a message to the `result` topic.

When performing prediction or testing on the edge side, the sequence of operations becomes more complex, as illustrated in the sequence diagram

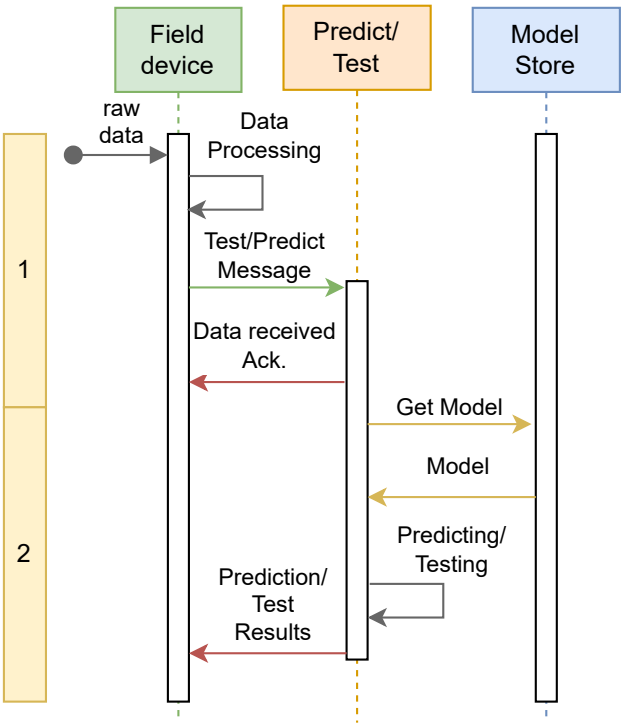


Figure 4.7: Predict and Test functions on cloud

in Figure 4.8. Specifically, upon receiving a message on the `predict` (resp. `test`) topic, the Edge Node function initiates its workflow by attempting to retrieve the training model from the Local Data Storage, which operates as a cache, as elaborated in Section 4.2.1. Two alternative execution flows are possible:

- if the model is available locally, the function directly loads it to carry out the prediction (resp. test) task, mirroring the previous scenario;
- if the model is not in the cache, the function must then retrieve it from the cloud-based Model Store. This action incurs latency and data network traffic penalties. Once obtained, the model is cached to streamline future runs, before executing the requested prediction (resp. test) and returning the results.

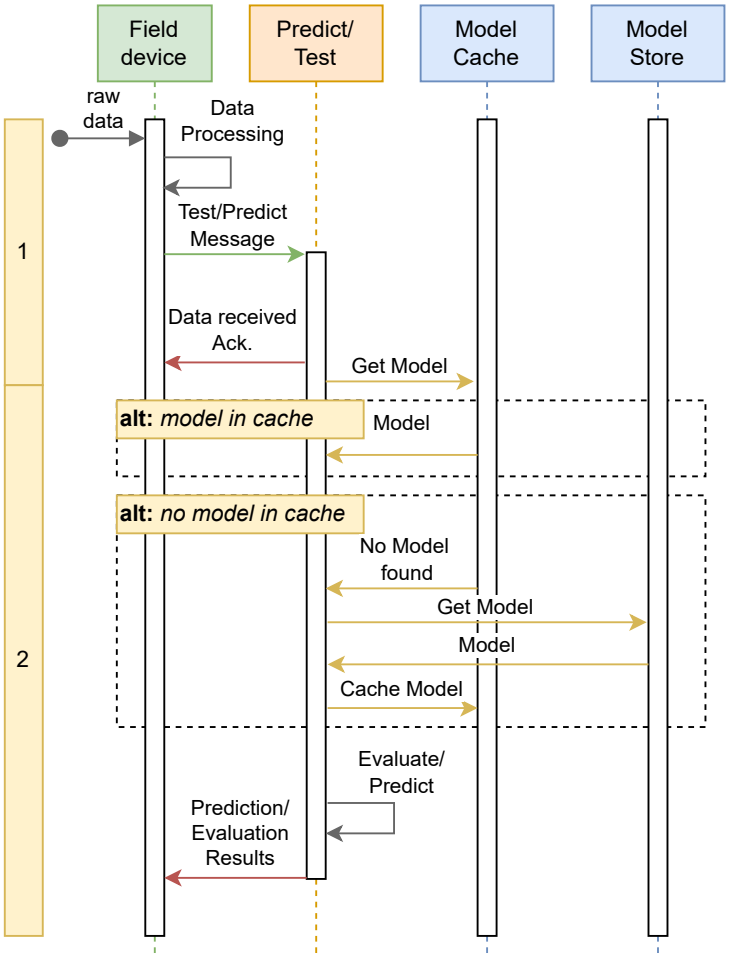


Figure 4.8: Predict and Test functions on edge

4.2.4 Portability

AWS has been selected as the technology provider for the prototypical implementation of the proposed framework, due to its comprehensive feature set, which has allowed to map directly all components of the architecture to available building blocks at the IoT, edge and cloud layers. However, it is essential to note this is not a mandatory choice. The same framework is portable to other providers by substituting the adopted managed services with equivalent offerings. The considerations below summarize some relevant examples of available offerings, but many other players exist in the server-

less computing and cloud-to-thing services markets and they are accelerating their pace of innovation, therefore new or improved off-the-shelf solutions can become available.

Microsoft Azure includes *Azure Functions* supporting multiple languages, *Azure Service Bus* for message management, *Blob Storage* and *Table Storage* for data storage, and *Azure IoT Edge* for deploying workloads on edge devices with enhanced device management capabilities. Unlike AWS Greengrass, Azure IoT Edge requires user's code to be run in containers: this can limit the choice and increase the cost of devices to be used as IoT FDs. Conversely, Azure IoT Edge automatically manages multiple-level hierarchies of devices, whereas AWS Greengrass requires the user to configure the device group hierarchy: this can be an important feature for large-scale federated learning applications. **IBM Cloud** services include *IBM Cloud Functions* based on *OpenWhisk*[2], *IBM Event Streams* using *Kafka*[3], *IBM Cloud Object Storage* and *Cloudant* –based on *CouchDB*[4]– for data storage, and *IBM Edge Application Manager* for managing edge device services, similar to Azure IoT Edge, with additional support for serverless computing through *Edge functions*. Edge computing solutions rely on the *IBM Cloud Satellite* service for managing hybrid cloud deployments, which is more powerful but more complex than AWS Greengrass or Azure IoT Edge.

Other Platform-as-a-Service offerings, focused on the cloud-to-edge and the cloud-to-thing continuum, have been introduced recently. For instance, **Particle**[5] provides a cloud-based management platform for devices running their *Device OS*, which simplifies user applications integration into firmware and lifecycle management. Similarly, the **Arancino**[6] [39] platform comprises: (i) a family of open-hardware dual-board devices, which can perform as edge computing devices as well as IoT field devices for sensing and actuation; (ii) a cloud-based IoT management platform grounded on OpenStack.

[2]Apache OpenWhisk: `https://openwhisk.apache.org/`
[3]Apache Kafka: `https://kafka.apache.org/`
[4]Apache CouchDB: `https://couchdb.apache.org/`
[5]Particle: `https://www.particle.io/`
[6]Arancino: `https://arancino.cc/`

EdgeImpulse[7] supports cloud-to-edge ML training and can integrate *Arduino*[8] microcontrollers for inference tasks.

Ideally, the machine learning deployment framework should encompass several key characteristics: support for incremental learning, federated learning, and capabilities for both training and inference from cloud to IoT environments. Additionally, an important requirement is that the framework's installation size must be sufficiently compact to fit within the constraints of a serverless function runtime. Specifically, for AWS Lambda, this means the compressed package size must be kept under 50 MB.

A significant gap has been identified in this regard, as there is a lack of a comprehensive framework that encompasses all these essential features, which becomes evident when considering the capability for both training and inference on IoT environments. For the deployment on cloud-to-edge, some state-of-the-art libraries such as **scikit-learn**[9] can be easily integrated into an AWS Lambda function, however larger frameworks such as **TensorFlow**[10] and **PyTorch**[11] exceed size limits and are not as easily integrated. **TensorFlow Lite** is exploitable for inference and fine-tuning on both cloud and edge devices, but its models can be run on IoT field devices only for inference.

4.3 Case study: federated learning for activity recognition

In order to clarify the proposal and highlight its features, a prototypical testbed has been fully developed for a federated learning case study concerning the domain of activity recognition. *Activity recognition* holds significant

[7]EdgeImpulse: `https://edgeimpulse.com/`
[8]Arduino: `https://www.arduino.cc/`
[9]scikit-learn: `https://scikit-learn.org`
[10]TensorFlow: `https://www.tensorflow.org/`
[11]PyTorch: `https://pytorch.org/`

relevance, since its applications span from health and fitness monitoring to smart homes and public safety, encompassing the ability to discern human activities like standing, walking, running, and more. This case study aims to illustrate how the proposed federated learning framework, with its decentralized model training approach, can be applied to an activity recognition dataset.

4.3.1 Prototype deployment

The prototype closely adheres to the description provided in Section 4.2.1. It includes: 1 AWS Cloud Provider node, 1 Edge Node, 3 IoT FDs attached to the Edge Device and 3 IoT FDs attached directly to the Cloud Provider node. The deployment strategy has followed a top-down approach, started by configuring the cloud infrastructure on AWS, then by provisioning the Edge Device, and finally by programming the FDs.

In the cloud infrastructure setup, the primary focus has been on developing the four AWS Lambda functions described in Section 4.2.1. Each lambda has been programmed and packaged in a standalone *.zip* archive, and subsequently uploaded to the cloud using the AWS Console. The Lambda functions responsible for data processing, model updating, and prediction are configured to be triggered by their corresponding MQTT messages, meanwhile the Update Model function is scheduled for periodic invocation through AWS EventBridge.

Field and Edge devices have been organized into IoT Groups at this stage. Within these groups, the necessary permissions and policies have been outlined, authorizing the nodes to access relevant resources. This includes authorizing the connectivity to Cloud MQTT Broker and allowing Edge Nodes to authenticate edge devices within their respective group.

To guarantee seamless compatibility between edge and cloud Lambda runtimes, certain requirements must be carefully considered during the Lambda development process. Specifically, the following key considerations must be

kept in mind:

- Lambda functions should be either uploaded as standalone *.zip* files, with a maximum file size of 50 MB, or developed directly using the inline code editor in the AWS Console. This is crucial because AWS Greengrass service does not support alternative Lambda formats, such as layered packages and container functions;
- the chosen runtime for the Lambda function must be compatible and available with both AWS Greengrass CoreDevice and the AWS Lambda execution environment;
- if the Lambda package includes native code —-such as libraries that have bindings to native libraries—- it is crucial to align the Central Processing Unit (CPU) architecture of the cloud Lambda runner with that of the edge device. As an alternative, separate Lambda functions should be deployed for each distinct CPU architecture to be supported.

To meet these requirements, Python (version 3.9) has been selected as the programming language to implement the aforementioned functions. This language ensures compatibility across both edge and cloud environments. Additionally, the *aarch64* ARM 64-bit architecture has been chosen for Cloud Provider node instances to match the CPU architecture of the Edge Node. The MAFALDA tool, originally implemented in Java [119], has been re-implemented in C with a Python wrapper: its compactness (357 kB overall, libraries included) is a beneficial feature, as it helps minimize the package size, thereby reducing load times.

For the Edge Node, a Raspberry Pi 4 Model B[12] hosts the GreenGrass CoreDevice service. This service enables remote control and monitoring of the Edge Device from the AWS Console, as well as the deployment of custom components. In a nutshell, CoreDevice provisioning can be split in two key parts:

[12]ARM® Cortex®-A72 Quad-Core CPU @ 1.5 GHz, 8 GB of RAM, and 32 GB of Secure Digital (SD) storage memory

1. **Setup of the Operating System**: installation of *Raspbian OS Bullseye* (version 11) is required, along with OpenJDK version 11.0.20, to meet the prerequisites of the AWS Greengrass CoreDevice installer.

2. **AWS Console CoreDevice Setup**: configuration of the CoreDevice software is performed via the AWS Console, which allows to download the installer package. Executing this installer on the Raspberry Pi completes the setup process.

Upon edge device registration, the configuration described earlier in Section 4.2.1 can be imported. This is achieved in two steps:

1. import the four Lambda functions into AWS Greengrass as *Components*;

2. create a new *Greengrass Deployment* with all the parts listed in Section 4.2.1, and specifically in Figure 4.4, including the newly created four Lambda Components;

Minimal component configuration is mandatory to complete the CoreDevice setup, specifically:

- the Nucleus authentication component requires permissions to enable the IoT Group to both publish and subscribe to MQTT topics;
- MQTT Bridge needs to be configured to relay messages from client devices (*i.e.*, `train`, `test` and `predict` topics) to the PubSub broker and messages from Lambda functions (with `result` topic) to the local MQTT broker, allowing client devices to communicate with Greengrass component;
- setup local MQTT topic to trigger Lambdas on the edge device;
- specify the local MQTT broker endpoint to the *Greengrass Discovery API*[13].

[13] `https://docs.aws.amazon.com/greengrass/v2/developerguide/greengrass-discover-api.html`

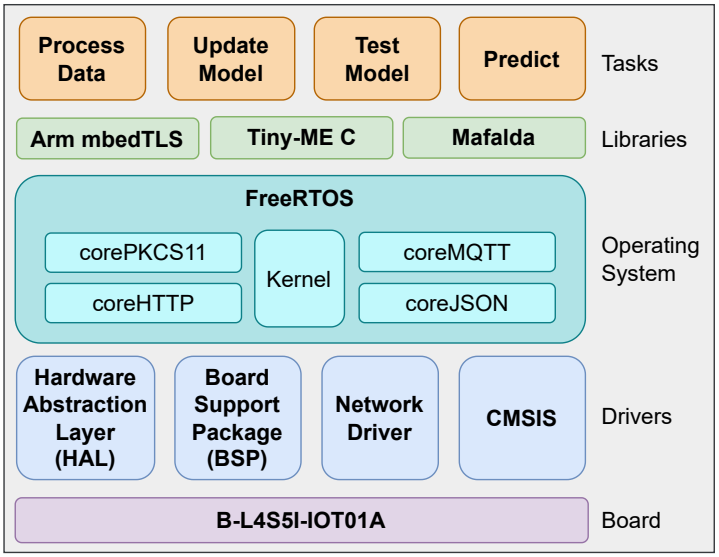


Figure 4.9: IoT Field Device firmware

The last task has been needed to develop FDs firmware, whose architecture is shown in Figure 4.9. It is designed following a layered software architecture. Starting from the bottom layer, the reference development board chosen for this experimentation has been the *STM32 Discovery kit IoT Node B-L4S5I-IOT01A*, having the following hardware configuration: STM32L4S5VIT6 Micro Controller Unit (MCU) with 120 MHz Arm Cortex-M4 core; 2 MB of flash memory; 640 kB of SRAM; wireless connectivity with Wi-Fi, Near Field Communication (NFC) and Bluetooth Low Energy (BLE); a wide range of sensors, including a gyroscope, accelerometer, magnetometer, proximity, pressure, humidity and a microphone; embedded *ST-LINK* debugger and programmer.

Drivers are the lowest layer of software. They include:

- **Hardware abstraction Layer**, specific for the processor, which provides human-readable names and functions to access hardware components of the MCU;
- **Board Support Package**, specific for the board, which provides higher-level interfaces to access sensors and hardware features;

- **Network Driver** for the ISM43362-M3G-L44 Wi-Fi module of the board, providing a complete TCP/IP network stack (mandatory requirement to support AWS Greengrass);
- **Common Micro-controller Software Interface Standard (CMSIS)**, which is a set of standard APIs and interfaces for ARM microprocessors to promote code reusability and interoperability across MCU vendors.

CMSIS is specifically required by **FreeRTOS+** [14] which is the main building block of the upper software layers. FreeRTOS+ is composed by the FreeRTOS kernel and some utility libraries. The FreeRTOS kernel provides concurrency as well as primitives and data structures for synchronization (Mutexes, Semaphores, Timers, Queues, etc.). FreeRTOS+ libraries include: (i) *corePKCS11*, implementing a subset of the PKCS11 API to access cryptographic objects; (ii) *coreMQTT*, providing a MQTT client; (iii) *coreHTTP* and *coreJSON* to serialize/deserialize HTTP and JSON messages, respectively. *ARM mbedTLS* is an additional library needed to support mutual authentication mechanisms and encrypted communications, which are mandatory when communicating with AWS IoT Core and GreenGrass Core-Device. The IDE of choice has been STM32CubeIDE, provided by the board vendor, with the *X-Cube-AWS* expansion pack that provides ports of the aforementioned FD building blocks.

Manual porting has been required of the other components, namely:

- **MAFALDA** [119] and the **Tiny-ME C** [123] reasoning engine libraries have been ported to STM32 in order to support on-device ML training and inference;
- **Process data**, **Update Model**, **Test Model** and **Predict** tasks, described in Section 4.2; unlike the edge and cloud functions, which are invoked via MQTT messages, here the tasks are triggered when new sensor data are acquired.

[14] `https://www.freertos.org/FreeRTOS-Plus/`

The firmware, developed exclusively in C, is uploaded on the device using the ST-LINK Programmer. To correctly provision the device, and specifically to be authenticated into AWS IoT Core, it is necessary to follow this procedure:

- provision the secure element and retrieve the client certificate;
- sign and upload the firmware to the device;
- create a new *thing* on AWS IoT Core with the certificate from the secure element;
- attach policies to the certificate allowing the relative device to connect and subscribe to MQTT Brokers;
- add the device to the IoT Group;
- add the node to the Discovery API list on Greengrass;

Upon completion of these steps, all devices in the final prototype can communicate through an encrypted and authenticated connection.

4.3.2 Reference dataset

To illustrate the usefulness of the proposed federated learning framework, a small case study has been developed leveraging the *MotionSense* [84] dataset, which is publicly accessible under the Open Database License (ODbL) v1.0 on *Kaggle*[15].

MotionSense comprises accelerometer and gyroscope sensor data collected from iPhone 6s devices through the *Core Motion API*[16]. Specifically, the dataset consists of the following sensor measurements:

[15]https://www.kaggle.com/datasets/malekzadeh/motionsense-dataset
[16]https://developer.apple.com/documentation/coremotion/cmdevicemotion

- **Attitude**: device orientation in terms of roll, pitch, and yaw;
- **Rotation Rate**: angular velocity of the device;
- **Gravity**: acceleration vector relative to gravity, expressed in the device own reference frame;
- **User Acceleration**: acceleration imparted to the device by the user.

Each type of sensor measurement is captured independently for each axis, for a total of 12 features.

These data were gathered from 24 different participants, each instructed to perform one of six activities: going downstairs, going upstairs, walking, jogging, sitting and standing. Each individual performed 15 different trials, during which the data were collected at 50 Hz sampling rate. Additionally, the dataset includes a label indicating which of the six activities was performed by the subject during the data collection process. MotionSense has been chosen in this work because it represents a realistic use case of data collection from multiple field devices, as the iPhone 6s could be replaced by a smaller wearable device. It can be assumed that each device monitors the activities of a subject.

Data have been preprocessed by aggregating 50 individual samples, corresponding to one second of data, into a single composite sample, incorporating both the mean and the standard deviation for each sensor across all its axes, thus resulting in 24 distinct features. It is important to note that this data aggregation was performed prior to the transfer of data to the designated field devices, thereby simplifying subsequent experimental procedures. In practical applications, the field devices are expected to conduct such preprocessing tasks in real-time. Consequently, it is essential that the computational requirements are compatible with the capabilities of the reference MCU. A preliminary test conducted with an STM32 board has validated this point: the board has been preloaded with raw data and instructed to execute the aggregation procedure. The computation time has been found to be, on average, 0.2 ms for each set of 50 samples, equating to one second of data

capture. Furthermore, the available onboard memory has been sufficiently large to buffer the generated batch of samples, in accordance with the experimental settings. These results support the capability of the STM32 board to effectively handle the specified preprocessing tasks.

Afterwards the preprocessed dataset has been divided by subjects. Specifically, data on subjects 1 to 6 are reserved to perform the initial MAFALDA model selection. This model resulting from the initial data is referred as the **bootstrap model**. While not strictly mandatory in the context of this framework, hyperparameter optimization at this stage may enhance model accuracy down the line. Also, it is reasonable to assume that limited data are available to choose an initial model. In any case, subsequent updates to the model are possible as more data are collected.

The remaining data have been split among all FDs as follows:

- *c1*: subjects 7 to 9;
- *c2*: subjects 10 to 12;
- *c3*: subjects 13 to 15;
- *e1*: subjects 16 to 18;
- *e2*: subjects 19 to 20;
- *e3*: subjects 21 to 24.

where FDs *c1*, *c2* and *c3* are associated to a Cloud IoT Group and FDs *e1*, *e2* and *e3* to an Edge IoT Group.

4.3.3 Illustrative examples

The versatility of the proposed framework extends its applicability to a wide array of use-cases, including privacy-sensitive domains like industrial workplace safety. In such environments, the worker can be equipped with wearable

IoT sensors that monitor movements and environmental conditions in real time. Each wearable could be configured to either locally update its machine learning model for immediate inference or to send the data to an on-site edge node, such as a Raspberry Pi. This edge node could perform more complex inferences and, if necessary, share only the aggregated and anonymized model updates with a centralized cloud service. In turn, the cloud could merge these updates with other models received by additional edge nodes from multiple sites and, if necessary, perform further data analytics.

One of the key features of this framework is its flexibility, since it offers training and inference across different layers of the network. This multi-layer approach enables the system to be fine-tuned according to varying requirements and limitations. For instance, if the network conditions are challenging or unreliable, the edge and IoT devices can still carry on with essential monitoring and prediction tasks. Moreover, this architecture supports scenarios where conventional cloud-based solutions might fall short, such as in compliance with privacy regulations that prohibit the fine-grained tracking of employees.

Another compelling application for this framework is in the area of elderly care, particularly for in-home or ambient-assisted living environments. In those settings, a network of IoT sensors could be strategically placed around the living space or even worn by the elderly individuals. The sensors could monitor a variety of metrics such as movement, heart rate, and even ambient conditions like room temperature or air quality. Like in industrial settings, these IoT devices could either update their machine learning models locally or transmit data to a nearby edge node for more complex analysis. An edge node could be a dedicated home server or a smart home hub capable of ML computations. The aggregated anonymized model data could then be sent to a centralized cloud service for larger-scale analytics, such as predictive health assessments or emergency event recognition.

The true advantage of this federated architecture becomes apparent when considering the delicate balance between the need for high-quality care and the privacy concerns often associated with monitoring vulnerable popula-

tions. By enabling machine learning to occur at the device or edge level, sensitive data can be processed locally, thereby reducing the amount of personal information that needs to be sent to the cloud, and for this reason, this approach aligns well with privacy regulations and ethical considerations.

Extending considerations beyond activity recognition, the proposed federated cloud-to-thing approach offers advantages in scenarios characterized by challenging network conditions. For example, considering a remote farming setting, where network connectivity is inconsistent. IoT sensors can be deployed throughout the farmland to monitor soil moisture, temperature, and other vital parameters for crop health. Analogously in smart grid systems, where network connectivity can often be unreliable, especially in remote areas, IoT devices embedded in transformers or substations can locally process data for anomaly or fault detection. When network conditions allow, these devices can send essential data to a local edge node for further analysis. This setup ensures monitoring is performed continuously, even when connectivity to the cloud is unstable.

4.4 Experiments

Following the case study described in Section 4.3, an experimental campaign on real devices has been carried out to prove the feasibility of the proposed approach and to assess its performance.

4.4.1 Materials and methods

The experimental testbed has been set up as described in Section 4.3.1. A Raspberry Pi 4 Model B+ has acted as the edge CoreDevice, while IoT FDs have been implemented with three B-L4S5I-IOT01A boards, chosen due to their compatibility with the AWS stack. In particular, AWS documentation provides instructions to install their Greengrass software specifically for Raspberry Pi and B-L4S5I-IOT01A boards have certified compatibility with

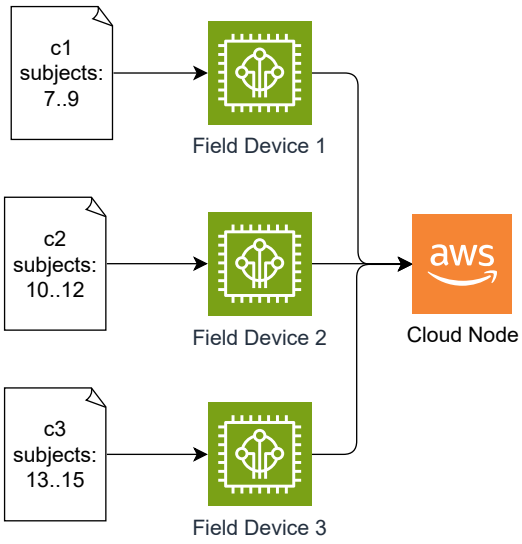


Figure 4.10: Experimental setup - cloud only

AWS, meaning that STM32 provides a set of libraries to facilitate the integration with AWS IoT Core. Using other boards or MCU is still possible, but requires to port FreeRTOS libraries to the new device manually, or to find existing working ports. For instance, the Arduino and Arancino families of development boards have a compatible implementation of FreeRTOS[17]. Greengrass is generally simpler to install since it requires only a Linux operating system on the target device and a compatible Java Runtime Environment (JRE).

The first step involves data preparation, as described in Section 4.3.2. Briefly, data have been partitioned by subject and, for each partition, 20% of data have been held out to evaluate model accuracy. Initial data of subjects 1 to 6 have been used to train the MAFALDA **bootstrap model** and select an appropriate threshold value, which has been found to be 0.11, maximizing the accuracy of the model w.r.t. the test samples of subjects 1 to 6. The bootstrap model has been therefore loaded on the Model Store.

In the first experiments, the Edge Node has been turned off and the three physically available STM32 boards have been connected directly to the cloud

[17]https://www.arduino.cc/reference/en/libraries/freertos/

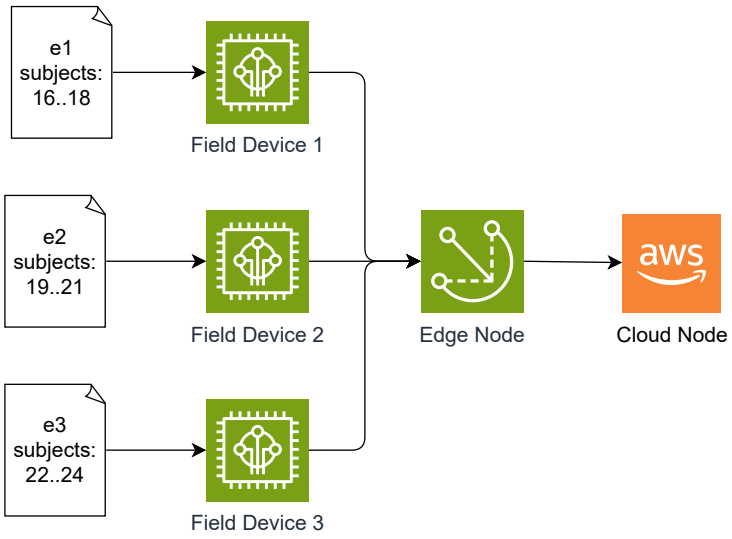


Figure 4.11: Experimental setup - cloud and edge

AWS IoT Core. Devices have been loaded with *c1*, *c2* and *c3* data subsets, respectively, and programmed to send data in batches of 256 samples each second. The training data have been sent on the `train` topic, while test data have been sent on the `test` topic; they all have been stored into the Cloud Data Store, as explained in Section 4.2.2 and Figure 4.5. This setup is sketched in Figure 4.10. Data has been also used to update and test the on-board ML model.

It is useful to note that MAFALDA's accuracy does not really depend on batch size, as training samples are processed one by one to build the TM incrementally, as explained in Section 4.1.1. For other algorithms, however, it might be necessary to determine the optimal value while training the bootstrap model. In this phase, FDs have logged the time elapsed between the publish operation and the receipt of each of the two acknowledgements sent by the Cloud Lambda function, as shown in Figure 4.5, in order to assess network latencies. Lambda functions add to acknowledgment messages of prediction and test the time elapsed during model inference, in order to profile the computational overhead due to MAFALDA invocation.

Subsequently the configuration has been modified as shown in Figure 4.11, by adding an Edge Node to the network. The same experiment has been

repeated using *e1*, *e2* and *e3* data subsets, now measuring communication latencies between the Greengrass Core Device and the FDs. All six data subsets are therefore uploaded to the Cloud Data Store.

For the last test, AWS EventBridge fires an event that triggers the *Update Model*, updating the model contained in Model Store with data from the cloud Data Store. Upon completion, notifications are sent to each Edge Node and Field Device, instructing them to perform the update with their respective data. The model that has been trained with data from all devices is identified hereafter as the **final model**. This final model has been evaluated against the bootstrap model to assess any improvements in accuracy.

Overall, the test has been conducted in two phases. In the first phase, the three available B-L4S5I-IOT01A devices have been attached to the AWS cloud in a Cloud IoT Group. At the second stage, they have been connected to the local network and configured in the Edge IoT Group managed via the Greengrass Edge Node. Despite the limitation of not having all six IoT devices simultaneously, the experimentation and the comparisons are still valid, as: (i) the test methods in the two phases are coherent; (ii) tests comparing the two configurations do not require all IoT devices to be online at the same time; (iii) the final model update occurs by processing all the six parts of the dataset mapped to the six IoT Field Devices anyway, as explained above.

4.4.2 Results

Building upon the prototype described in Section 4.3.1 and test methodology outlined in Section 4.4.1, this section reports results, focusing on three critical performance metrics: communication latencies, processing times, and model accuracy. The aim is to assess the overall framework performance, strengths and weaknesses and to evaluate its applicability in real-world scenarios.

Latency metrics: Latency data contains the round-trip time between the initiation of a request and the receipt of the acknowledgment messages,

Table 4.4: Communication latencies for cloud-connected Field Devices (ms)

		Dataset subset			**Overall**
		c1	*c2*	*c3*	
First Ack.	*Avg.*	992	609.72	600	**741.29**
	Max	2969	714	687	**2969**
	Std.Dev.	725.32	51.25	52.37	**472.25**
Second Ack.	*Avg.*	47.9	47.18	47.18	**46.35**
	Max	151	76	79	**151**
	Std.Dev.	36.18	18.47	20.06	**26.36**
Total	*Avg.*	1039.91	656.91	647.18	**787.64**
	Max	3000	747	758	**3000**
	Std.Dev.	726.29	48.51	61.05	**437.34**

providing insights into the network overhead of data transfer and processing within the system. Measurement has been conducted at the Field Device level, which are the most directly impacted by the communication latency. Average, maximum and standard deviation results are reported in Table 4.4 for FDs connected to the cloud, and in Table 4.5 and for edge-connected FDs. The data are visualized in Figure 4.12 and Figure 4.13 for cloud and edge-connected Field Devices respectively, meanwhile Figure 4.14 provides an aggregated, side-by-side comparison of overall mean network latency for the two cases.

Specifically, regarding the interactions illustrated in Figure 4.5, the rows in Table 4.4 and Table 4.5 correspond to the following latency metrics:

- **First Ack.**: the time from the publication of raw training data to the receipt of an acknowledgment confirming data receipt.
- **Second Ack.**: the interval between receiving the first acknowledgment (data received) and the second acknowledgment (data stored).
- **Total**: the total time from the publication of raw training data to the acknowledgment confirming data storage.

Table 4.5: Communication latencies for edge-connected field devices (ms)

		Dataset subset			**Overall**
		e1	*e2*	*e3*	
First Ack.	*Avg.*	284.9	285.9	284.4	**285.06**
	Max	322	310	325	**325**
	Std.Dev.	17.59	15.87	22.16	**18.7**
Second Ack.	*Avg.*	101.63	53.2	60.1	**72.61**
	Max	296	73	206	**296**
	Std.Dev.	71.89	13.81	49.47	**56.18**
Total	*Avg.*	386.54	308.27	313	**357**
	Max	596	371	467	**596**
	Std.Dev.	76	25.71	46.78	**58.58**

The columns in these tables classify data by device, each associated with specific data subsets. An *Overall* column provides aggregated statistics across all three devices.

As expected, the latency results indicate that edge computing outperforms cloud computing in terms of response times. This difference is primarily due to the network overhead inherent to cloud infrastructure, as the edge node, being located near the IoT devices within the same network, avoids the transmission delays associated with the cloud's more remote location.

Notably, an initial latency spike is observed in the cloud node, particularly during the first invocation of the Lambda function, shown under the 'c1 Max Response Time' metric. This peak is attributed to the cold-start issue, a known drawback of serverless architectures. However, this limitation does not significantly affect the proposed framework's efficiency. The Edge Node does not exhibit this cold-start latency, underscoring an advantage of the Edge Lambda Runtime over the Cloud Runtime.

Processing time: Computational turnaround times were evaluated to measure the processing overhead on each node. Measurements were recorded directly on the node performing the computation, isolating the assessment

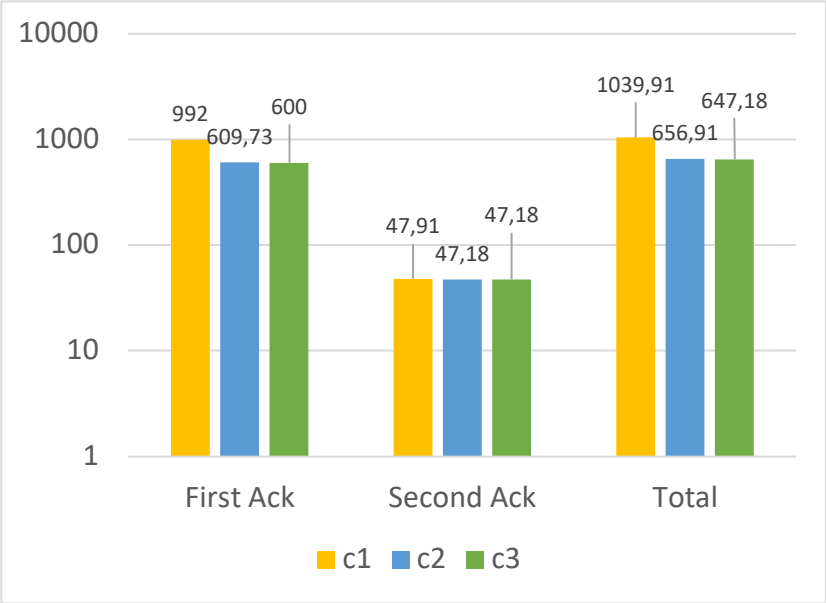


Figure 4.12: Network Latency - FD to cloud (ms)

from Lambda startup delays and network latencies. This analysis focuses on verifying that MAFALDA operates effectively across all layers of the architecture, ensuring the framework's lightweight and practical nature for real-world deployment.

In serverless computing, resource allocation is crucial for function execution. In AWS Lambda, the CPU tier assigned to a function is indirectly controlled by specifying the required Random Access Memory (RAM).[18] This allocation also impacts the cost per second billed:[19] Table 4.6 summarizes pricing for Lambda functions in the *eu-central-1* region (as of December 20, 2023) across five RAM tiers using ARM CPUs. AWS charges proportionally to the function's duration. Table 4.7 reports execution time measurements and costs for three cloud-based operations for each memory tier: downloading and decompressing a 256-batch, updating the model with a single batch, and predicting a batch of data. The 2048 MB configuration consistently provides the fastest execution time, while the 512 MB tier emerges as the most

[18] `https://docs.aws.amazon.com/lambda/latest/operatorguide/computing-power.html`

[19] `https://aws.amazon.com/lambda/pricing/`

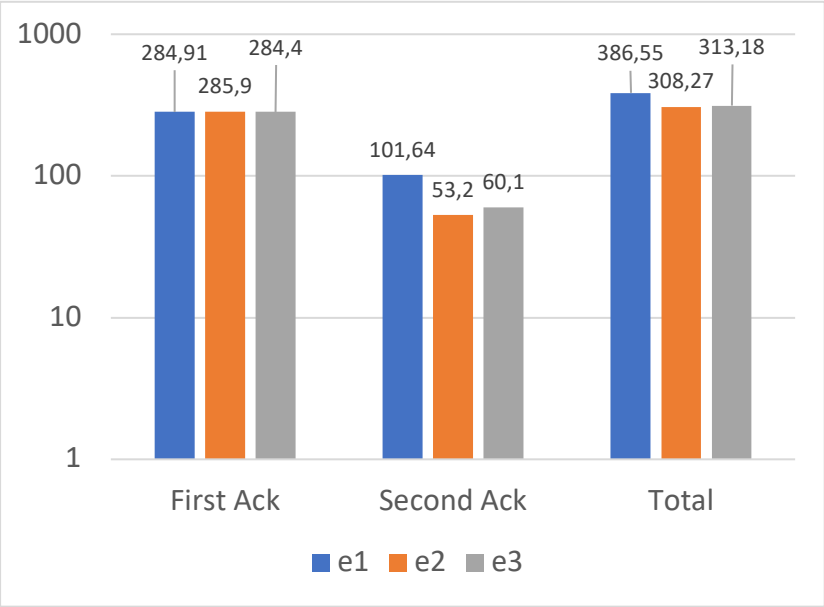


Figure 4.13: Network Latency - FD to edge (ms)

cost-effective overall. For training tasks specifically, the 1024 MB tier offers the best cost-efficiency. These findings highlight the importance of testing function configurations to determine the optimal cloud node setup for each specific task.

Table 4.8 and Figure 4.15 compare results on the average, standard deviation, and maximum execution time among Field Devices, Edge Devices, and the 512 MB tier Cloud Nodes.

Table 4.6: AWS Lambda Pricing in eu-central-1 region (10^{-7}\$ / ms)

Memory Config. (MB)	**ARM CPU Pricing**
128	0.021
512	0.083
1024	0.167
1536	0.25
2048	0.333

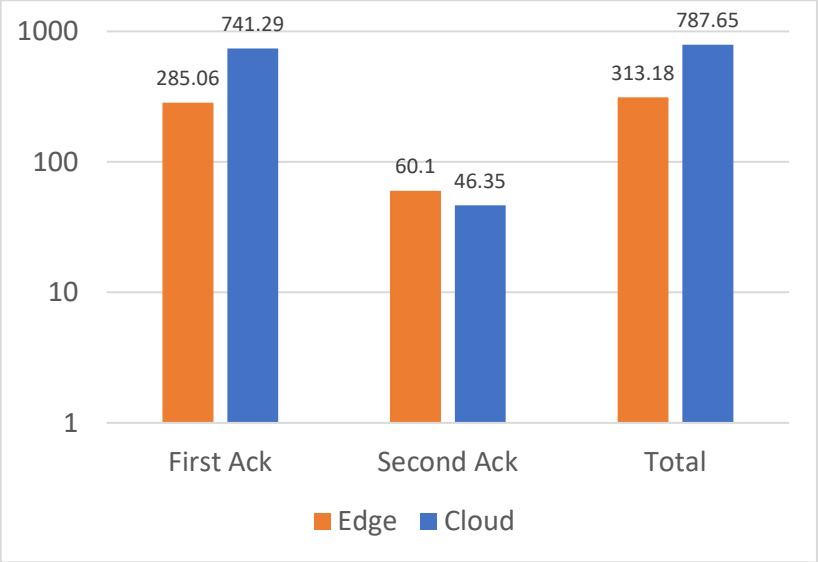


Figure 4.14: Network Latency - Comparison between FD to cloud and FD to edge (ms)

In the experimental setup, the Edge Node has shown faster execution times compared to the cloud in terms of computational speed, meanwhile the time required to download and prepare the batch for processing is higher on edge. Both the edge and cloud infrastructures offer vertical scalability, since the Greengrass software stack can be installed on more capable hardware at the edge, while the cloud AWS Lambda execution tier can be increased. A direct cost comparison is a complex task, since it should consider edge hardware and energy cost models as well as IoT Core messaging and storage expenses. AWS charges each device a fixed cost of $0.18 per month, regardless of its capabilities. Therefore, when the number of requests increases significantly, edge devices may become more cost-effective.

An expected advantage of the cloud infrastructure is its superior horizontal scalability compared to the edge devices, although the limited scope of this study, focusing on a small number of devices, could not provide the conditions under which this advantage could be clearly exposed and measured.

In addition, it is possible to note how IoT devices have been able to perform machine learning tasks, thanks to MAFALDA's ability to operate

Table 4.7: Cloud Node: execution time (ms) and cost (10^{-7}\$)

Memory Config (MB)		Retrieve Batch	Train on Batch	Predict Batch	Total
128	*Time*	285.72	448.14	193.23	927.09
	Cost	6.00	9.41	4.06	19.47
512	*Time*	71.16	108.62	48.31	228.09
	Cost	**5.91**	9.02	**4.01**	**18.93**
1024	*Time*	60.84	49.74	31.83	142.41
	Cost	10.16	**8.31**	5.32	23.78
1536	*Time*	57.00	34.41	17.89	109.30
	Cost	14.25	8.60	4.47	27.33
2048	*Time*	51.31	31.26	11.18	93.75
	Cost	17.09	10.41	3.72	31.22

in resource-constrained environments. Although the processing time on IoT devices is considerably higher than both the edge and the cloud, even when accounting for network latencies, considering that every batch amounts to 256 s of data acquisitions w.r.t. the reference dataset, IoT model training and prediction times can be deemed as acceptable in a realistic scenario. The capability of MAFALDA to execute on these devices is a distinct advantage of the proposed framework. This is especially relevant in harsh or challenging environments where network communications may be limited or unreliable, thereby enhancing the framework versatility.

Model accuracy: finally, the effectiveness of the ML algorithm and the impact of the federated learning environment have been evaluated through prediction accuracy measurements. This addresses the algorithm's ability to incrementally learn across the architecture by evaluating its accuracy metrics derived from the confusion matrix. The evaluation consists in three steps: initially, a bootstrap model is selected using a subset of the original dataset, as described in Section 4.3.2 ; this model is trained and evaluated to record accuracy metrics; subsequently, the same metrics are captured for the final model that has undergone updates with partial models and data from the FDs. By comparing these two sets of measurements, the analysis aims to as-

Table 4.8: Mafalda functions training and prediction performance (ms)

		Retrieve Batch	**Train on Batch**	**Predict Batch**
Field Device	*Avg.*	*N/A*	10550	9286.2
	Max	*N/A*	10625	9527
	Std.Dev.	*N/A*	66.67	160.51
Edge Device	*Avg.*	166.61	68.97	26.47
	Max	185.14	73.22	28.07
	Std.Dev.	14.09	2.29	1.44
Cloud Node	*Avg.*	71.16	108.62	48.31
	Max	116.16	135.53	64.60
	Std.Dev.	23.22	16.17	21.15

Table 4.9: Bootstrap model - Confusion matrix

dws	**jog**	**sit**	**std**	**ups**	**wlk**	*<– Classified as*
134	3	1	4	11	12	**dws**
23	138	0	1	4	18	**jog**
0	0	431	21	0	0	**sit**
0	0	5	395	0	0	**std**
40	1	0	7	122	13	**ups**
91	18	7	3	25	260	**wlk**

sess whether the framework can achieve incremental improvement. Table 4.9 reports the confusion matrix for the bootstrap model and Table 4.10 shows the associated performance metrics, including precision, recall, and F1 score for each class, as well as the overall accuracy. Table 4.11 Table 4.12 provide the same data for the final model.

The prediction accuracy has exhibited only a slight improvement of $\sim 2\%$ from the initial bootstrap model to the final checkpoint. This modest gain can be deemed as more indicative of limitations of Mafalda itself rather than a shortcoming of the federated learning approach. One significant limitation is Mafalda's need for incremental training across all layers of the architec-

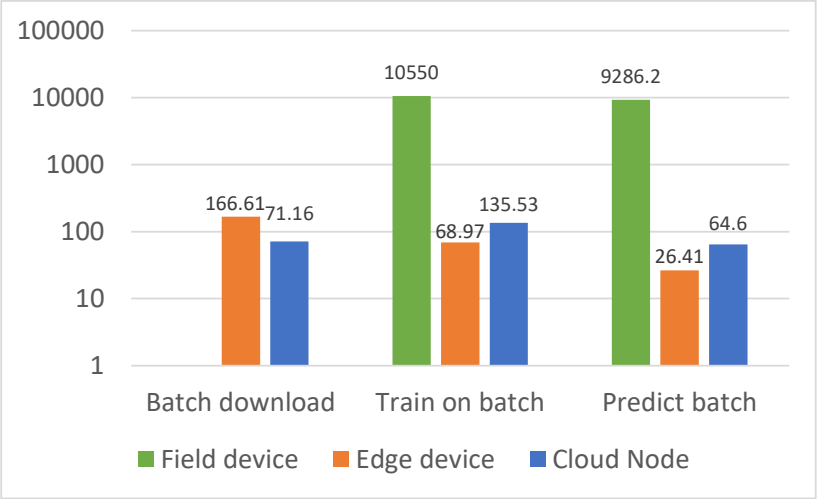


Figure 4.15: Mafalda train and predict performance (ms)

Table 4.10: Bootstrap model - Evaluation summary

Precision	Recall	F1Score	Accuracy	Class
0.465	0.812	0.592	-	*dws*
0.863	0.750	0.802	-	*jog*
0.971	0.954	0.962	-	*sit*
0.916	0.988	0.951	-	*std*
0.753	0.667	0.707	-	*ups*
0.858	0.644	0.736	-	*wlk*
0.804	0.802	0.792	0.828	***Average***

ture without increasing its expressiveness, which may not optimally leverage the computational capabilities at each layer. It is worth noting that the framework enables a model update feedback loop for progressive performance improvement in realistic applications based on continuous data streams. Furthermore, it is able to execute other ML algorithms, including neural networks that are trained incrementally in mini-batches. However, a careful trade-off with the computational demands of more complex algorithms is required in order to not preclude their deployment to resource-constrained IoT devices.

Table 4.11: Final model - Confusion matrix

dws	**jog**	**sit**	**std**	**ups**	**wlk**	<– *Classified as*
128	2	1	4	5	25	**dws**
21	145	1	0	3	14	**jog**
0	0	452	0	0	0	**sit**
0	0	0	400	0	0	**std**
17	0	0	4	132	30	**ups**
82	5	11	2	39	265	**wlk**

Table 4.12: Final model - Evaluation summary

Precision	**Recall**	**F1Score**	**Accuracy**	**Class**
0.516	0.776	0.620	-	*dws*
0.954	0.788	0.863	-	*jog*
0.972	1.000	0.986	-	*sit*
0.976	1.000	0.988	-	*std*
0.737	0.721	0.729	-	*ups*
0.793	0.656	0.718	-	*wlk*
0.825	0.824	0.817	0.851	***Average***

Based on the application requirements, a two-tiered approach could also be envisioned: deploying a more expressive model on the cloud and edge layers while retaining a less computationally intensive model like Mafalda on Field Devices. This strategy would aim to balance the trade-offs between computational resources and prediction accuracy, thereby providing a good compromise between the two solutions.

Chapter 5

Semantic-enhanced blockchain for service-oriented pervasive Cyber-Physical Systems

Intelligent objects embedded with sensing, processing, and communication abilities are interconnected via mobile networks as envisioned by the IoT. These micro-devices gather, generate, and handle data while exchanging information and services in their immediate environment. This setup facilitates detailed, distributed control with minimal response time and efficient energy use. In this context, architectures leveraging the MEC model support the IoT by enhancing context-awareness and traceability across various industries, including manufacturing, healthcare, environmental monitoring, and Smart Mobility.

Among the significant challenges hindering widespread IoT adoption in such contexts are security and trust. In traditional distributed systems, the integrity of data and service exchanges relies on transactions validated by a trusted central authority. However, this model is unsuitable for ephemeral IoT environments, where ensuring security during rapid digital transactions is challenging due to the broad attack surface presented by edge servers, network domains, and software-defined networking [85].

Blockchain technology presents viable solutions in this regard. Essentially, a blockchain is a shared database or *ledger* distributed among multiple

node peers, accompanied by a protocol for recording transactions within a specific timeframe. The dependability of blockchain-based platforms is derived from requiring that all transactions gain *consensus* from peer nodes, thus facilitating trustworthy data and service exchanges in decentralized and inherently *trustless* settings, without relying on central authorities. Already revolutionizing various sectors, blockchain aids in mitigating fraud risks and curbing operational expenses. In these environments, smart contracts deliver advanced services. A smart contract is an automated software routine that executes the terms of an agreement in a format understandable by machines.

The inherent uncertainty in resource availability caused by node volatility presents obstacles for IoT. Thus, implementing sophisticated techniques for decentralized and dynamic service/resource discovery is essential [107, 111]. Unfortunately, in terms of trust management, more adaptable discovery methods are constrained by the rigidity of existing blockchain systems, which only allow for retrieving services/resources via identifier string-matching or simple attributes.

The SWoT framework [126] offers a promising approach to overcoming these constraints by integrating KR methods, particularly Semantic Web technologies, into the resource discovery in mobile systems. It proposes the deployment of numerous micro-devices in edge environments, providing resources with concise, formal annotations linked to a shared vocabulary (*ontology*). This enables semantic matchmaking, allowing for interoperable resource discovery that aligns with the user's request, thus granting decision-making autonomy in an IoT context. While SWoT alone does not specifically resolve issues related to trust management and transaction reliability during and after resource discovery in large-scale MEC contexts, blockchain technology complements it by safeguarding against data tampering through transaction verification using consensus protocols, thereby upholding smart contracts. A SWoT blockchain acts as a SOA for registering, discovering, and selecting annotated services and resources, facilitated by distributed smart contracts authenticated by consensus, paving the way for creating interoperable, adaptable, robust, and scalable infrastructures.

Moreover, within architectures marked by a growing and substantial number of interconnected devices, scalability becomes an essential requirement, with optimal performance being vital to ensure prompt service delivery for each request. Consequently, it is crucial to examine the variation in blockchain performance across different node densities when it is employed as the foundational infrastructure for services marketplaces. Furthermore, services must be designed to accommodate various computational limitations present in an IoT setting.

This chapter introduces *SeeSaw* (SEmantic-Enhanced SAWtooth), a semantically enriched SOA designed for trustless cooperation within MEC and pervasive computing, targeting sophisticated IoT environments [118]. It is developed upon the *Hyperledger Sawtooth* [1] open-source blockchain framework [98]. The primary contributions of this work are outlined below:

- Integration into blockchain systems of a dynamic service discovery layer that utilizes semantic matchmaking. This layer facilitates logic-based service prioritization and—importantly—provides reasoning for decisions, through non-standard inferences executed as smart contracts.
- To enhance transparency and interoperability, services are recorded as *assets* on the blockchain, with semantic annotations articulated in the *OWL* 2 [145] standard.
- SeeSaw has been implemented in a fully operational prototype and evaluated through an experimental study, simulating progressively larger MEC networks. Initial findings confirm the approach's validity and highlight its computational efficiency and sustainability.
- A case study is presented to demonstrate the proposal's features and potential advantages, concentrating on energy management of *PEVs* within a *smart grid* setting.

The remainder of the paper is as follows. Section 5.1 provides the related

[1] `https://wiki.hyperledger.org/display/sawtooth`

work section. The proposed framework is in Section 5.2, followed by the case study in Section 5.3. Experiments are presented in Section 5.4.

5.1 Integration of logic-based technologies and blockchain

The adoption of logic-based technologies is presented as a new and growing perspective for the integration of IoT and blockchain technologies in business applications [48]. In [48], Smart contracts are leveraged to handle service invocations as transactions on the blockchain, mainly including both queries and updates. Inference services are mentioned for smart contract verification purposes, to ensure their execution does not violate the constraints associated to modelled business processes.

In [136], a blockchain-based framework uses smart contracts to mediate robot coalition formation, where both robot sensors/actuators and environmental parameters are exposed as resources annotated w.r.t. an ontology. The work mentions ontology matching methods for information exchange and a Knowledge Processor component for data processing, but no details are given about the provision of reasoning services. Moreover, albeit the proposal addresses the integration of blockchain with a robot ecosystem as a complex CPS, it does not include a discussion about computational load or energy consumption.

Ontology-based smart contract design in [59] supports traceability in supply chains, but the discussion lacks design or implementation details for inference services integrated in the blockchain platform. The ontology in [36] has been proposed for annotating transactions to facilitate searching for blockchain contents by semantic-enabled user agents. While the work mentions the potential of integrating blockchain in the context of *Industry 4.0* backed by IoT devices, the proposal does not specifically address possible deployments in IoT scenarios. A comparison of the mentioned proposals is in

Table 5.1: most of reviewed works propose the modeling of a domain ontology, but they either partially introduce or completely lack the detailed design of a semantic-based inference service layer. Moreover, this work presents a framework specifically tailored for IoT infrastructures, while reviewed works only discuss benefits of semantic-based integration in blockchain, without experimental evaluations or optimization techniques for scenarios comprising resource-constrained devices.

Table 5.1: Related work comparison (✓: supported, ✗: not supported, ✶: partial support)

	[48]	[136]	[59]	[36]	SeeSaw
Semantic-based Smart Contracts					
Ontology-based modelling	✓	✓	✓	✓	✓
Logic-based inference services	✶	✗	✶	✗	✓
IoT infrastructure					
Discussion	✗	✶	✗	✓	✓
Prototype	✗	✶	✗	✗	✓
Performance evaluation	✗	✗	✗	✗	✓
Transactions compression	✗	✗	✗	✗	✓

Other existing logical frameworks have been suggested for SC specification and execution, such as *Defeasible Reasoning* in [49] and *Linear Temporal Logic* in [48], which are already employed widely for the formalization of legal contracts and for model checking, respectively. In [117] the first semantic-based discovery approach for blockchain systems has been proposed. SeeSaw starts from that work. First of all, the Hyperledger *Iroha*[2] blockchain substratum is replaced with Sawtooth. From an architectural viewpoint, the consortium blockchain model has a satisfactory compliance with MEC infrastructures, as it is straightforward mapping to networks of business partners and resource/service marketplaces [71, 57]. Hyperledger Sawtooth [98] and the SeeSaw extension proposed here follow these settings. Its architecture is particularly suitable for MEC: it is based on decoupled components with well-defined resposabilities and customizable consensus, which can allow to

[2]Iroha Project: `https://www.hyperledger.org/projects/iroha`

choose algorithms that requires lower computational and energy resources than PoW and most BFT-like protocols [100]. Additionally, a more flexible discovery mechanism has been designed, allowing it to adapt to different requirements in terms of timeliness and completeness of the outcomes.

5.2 SeeSaw: SEmantic-Enhanced SAWtooth blockchain

The SeeSaw framework advances the Sawtooth blockchain by incorporating semantic knowledge representation and automated reasoning to optimize resource discovery and selection in distributed systems. First, the knowledge representation layer leverages OWL 2 annotations, providing a foundation for meaningful resource identification and interaction within the blockchain. This layer is essential for ensuring that resources are contextually relevant and easily searchable. Building on this, the automated reasoning component applies semantic matchmaking techniques, dynamically ranking resources by relevance and supporting efficient, informed service selection. Together, these layers establish SeeSaw as a powerful tool for managing resources in IoT and service-oriented environments, and the following sections detail each of these components and their contributions.

5.2.1 Blockchain framework architecture

The distributed architecture of Sawtooth has been expanded, as illustrated in Figure 5.1. This figure depicts the key elements of the infrastructure, each of which plays a specific role in the process of resource publication, discovery, and transaction validation within the system.

Producer (P): This component is responsible for publishing resources by registering them as assets on the blockchain, thereby making them available to Consumers.

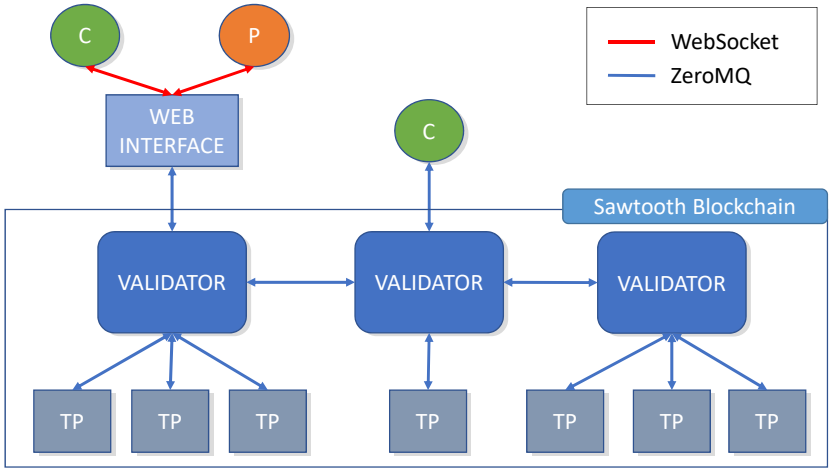


Figure 5.1: Framework architecture for the Sawtooth distributed system

Consumer (C): A Consumer component requests resources via a semantic-based process that involves three stages—Discovery, Explanation (optional), and Selection (detailed further in Section Section 5.2.2).

Web Interface (WI) (Web Interface): Acting as a REST gateway, the WI collects inputs from Producers and Consumers and routes them to a Validator. A single WI instance can handle multiple Producer and Consumer instances. Currently, the system uses the *WebSocket* protocol [38] to efficiently manage resource-constrained IoTs environment [93]; however, other point-to-point or mesh protocols for IoTs can be implemented as alternatives. The WI communicates with a single Validator using the *ZeroMQ* protocol[3], which facilitates distributed messaging. The WI aggregates requests from Producers and Consumers into a format suitable for processing by the Validator node and forwards them accordingly. As an alternative, Consumers can also directly send pre-processed inputs as ZeroMQ messages to a Validator node, as shown in Figure 5.1.

Transaction Processor (TP): The TP executes transactions at the network edge, implementing smart contracts. Sawtooth supports TPs in languages such as Python, C++, Go, Java, JavaScript, and Rust. In this framework, C++ is chosen for its efficiency. Each TP communicates with

[3]Project ZeroMQ: `http://zeromq.org/`

the Validator via ZeroMQ.

Validator: The Validator accesses the *radix Merkle tree* data structure of the blockchain, handling transaction requests from the WI. It distributes transaction processing among all connected TPs according to a manager/-worker model. If a TP is overloaded, it rejects transactions as invalid, and the WI then uses a backoff mechanism to wait before resubmission. Validators form a peer-to-peer network, communicating through a gossip protocol built on ZeroMQ for the replication of smart contract execution and the consensus protocol. Sawtooth adopts Proof of Elapsed Time (PoET), where a Validator is elected as leader through a lottery-like approach and is allowed to add a new block of transactions to the chain. Validator are implemented in Python.

Depending on target scenarios, several deployment configurations are possible:

- Producers and Consumers are components that require minimal computational resources, making them suitable for field-deployed nodes (*e.g.*, mobile devices or embedded in a CPS).
- WIs can be deployed either as dedicated devices at the network edge or integrated within Validators.
- Validators require high storage and bandwidth due to blockchain storage and consensus needs, making them ideal for hosting on-premises within the organization's core network.
- TPs have relatively low storage and bandwidth needs but require sufficient processing power. With optimized smart contracts, even low-cost single-board computers, such as *Raspberry Pi*, are viable platforms.

All exchanged messages are serialized in the *Protocol Buffers* format[4]. Transactions can be processed and validated individually or in *batches*, depending on request parameters. In a batch, transactions are sequentially

[4]Protocol Buffers: `https://protobuf.dev/`

dependent: if one fails, the following ones are skipped, and the entire batch is invalidated.

5.2.2 Knowledge representation and smart contracts

The proposed approach enables a semantic-based resource/service discovery in a IoT-oriented blockchain. As a consequence, the blockchain itself can be considered as a SOA, implementing resource registration, discovery, and selection as SCs, with distributed execution and consensus-based validation.

Discovery is based on semantic matchmaking of descriptions of a request and a set of resources, annotated as $\mathcal{ALN}$ DL concept expressions in OWL 2 w.r.t. a shared ontology. For each request-resource annotation pair, a *semantic relevance score* is computed from a combination of penalties induced by Concept Contraction and Concept Abduction, recalled in Section 3.2.1 [115]. This introduces a formally founded relevance ranking of all available resources on the blockchain that are described w.r.t. the same ontology as the request. The adopted inference services also return a logical *explanation* of discovery outcomes, which improves understandability of results w.r.t. other types of approaches. Transactions are recorded on the blockchain for robustness, traceability and accountability purposes. SOA primitives and corresponding SCs are reported hereafter.

A. Registration. Several resource domains can co-exist in the same blockchain. Each domain is associated to a different ontology, which provides the reference conceptual vocabulary to annotate resources. Every ontology is identified by a unique Uniform Resource Identifier (URI), as per OWL specifications. Each node can own resource instances, characterized by:

- a URI identifying the resource unambiguously;
- a semantic annotation in OWL language, modeling high-level descriptive information of resource features;
- the URI of the reference ontology;

- a set of data-oriented attributes stored as a key-value pair, allowing to integrate and extend logic-based inferences with application-specific and context-aware information processing.

In order to make a resource available for discovery and usage, the owner registers it as an asset on the blockchain storage. Ontologies are stored in the same way. Through this SC a blockchain-backed ubiquitous Knowledge Base (u-KB) [114] is thus obtained.

B. Resource discovery. IoT-based applications vary widely in functional and Service Level Agreement (SLA) requirements. Some use cases need quick-response resource discovery and best-effort recall is tolerated; this is typical of pervasive computing contexts. Other applications need an exhaustive search space exploration to guarantee that the best possible resources are found. In order to cope with the widest range of scenarios, SeeSaw includes two discovery modes, named *fast* and *full*. Supposing n annotated resources are associated with a domain ontology, any discovery request will generate up to $k + 1$ SC transactions, one for each piece originated from considering different sets of p resources from the whole set (*i.e.*, the ABox of the u-KB), with no overlap. Each of the first $k = \lfloor n/p \rfloor$ transactions will refer to exactly p resources and the last piece to the remaining $n - kp$ ones.

A transaction yields a *hit* if at least one of the resources in the piece has a semantic affinity higher than a given threshold, a *miss* otherwise. Hit resources are returned to the requester. In full discovery, all $k+1$ transactions are submitted simultaneously and the receiving Validator will take care of load balancing among Transaction Processors; the requester will receive all resources above the semantic relevance threshold. Furthermore, both hit and miss transactions are committed to the blockchain for traceability purposes. Conversely, fast discovery submits *clusters* of at most $c \leq k$ transactions at a time and it is limited by an overall *timeout*[5]. As soon as a cluster returns a hit or when the timeout expires, remaining clusters are not submitted. More-

[5]In the current implementation, cluster size and timeout length are static system-wide parameters. Dynamic adaptation w.r.t. network status and past performance is possible; studying strategies is left for future work.

over, in fast discovery miss transactions are invalidated and not committed to the chain, in order to reduce consensus stress on Validators. The adoption of clusters aims at a trade-off between a completely serial and parallel piece processing: the former minimizes blockchain load, but may increase the length and variability of hit latency, potentially incurring in more frequent timeouts; the latter ensures that a hit is found if it exists in the chain, but places a heavier computational burden and incurs in higher turnaround time.

Parameters of discovery SC are:

- mode: `fast` or `full`;
- URI of the reference ontology: this determines the resource domain as well as the vocabulary used to express both the request and the resources to be retrieved;
- semantic annotation of the request in OWL language, specifying desired resource features and constraints;
- maximum acceptable value for the i^{th} data-oriented attribute $a_{i_{max}}$ (*e.g.*, the maximum price the requester is willing to pay). Resources with at least one value higher than this threshold will be skipped from matchmaking (thus reducing computational overhead);
- minimum semantic relevance threshold s_{min}, as a floating-point number in the $[0, 1]$ interval, with a value of 1 corresponding to a full match and 0 to a complete mismatch (both rare situations in realistic scenarios); after matchmaking, resources with a relevance score below this threshold will not be returned, as deemed irrelevant to the requester.

C. Explanation. This SC is used to request a motivation for matchmaking outcomes. This may be useful for request revision and refinement [115] as well as *post-hoc* audit of the discovery process. Explanation reinforces the overall trust in the blockchain framework not only at data management level, but also at application level. Parameters of the SC are: (i) the request annotation and (ii) the URI of the discovered resource. SC result consists of

the semantic affinity score $0 \leq s_i \leq 1$ and concept expressions of G and K from Concept Contraction and of H from Concept Abduction.

D. Resource selection. After receiving all results exploiting full discovery, or a subset with fast discovery, the requester can select the best discovered resource with this SC. The complete registered resource representation is retrieved from the blockchain and returned. The proposal does not constrain resource fruition in any way, leaving application-specific details –such as interface endpoint or payment method– to resource annotations themselves.

5.3 Case study: IoT and blockchain for green mobility

A case study on the power management of PEVs in a Smart Grid has been developed to validate capabilities of the proposed framework in dynamic IoT scenarios. The illustrative example discussed hereafter is provided to clarify the proposal.

A smart PEV V_1 informs its driver that battery level is below 50%. The driver confirms that charging is needed. Blockchain assets available are charging slots offered in parking areas. High-level descriptive features of chargers are annotated w.r.t. a domain ontology (not reported due to dearth of space). Data-oriented attributes (*e.g.*, latitude and longitude of parking areas) are also included. The providers register chargers by invoking the Registration SC Section 5.2.2) through smart parking area WI node.

Since V_1's battery level is not critical, it looks for the best available charging service in the area. The vehicle acts as a Consumer on the blockchain, requesting the execution of a Discovery SC in *full* mode. Figure 5.2 reports

```
ChargeRequest EquivalentTo: V1 and (has_Charging_Rate_Per_Hour only (km
min 25)) and (has_Available_Power only (kWh min 12)) and (has_Dispatcher_-
Distance only (km max 50)) and (has_Base_Fee_Per_kWh only (cents max 50))

V1 EquivalentTo: Electric_Vehicle and (has_Output_Rate only (daW max 37))
and (has_Current only (Ampere max 16)) and (has_Voltage only (Volt max
230)) and (has_AC_Compatible_Plug only VDE-AR-E_2623-2-2Type2_Plug) and
(has_DC_Compatible_Plug only CCSCOMBO2_Plug)
```

Figure 5.2: Semantic annotation of vehicle request

on V_1's charge request[6] in OWL 2 *Manchester syntax* [46]: charging rate of at least 25 km per charging hour, minimum available energy of 12 kWh at charging station, maximum distance of 50 km between charging station and energy dispatcher, and maximum fee of 50 cents per kWh are requested. Car location and maximum distance between vehicle and charging station are specified as data-oriented request attributes. The request also defines the start time and duration of the charging service the vehicle is willing to reserve. The vehicle model adopts a standard *VDE-AR-E 2623-2-2* (Type 2) connector and receptacle for AC charging, incompatible with *SAEJ1772* (Type 1). Mutually incompatible *CHAdeMO*, *CCS Combo1* or *CCS Combo2* connectors exist for fast DC charging. Vehicle V_1 supports CCS Combo2 plug only.

The Discovery request is divided in pieces through the Web Interface and forwarded to a Validator. In the reference scenario, parking areas play the role of Validators because they have no mobility issues or energy supply limits, so they can run the consensus protocol reliably. The smart vehicles parked or in charging within parking areas work as Transaction Processors, since they have enough on-board computing capabilities; they receive a compensation for their work in the form of savings on current or future charging fees. However, as Figure 5.3 highlights, heterogeneous devices –even embedded ones– can be also exploited as Transaction Processors, thanks to

[6]For the sake of compactness, concept expressions are simplified w.r.t. the ones actually used for the case study. In particular, for each universal restriction of the form `role only concept`, the reader should assume there is a corresponding `role some owl:Thing` existential restriction in conjunction. Reported matchmaking results refer to the complete expressions.

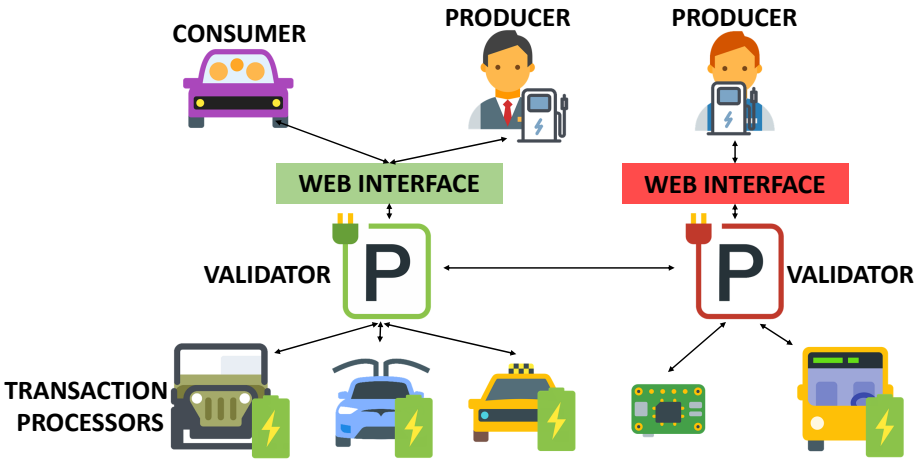


Figure 5.3: Roles in the Smart Mobility case study

the multiplatform IoT-oriented support of Sawtooth. The Discovery SC pre-filters available resources on location and temporal context attributes: resources outside the query area or unavailable at the specified time are discarded. The chain resources satisfying the above constraints are shown in Figure 5.4. Full discovery based on semantic-matchmaking returns a ranked list of the three resources w.r.t. the request, where *BuildingB_Slot1* is the resource having the highest semantic affinity score, despite the low charging rate per hour. In detail, the overall match score s for resource S_i is computed by means of the formula:

$$s(S_i) = \begin{cases} 1 - p(S_i), & 1 - p(S_i) > s_{min} \\ 0, & otherwise \end{cases} \tag{5.1}$$

with the semantic penalty function p computed as:

$$p(S_i) = \frac{w \cdot penalty_c(S_i) + (1 - w) \cdot penalty_a(S_i)}{penalty_{max}} \tag{5.2}$$

where $penalty_c$ is the penalty calculated by Concept Contraction between the request R and each resource annotation S_i, while $penalty_a$ is the penalty value of the Concept Abduction procedure between the consistent part K of the request R and S_i. The value of p is normalized w.r.t. $penalty_{max}$, *i.e.*, the maximum possible semantic distance (which is the one between R and the most generic concept $\top$, and depends only on axioms in the reference

```
ChargingStationA_Slot1 EquivalentTo: (has_AC_Compatible_Plug only
SAEJ1772-2009Type1_Plug) and (has_EVSE_Profile only AC_Level_2) and (has_-
Available_Power only (kWh exactly 500)) and (has_Base_Fee_Per_kWh only
(cents exactly 55)) and (has_Charging_Rate_Per_Hour only (km exactly 45))
and (has_Dispatcher_Distance only (km exactly 60))

ChargingStationA_Slot2 EquivalentTo: (has_DC_Compatible_Plug only
CCSCOMBO2_Plug) and (has_EVSE_Profile only DC_Level_1) and (has_-
Available_Power only (kWh exactly 500)) and (has_Base_Fee_Per_kWh only
(cents exactly 85)) and (has_Charging_Rate_Per_Hour only (km exactly 200))
and (has_Dispatcher_Distance only (km exactly 60))

BuildingB_Slot1 EquivalentTo: (has_AC_Compatible_Plug only VDE-AR-E_-
2623-2-2Type2_Plug) and (has_EVSE_Profile only AC_Level_1) and (has_-
Available_Power only (kWh exactly 100)) and (has_Base_Fee_Per_kWh only
(cents exactly 45)) and (has_Charging_Rate_Per_Hour only (km exactly 20))
and (has_Dispatcher_Distance only (km exactly 25))
```

Figure 5.4: Semantic annotations of available resources

domain ontology [113]). The parameter w can be set in the $]0, 1[$ interval and it determines the relative weight of explicitly conflicting elements in S_i w.r.t. R; for the proposed case study $w = 0.8$ has been selected after preliminary tests.

ChargingStationA_Slot1 and *ChargingStationA_Slot2* are both too expensive and too far, with *ChargingStationA_Slot2* being the most expensive one. *ChargingStationA_Slot1* also equips an incompatible charging plug. V_1 has the option to retrieve motivations for the discovery result by means of the Explanation SC. Then the Selection SC allows selecting the best service: V_1 *pays the charging fee and reserves its use for the booked time.*

5.4 Experiments

The SeeSaw framework has been evaluated through extensive experimental testing to assess its effectiveness and scalability in integrating Semantic Web technologies into the Sawtooth blockchain. The experiments involved prototypical deployments with varying network sizes, enabling the measurement

of key performance metrics such as request turnaround time, hit ratio, and resource utilization. By analyzing both fast and full discovery scenarios, the results demonstrate the framework's ability to maintain efficient performance and scalability, even in large-scale deployments. These findings highlight the feasibility of the proposed approach for IoT and pervasive environments. The following sections detail the experimental setup, metrics, and results.

5.4.1 Experimental setup

A prototypical version of the proposed SeeSaw framework has been implemented to assess effectiveness and scalability of the approach. The overall architecture is depicted in Figure 5.5, including the components described in what follows.

Semantic Transaction Processor (STP): handles semantic transactions submitted to Validator nodes. The STP has been implemented using the Sawtooth Software Development Kit (SDK) for C++ (`https://github.com/hyperledger/sawtooth-sdk-cxx`). The *Tiny-ME* reasoning and matchmaking engine [115] is used to execute inference tasks. The Address Mapper component generates encoded string addresses, corresponding to each input and output data. The State Viewer retrieves data from the radix Merkle tree managed by the Validator.

Semantic Gateway (SG): implements the Web interface role in the Sawtooth architecture. The Request Handler component receives client requests. The Transaction Manager encapsulates semantic transactions into batches, which are submitted to the Validator. The Event Listener is notified when results are committed.

The Semantic Transaction Family (STF) has been introduced, declaring overall rules about how semantic annotations are addressed and stored. The adopted addressing scheme is shown in Figure 5.6. Each 70 hexadecimal digits long address contains the following elements: (i) Namespace, which uniquely identifies STF; (ii) Type, representing the data category (ontology,

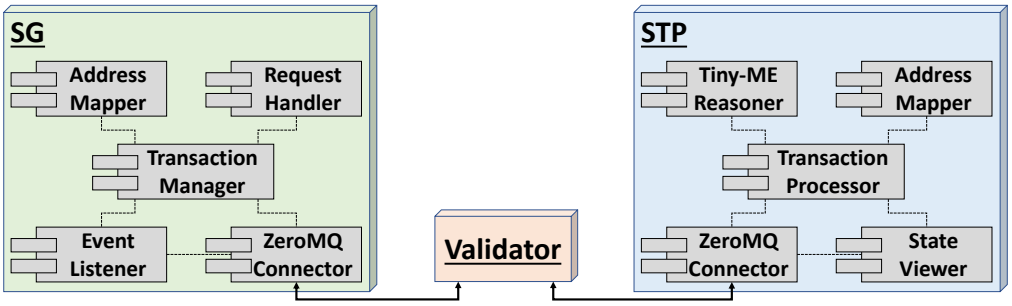


Figure 5.5: Prototype component diagram

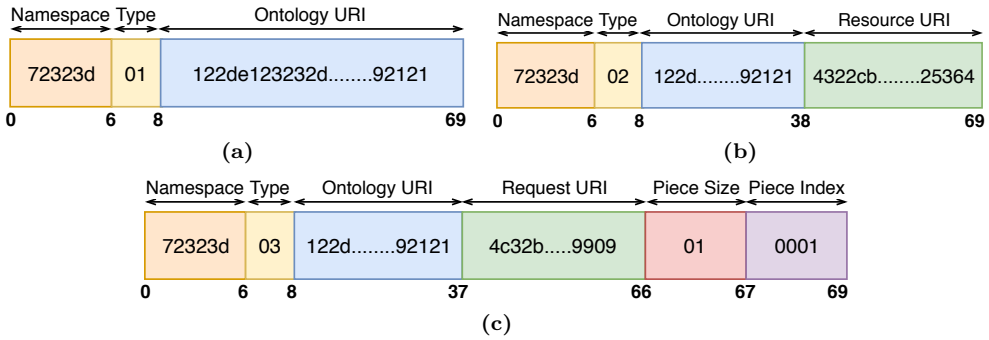


Figure 5.6: Merkle tree address structure: a) ontology, b) resource; c) semantic task result

annotation, semantic task result); (iii) hash values of ontology URI and resource URI; (iv) Piece Size and Index corresponding to the discovery result subset. The Protocol Buffer serialization is adopted to encapsulate transaction payload, represented in Figure 5.7. The main elements are: (i) Semantic Task code, which specifies the SC to be executed; (ii) reference ontology URI; (iii) discovery threshold; (iv) Piece Size and Index, which allows to adapt discovery range to network traffic load. One or more Semantic Data structure are added to transaction payload, containing resource type, resource URI, encoding format and serialized annotation data.

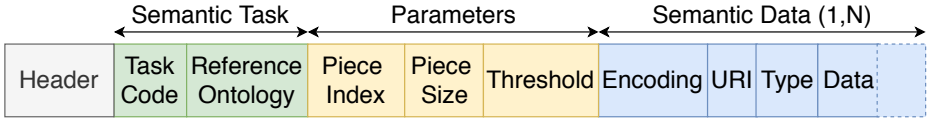


Figure 5.7: Semantic Transaction structure

5.4.2 Results and analysis

Small, medium, and large-scale scenarios have been defined for a quantitative performance analysis, involving 100, 200, and 400 nodes, respectively. Nodes are divided into three groups: *Clients (20%)*, which act as workload generators (e.g., Producers of annotated resources registered in the blockchain and Consumers requesting resources); *Validators (20%)*; and *Transaction Processors (60%)*. The experimental parameters are set as follows: (i) a duration of 300 s, excluding an initial resource registration phase (excluded due to its lightweight nature); (ii) each Producer generates 6 random annotations; (iii) Consumers generate a new random request every 8 s; and (iv) a minimum semantic affinity threshold of 0.85. Each scenario is tested with all combinations of the following parameters: (i) discovery timeout (T) set to 2, 6, or 10 s; (ii) piece size (P) (Section 5.2.2) set to 10, 30, or 50 resources; and (iii) either *fast* or *full* discovery. Each test uses the average result from two runs.

The testbed configuration is as follows: (i) each prototype component is compiled as a *Docker* image; (ii) each node is executed as a container instance of its respective compiled image; (iii) a Docker *Swarm mode* cluster is deployed across 10 *VMware vSphere* virtual machines on two server blades (Intel Xeon E5-2650 v3 CPU –8 cores/16 threads at 2.30 GHz– and 96 GB of RAM per blade, Ubuntu 16.04 64bit OS) with an overlay network enabling node communication; (iv) experiment execution is managed via the Docker API SDK[7].

The performance metrics measured include:

- Mean turnaround time and standard deviation for request fulfillment, corresponding to the sum of transaction latencies for Discovery, Explanation, and Selection, with individual transaction latencies reported for each of the three SC invocations.
- Mean hit ratio, i.e., the percentage of requests retrieving at least one

[7] `https://github.com/atlassian-forks/docker-client`

resource that satisfies semantic relevance constraints within the given timeout.

- Mean and standard deviation of memory and CPU load for Transaction Processor and Validator nodes.

No constraints were imposed on RAM or CPU in the Docker configuration to prevent any interference with other performance metrics.

The primary purpose of these metrics is to evaluate the feasibility of integrating a semantic service layer into the SeeSaw blockchain infrastructure, designed for IoT and pervasive environments. Hence, the experimental focus is on client-side performance metrics related to semantic service execution rather than platform-specific metrics (e.g., transaction throughput), which are less pertinent to user-perceived semantic services. Given the SeeSaw architecture's varied roles and device types, infrastructure load was assessed by measuring memory and CPU usage for Validators and Transaction Processors, the most critical nodes.

Fast discovery. The results for fast discovery are illustrated in Figure 5.8. In the scenario with 100 nodes (Figure 5.8a), mean turnaround time remains low, with discovery, explanation, and selection services completing in under 1 s. Higher timeout values (T) reduce standard deviation and increase hit ratio variability, peaking at T=10 and P=50. Generally, longer timeouts yield more successful transactions but also introduce greater network load variability, explaining why hit ratio does not consistently increase with timeout values but shows higher variability due to increased load.

In the 200-node scenario (Figure 5.8b), mean turnaround time peaks for P=50 and T=10, with increased standard deviation and minimum hit ratio occurring for the same values. The 400-node scenario (Figure 5.8c) shows that mean turnaround time peaks for P=30 and T=10, with generally higher standard deviation than in smaller deployments. For each piece size, hit ratio reaches its maximum at the highest timeout value, as expected.

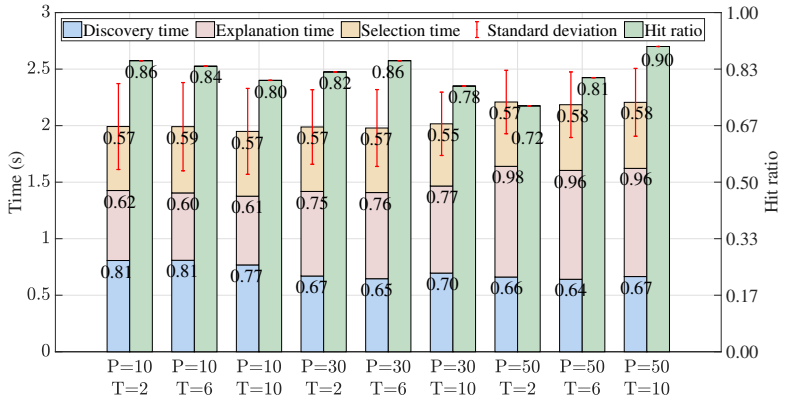
Across all scenarios, the time required for discovery is similar to that for

the explanation and selection phases, indicating that consensus is the longest task, while transaction processing remains relatively brief. This suggests that the performance impact of the added semantic SOA layer is minimal. Additionally, the high variability in hit ratio across scenarios may stem from the testbed CPU's limited compatibility with the PoET consensus algorithm, potentially adding CPU load to containerized environments.

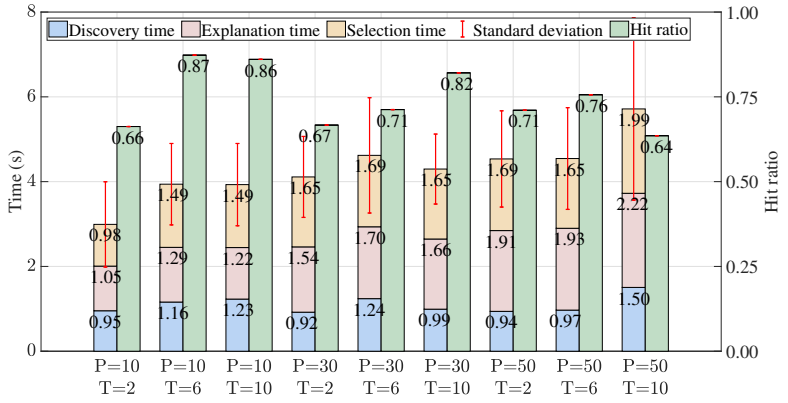
As shown in Figure 5.9, average CPU and RAM usage per Validator increases with scenario size, with slight influence from P values. Figure 5.10 shows that CPU load for TPs also rises with larger scenarios but remains below 2% on average; memory consumption remains within acceptable limits.

Full discovery. Figure 5.11a shows that average turnaround time is closely related to the number of nodes and registered resources. The best results are obtained in the 100 nodes scenario. Turnaround time decreases when P is larger, due to fewer committed transactions. Moreover, the standard deviation is higher for smaller P, because discovery requests generate more blocks to be committed by Validators, thus stressing the consensus protocol. Figure 5.11b reports on the average CPU and RAM usage per Validator: also in this case higher values and higher variance are found for larger scenarios, with practically no influence of Piece size. Figure 5.11c shows average values for Transaction Processors, denoting similar trends.

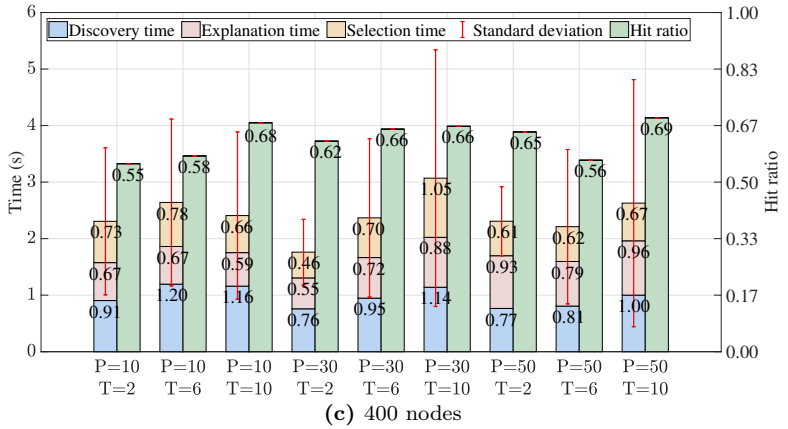
The experimental results support the feasibility of the proposed approach evidencing a limited overhead of semantic matchmaking. The Sawtooth architecture has allowed a more extensive experimental campaign on a larger number of nodes with respect to the similar semantic-based approach in [117]. Fast discovery has stable times, best-effort hit ratio and graceful degradation at larger scales, while full discovery guarantees the best semantic matchmaking results with a linear turnaround time increase w.r.t. network scale. Overall, the SeeSaw architecture appears as amenable to Industrial Internet of Things (IIoT) and MEC scenarios, where semantic blockchain load can be sustained by Validators associated with relatively large numbers of lightweight mobile and embedded devices, acting as semantic Transaction Processors at the network edge and in the field.



(a) 100 nodes

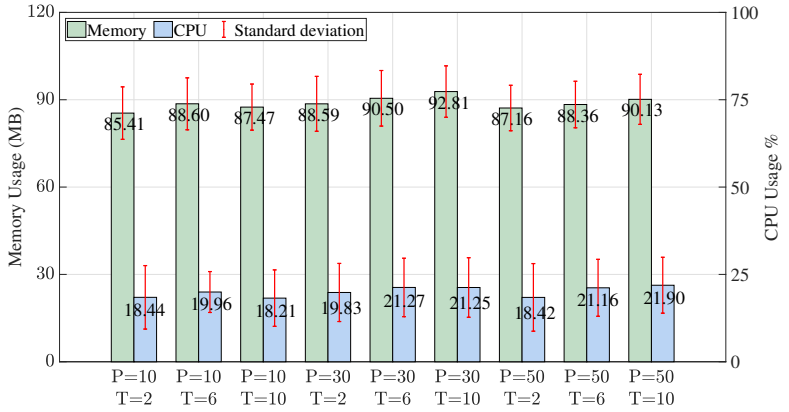


(b) 200 nodes

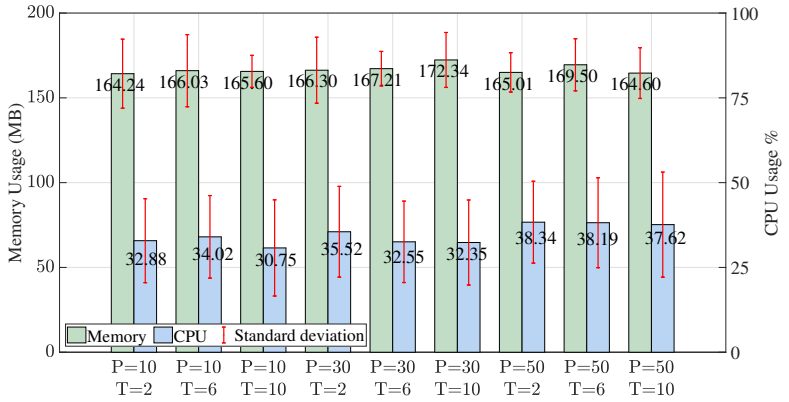


(c) 400 nodes

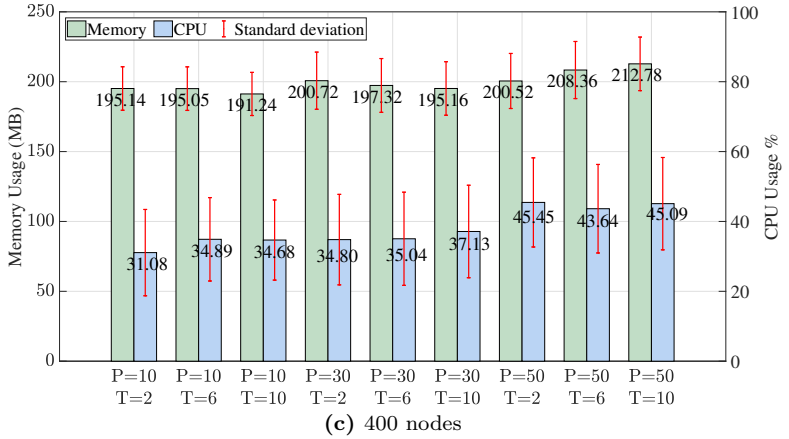
Figure 5.8: Fast discovery turnaround time and hit ratio; P: piece size; T: time-out (s)



(a) 100 nodes

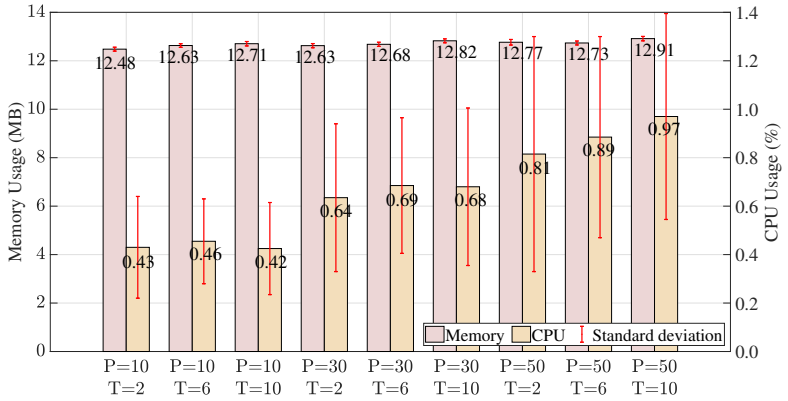


(b) 200 nodes

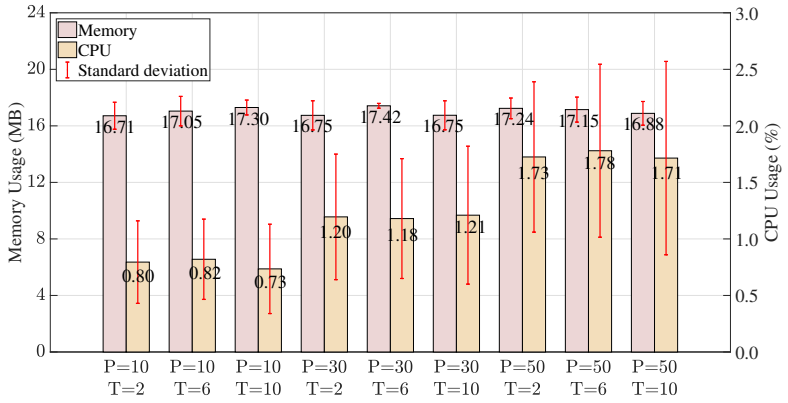


(c) 400 nodes

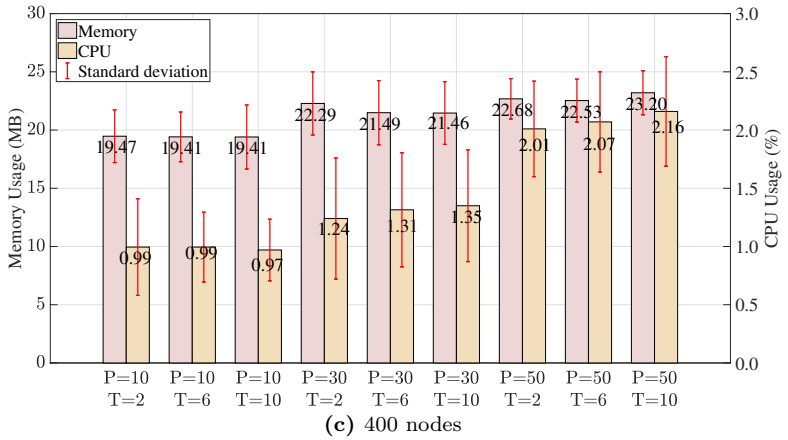
Figure 5.9: Fast discovery Validator memory and CPU usage; P: piece size; T: timeout (s)



(a) 100 nodes

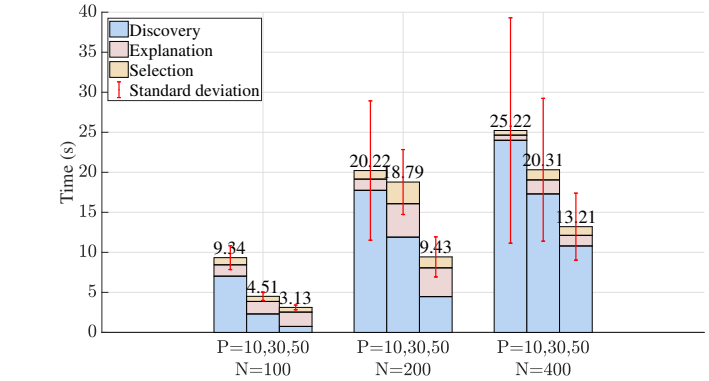


(b) 200 nodes

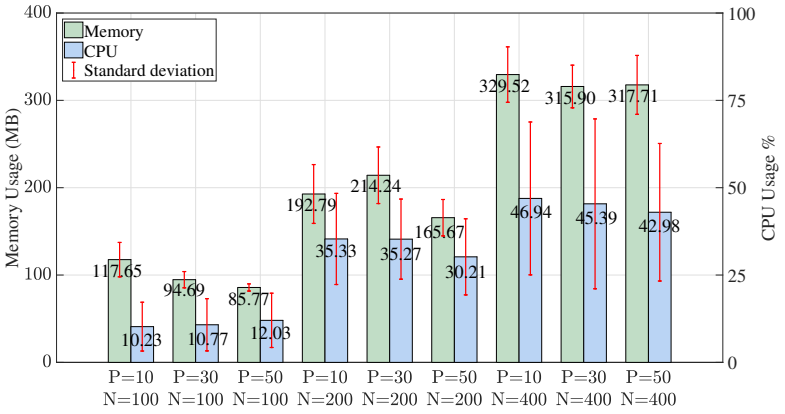


(c) 400 nodes

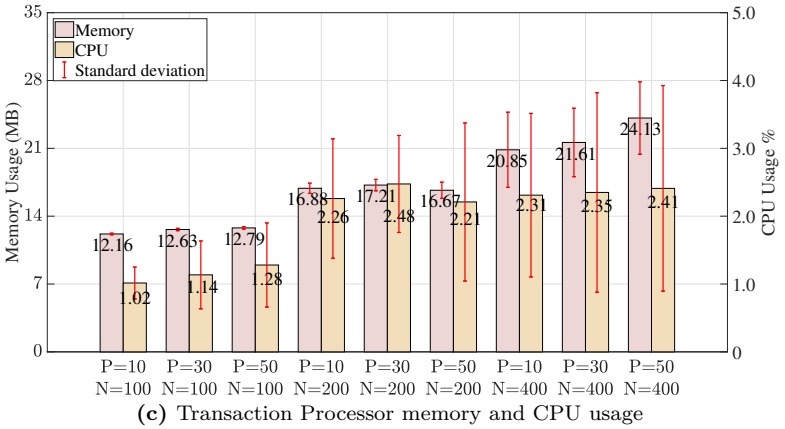
Figure 5.10: Fast discovery Transaction Processor memory and CPU usage; P: piece size; T: timeout (s)



(a) Turnaround time



(b) Validator memory and CPU usage



(c) Transaction Processor memory and CPU usage

Figure 5.11: Full discovery performance; P: piece size; N: nodes

Conclusion and perspectives

The integration of *DLTs* with *CPSs* forms a key part of the ongoing evolution toward more secure, transparent, and intelligent systems. This dissertation has explored the design and development of a scalable and secure architecture for pervasive CPSs, leveraging the capabilities of DLTs to enhance trust and security. By combining DLTs with SWoT, the research has addressed the unique challenges posed by decentralized environments, offering a novel approach to managing distributed resources in real time.

The proposed framework demonstrates that it is feasible to deploy DLT platforms in resource-constrained CPS environments, providing both transparency and reliability without compromising performance. The integration of semantic technologies allows for the enrichment of data, enabling devices to interact intelligently and autonomously. This combination paves the way for more robust and adaptive systems that can operate effectively across various scales, from powerful cloud infrastructures to small, resource-constrained field devices.

The developed stack, which includes novel components for knowledge representation and automated reasoning, effectively bridges the gap between theoretical advancements in SWoT and practical application in real-world CPS scenarios. The architectural choices made in this dissertation—specifically the use of DLTs to support decentralized interactions and the adoption of semantic technologies to enhance reasoning capabilities—have been validated through extensive case studies, showing their relevance and applicability in various industrial contexts.

Looking forward, there are several directions for future research. One avenue involves expanding the capabilities of the proposed system to support

a broader range of reasoning tasks and exploring more advanced consensus mechanisms to further optimize the performance of DLTs in CPS environments. Additionally, there is potential to integrate advanced machine learning models into the system, enabling it to predict and adapt to changing conditions in real time, further enhancing its intelligence and autonomy.

The transition from academic research to industrial application is crucial for the success of the proposed architecture. Future efforts will focus on improving the Technology Readiness Level (TRL) of the developed solutions, ensuring they meet the demands of various vertical sectors, such as manufacturing, transportation, and smart cities. Collaborations with industry and academic partners will be essential in refining and scaling the technologies, facilitating their integration into existing systems, and demonstrating their real-world value.

Community engagement and collaboration are also key factors in advancing the work presented in this dissertation. The public release of the developed tools will foster further experimentation and feedback from both researchers and industry practitioners. This approach will ensure that the solutions continue to evolve and remain relevant in the ever-changing landscape of CPSs and IoT technologies.

In conclusion, this thesis presents a foundation for the adoption of semantic-based DLTs within CPSs, creating new possibilities for intelligent and decentralized systems. The proposed architecture demonstrates how the combination of DLT and SWoT can enhance security, adaptability, and autonomy in cyber-physical systems. This work contributes to advancing our interaction with these systems, fostering the development of more efficient and resilient infrastructures in an increasingly connected world.

Bibliography

[1] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. Mobile Edge Computing: A Survey. *IEEE Internet of Things Journal*, 5(1):450–465, 2017.

[2] Nora Abdelmageed and Sirko Schindler. JenTab: Matching Tabular Data to Knowledge Graphs. In *Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Workshop, International Semantic Web Conference*, pages 40–49, 2020.

[3] Laith Abualigah, Ali Diabat, Putra Sumari, and Amir H Gandomi. Applications, deployments, and integration of Internet of Drones (IoD): a review. *IEEE Sensors Journal*, 21(22):25532–25546, 2021.

[4] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. The K-means algorithm: a comprehensive survey and performance evaluation. *Electronics*, 9(8):1295, 2020.

[5] Julián Arenas-Guerrero, David Chaves-Fraga, Jhon Toledo, María S Pérez, and Oscar Corcho. Morph-KGC: Scalable knowledge graph materialization with mapping partitions. *Semantic Web*, pages 1–20, 2022.

[6] Peter Arthurs, Lee Gillam, Paul Krause, Ning Wang, Kaushik Halder, and Alexandros Mouzakitis. A Taxonomy and Survey of Edge Cloud Computing for Intelligent Transportation Systems and Connected Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6206–6221, 2022.

[7] Franz Baader, Diego Calvanese, Deborah L McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Hand-*

book: Theory, Implementation and Applications. Cambridge University Press, 2007. 2nd Ed.

[8] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems. *Applied Intelligence*, 4(2):109–132, 1994.

[9] Tim Berners-Lee, Roy T. Fielding, and Larry M Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, January 2005.

[10] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):28–37, 2001.

[11] Nicola Bicocchi, Giacomo Cabri, Federica Mandreoli, and Massimo Mecella. Dealing with data and software interoperability issues in digital factories. In *Proceedings of the 25th ISPE Inc. International Conference on Transdisciplinary Engineering*, volume 7, page 13. IOS Press, 2018.

[12] Ivano Bilenchi, Filippo Gramegna, Giuseppe Loseto, Saverio Ieva, Floriano Scioscia, and Michele Ruta. Cowl: Pushing OWL 2 over the Edge. *Elsevier Internet of Things Journal*, 29(101439):1–20, Jan 2025.

[13] Ivano Bilenchi, Floriano Scioscia, and Michele Ruta. Cowl: A Lightweight OWL Library for the Semantic Web of Everything. In Agapito *et al.*, editor, *Current Trends in Web Engineering. ICWE 2022.*, pages 100–112, Cham, 2023. Springer.

[14] Ivano Bilenchi, Arnaldo Tomasino, Filippo Gramegna, Saverio Ieva, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. Knowledge Representation and Reasoning for Unmanned Aerial Vehicle Intelligence. In *7th Italian Workshop on Embedded Systems (IWES 2022)*, 2022.

[15] Luiz Bittencourt, Roger Immich, Rizos Sakellariou, Nelson Fonseca, Edmundo Madeira, Marilia Curado, Leandro Villas, Luiz DaSilva, Craig Lee, and Omer Rana. The Internet of Things, Fog and Cloud

continuum: Integration and challenges. *Internet of Things*, 3-4:134–155, 2018.

[16] Martin Björklund. The YANG 1.1 Data Modeling Language. RFC 7950, August 2016.

[17] Martin Björklund and Lou Berger. YANG Tree Diagrams. RFC 8340, March 2018.

[18] A. Borgida. Description Logics in Data Management. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):671–682, 1995.

[19] R. Brachman and H. Levesque. The Tractability of Subsumption in Frame-based Description Languages. In *4th National Conference on Artificial Intelligence (AAAI-84)*, pages 34–37. Morgan Kaufmann, 1984.

[20] Tim Bray. RFC 8259: The JavaScript object notation (JSON) data interchange format, 2017.

[21] Héctor Cañas, Josefa Mula, Manuel Díaz-Madroñero, and Francisco Campuzano-Bolarín. Implementing Industry 4.0 principles. *Computers & industrial engineering*, 158:107379, 2021.

[22] Keyan Cao, Yefan Liu, Gongjie Meng, and Qimeng Sun. An Overview on Edge Computing Research. *IEEE Access*, 8:85714–85728, 2020.

[23] Kun Cao, Shiyan Hu, Yang Shi, Armando Walter Colombo, Stamatis Karnouskos, and Xin Li. A Survey on Edge and Edge-Cloud Computing Assisted Cyber-Physical Systems. *IEEE Transactions on Industrial Informatics*, 17(11):7806–7819, 2021.

[24] Ryan Chard, Yadu Babuji, Zhuozhao Li, Tyler Skluzacek, Anna Woodard, Ben Blaiszik, Ian Foster, and Kyle Chard. Funcx: A federated function serving fabric for science. In *Proceedings of the 29th International symposium on high-performance parallel and distributed computing*, pages 65–76, 2020.

[25] Kang Leng Chiew, Choon Lin Tan, KokSheik Wong, Kelvin S.C. Yong, and Wei King Tiong. A New Hybrid Ensemble Feature Selection Framework for Machine Learning-Based Phishing Detection System. *Information Sciences*, 484(C):153–166, may 2019.

[26] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and Smart Contracts for the Internet of Things. *IEEE Access*, 4:2292–2303, 2016.

[27] David D. Clark, Craig Partridge, J. Christopher Ramming, and John T. Wroclawski. A knowledge plane for the internet. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, page 3–10, New York, NY, USA, 2003. Association for Computing Machinery.

[28] Souripriya Das, Seema Sundara, and Richard Cyganiak. R2RML: RDB to RDF Mapping Language, September 2012. `https://www.w3.org/TR/r2rml/`.

[29] Diego De Una, Nataliia Rümmele, Graeme Gange, Peter Schachte, and Peter J Stuckey. Machine Learning and Constraint Programming for Relational-To-Ontology Schema Mapping. In *International Joint Conference on Artificial Intelligence*, pages 1277–1283, 2018.

[30] Thomas Delva, Julián Arenas-Guerrero, Ana Iglesias-Molina, Oscar Corcho, David Chaves-Fraga, and Anastasia Dimou. RML-star: A declarative mapping language for RDF-star generation. In *ISWC2021, the International Semantic Web Conference*, volume 2980. CEUR, 2021.

[31] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y Zomaya. Edge Intelligence: the Confluence of Edge Computing and Artificial Intelligence. *IEEE Internet of Things Journal*, 7(8):7457–7469, 2020.

[32] Tommaso Di Noia, Eugenio Di Sciascio, and Francesco M. Donini. Semantic Matchmaking as Non-Monotonic Reasoning: A Description

Logic Approach. *Journal of Artificial Intelligence Research (JAIR)*, 29:269–307, 2007.

[33] Anastasia Dimou, Miel Vander Sande, Pieter Colpaert, Ruben Verborgh, Erik Mannens, and Rik Van de Walle. RML: A generic language for integrated rdf mappings of heterogeneous data. In *Workshop on Linked Data on the Web, 23rd International World Wide Web Conference*, pages 1–5, 2014.

[34] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in description logics. *Principles of Knowledge representation*, 1:191–236, 1996.

[35] Martin Duerst and Michel Suignard. Internationalized Resource Identifiers, January 2005. `https://rfc-editor.org/rfc/rfc3987.txt`.

[36] Matthew English, Sören Auer, and John Domingue. Block Chain Technologies & The Semantic Web: A Framework for Symbiotic Development. In *Computer Science Conference for University of Bonn Students*, pages 47–61, 2016.

[37] Rob Enns, Martin Bjorklund, and Juergen Schoenwaelder. NETCONF configuration protocol, Dec 2006.

[38] I. Fette and A. Melnikov. The WebSocket Protocol. Internet Requests for Comments, December 2011.

[39] Maurizio Giacobbe, Francesco Alessi, Angelo Zaia, Antonio Puliafito, et al. Arancino.cc: an open hardware platform for urban regeneration. *INTERNATIONAL JOURNAL OF SIMULATION & PROCESS MODELLING*, 15(4):343–357, 2020.

[40] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT press Cambridge, 2016.

[41] Andreas Grafberger, Mohak Chadha, Anshul Jindal, Jianfeng Gu, and Michael Gerndt. FedLess: Secure and Scalable Federated Learning Using Serverless Computing. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 164–173, 2021.

[42] Filippo Gramegna, Arnaldo Tomasino, Saverio Ieva, Ivano Bilenchi, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. RideMATCHain: a Semantic-enhanced Blockchain Marketplace for Ridesharing. In *8th Italian Conference on ICT for Smart Cities And Communities (I-CiTies 2022)*, 2022.

[43] R. V. Guha, Dan Brickley, and Steve Macbeth. Schema.Org: Evolution of Structured Data on the Web. *Commun. ACM*, 59(2):44–51, jan 2016.

[44] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Synthesis Lectures on the Semantic Web. Morgan & Claypool Publishers, 2011.

[45] Matthew Horridge and Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. *Semantic Web*, 2(1):11–21, 2011.

[46] Matthew Horridge and Peter Patel-Schneider. OWL 2 Web Ontology Language Manchester Syntax (Second Edition), December 2012. `http://www.w3.org/TR/owl2-manchester-syntax`.

[47] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, May 2014. https://www.w3.org/Submission/SWRL/.

[48] Richard Hull, Vishal S Batra, Yi-Min Chen, Alin Deutsch, Fenno F Terry Heath III, and Victor Vianu. Towards a shared ledger business collaboration language based on data-aware processes. In *International Conference on Service-Oriented Computing*, pages 18–36. Springer, 2016.

[49] Florian Idelberger, Guido Governatori, Régis Riveret, and Giovanni Sartor. Evaluation of logic-based smart contracts for blockchain systems. In *International Symposium on Rules and Rule Markup Languages for the Semantic Web*, pages 167–183. Springer, 2016.

[50] IEC PAS 63088. *Smart manufacturing – Reference architecture model industry 4.0 (RAMI4.0).* IEC – International Electrotechnical Commission, 2017.

[51] Saverio Ieva, Ivano Bilenchi, Filippo Gramegna, Floriano Scioscia, Michele Ruta, and Giuseppe Loseto. Integrating AI-based fleet optimization with mixed reality to enhance last-mile delivery. In *10th Italian Conference on ICT for Smart Cities and Communities (I-CiTies 2024)*, 2024.

[52] Saverio Ieva, Davide Loconte, Giuseppe Loseto, Michele Ruta, Floriano Scioscia, Davide Marche, and Marianna Notarnicola. A Retrieval-Augmented Generation Approach for Data-Driven Energy Infrastructure Digital Twins. *Smart Cities*, 7(6):3095–3120, 2024.

[53] Saverio Ieva, Davide Loconte, Agnese Pinto, Filippo Gramegna, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. Semantic-enhanced Network Orchestration to Support Smart City Services. In *8th Italian Conference on ICT for Smart Cities And Communities (I-CiTies 2022)*, 2022.

[54] Saverio Ieva, Davide Loconte, Agnese Pinto, Floriano Scioscia, and Michele Ruta. Semantic-Based Decision Support for Network Management and Orchestration. In *Current Trends in Web Engineering*, pages 125–136, Cham, 2023. Springer Nature Switzerland.

[55] Krzysztof Janowicz, Armin Haller, Simon JD Cox, Danh Le Phuoc, and Maxime Lefrançois. SOSA: A lightweight ontology for sensors, observations, samples, and actuators. *Journal of Web Semantics*, 56:1–10, 2019.

[56] Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Dmitriy Zheleznyakov, Ian Horrocks, Christoph Pinkel, Martin G Skjæveland, Evgenij Thorstensen, and Jose Mora. BootOX: Practical mapping of RDBs to OWL 2. In *Proceedingsof the 14th International Semantic Web Conference, Part II*, pages 113–132. Springer, 2015.

[57] Jiawen Kang, Rong Yu, Xumin Huang, Sabita Maharjan, Yan Zhang, and Ekram Hossain. Enabling localized peer-to-peer electricity trading among plug-in hybrid electric vehicles using consortium blockchains. *IEEE Transactions on Industrial Informatics*, 13(6):3154–3164, 2017.

[58] Andrew Kennedy, Michele Southall, Gena Morgan, Ken Traub, et al. EPC Information Services (EPCIS) Specification, May 2014. Available: `http://www.gs1.org/epcis/epcis/1-1`.

[59] Henry M Kim and Marek Laskowski. Toward an ontology-driven blockchain design for supply-chain provenance. *Intelligent Systems in Accounting, Finance and Management*, 25(1):18–27, 2018.

[60] A Kojukhov, AM de Nicolas, B Chatras, D Druta, D Gassanov, M Brunner, M Brenner, S Li, T Nguyenphu, U Rauschenbach, et al. Network functions virtualisation (nfv) release 2; protocols and data models; vnf package specification. GS NFV-SOL 004 V2. 3.1. *Group specification, ETSI*, 2017.

[61] Kari Korpela, Jukka Hallikas, and Tomi Dahlberg. Digital Supply Chain Transformation toward Blockchain Integration. In *Proc. of 50th Hawaii International Conference on System Sciences*, pages 4182–4191, 2017.

[62] Mirko Koscina, Marius Lombard-Platet, and Claudia Negri-Ribalta. A blockchain-based marketplace platform for circular economy. In *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pages 1746–1749, 03 2021.

[63] Nikita Kotsehub, Matt Baughman, Ryan Chard, Nathaniel Hudson, Panos Patros, Omer Rana, Ian Foster, and Kyle Chard. FLoX: Federated Learning with FaaS at the Edge. In *2022 IEEE 18th International Conference on e-Science (e-Science)*, pages 11–20, 2022.

[64] Jean-Baptiste Lamy. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artificial Intelligence in Medicine*, 80:11–28, 2017.

[65] J. Lee, Behrad Bagheri, and Hung-An Kao. A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing letters*, 3:18–23, 2015.

[66] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

[67] Jiewu Leng, Weinan Sha, Baicun Wang, Pai Zheng, Cunbo Zhuang, Qiang Liu, Thorsten Wuest, Dimitris Mourtzis, and Lihui Wang. Industry 5.0: Prospect and retrospect. *Journal of Manufacturing Systems*, 65:279–295, 2022.

[68] Ladislav Lhotka. JSON Encoding of Data Modeled with YANG. RFC 7951, August 2016.

[69] Lei Li and Ian Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. *International Journal of Electronic Commerce*, 8(4), 2004.

[70] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[71] Zhetao Li, Jiawen Kang, Rong Yu, Dongdong Ye, Qingyong Deng, and Yan Zhang. Consortium blockchain for secure energy trading in Industrial Internet of Things. *IEEE Transactions on Industrial Informatics*, 14(8):3690–3700, 2018.

[72] Thorsten Liebig, Marko Luther, Olaf Noppens, and Michael Wessel. OWLlink. *Semantic Web*, 2(1):23–32, January 2011.

[73] Hong Liu, Yan Zhang, and Tao Yang. Blockchain-Enabled Security in Electric Vehicles Cloud and Edge Computing. *IEEE Network*, 32(3):78–83, 2018.

[74] Jixiong Liu, Yoan Chabot, Raphaël Troncy, Viet-Phi Huynh, Thomas Labbé, and Pierre Monnin. From tabular data to knowledge graphs: A survey of semantic table interpretation tasks and methods. *Journal of Web Semantics*, page 100761, 2022.

[75] Jixiong Liu and Raphaël Troncy. DAGOBAH: an end-to-end context-free tabular data semantic annotation system. In *Semantic Web Challenge on Tabular Data to Knowledge Graph Matching Workshop, International Semantic Web Conference*, pages 41–48, 2019.

[76] Mengting Liu, F Richard Yu, Yinglei Teng, Victor CM Leung, and Mei Song. Computation offloading and content caching in wireless blockchain networks with mobile edge computing. *IEEE Transactions on Vehicular Technology*, 67(11):11008–11021, 2018.

[77] Shanhong Liu. Edge Computing, 2021. `https://www.statista.com/study/82641/edge-computing/`.

[78] Davide Loconte, Saverio Ieva, Filippo Gramegna, Ivano Bilenchi, Corrado Fasciano, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, Michele Ruta, and Eugenio Di Sciascio. Serverless Microservice Architecture for Cloud-Edge Intelligence in Sensor Networks. *IEEE Sensors Journal*, In press.

[79] Davide Loconte, Saverio Ieva, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. Expanding the cloud-to-edge continuum to the IoT in serverless federated learning. *Future Gener. Comput. Syst.*, 155(C):447–462, July 2024.

[80] Victor Lopez, Ricard Vilalta, Victor Uceda, Arturo Mayoral, Ramon Casellas, Ricardo Martinez, Raul Munoz, and Juan Pedro Fernandez Palacios. Transport API: A solution for SDN in carriers networks. In *ECOC 2016; 42nd European Conference on Optical Communication*, pages 1–3. VDE, 2016.

[81] Giuseppe Loseto, Floriano Scioscia, Michele Ruta, Filippo Gramegna, Saverio Ieva, Corrado Fasciano, Ivano Bilenchi, and Davide Loconte.

Osmotic Cloud-Edge Intelligence for IoT-based Cyber-Physical Systems. *Sensors*, 22(6):2166, 2022.

[82] Giuseppe Loseto, Floriano Scioscia, Michele Ruta, Filippo Gramegna, Saverio Ieva, Corrado Fasciano, Ivano Bilenchi, Davide Loconte, and Eugenio Di Sciascio. A Cloud-Edge Artificial Intelligence Framework for Sensor Networks. In *9th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI 2023)*, pages 149–154, 2023.

[83] Theo Lynn, John G. Mooney, Brian Lee, and Patricia Takako Endo. *The Cloud-to-Thing Continuum*. Palgrave Macmillan, Cham, Switzerland, 2020.

[84] Mohammad Malekzadeh, Richard G. Clegg, Andrea Cavallaro, and Hamed Haddadi. Mobile Sensor Data Anonymization. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, IoTDI '19, pages 49–58, New York, NY, USA, 2019. ACM.

[85] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B Letaief. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–58, 2017.

[86] Juri Mattila, Timo Seppälä, and Jan Holmström. Product-centric Information Management: A Case Study of a Shared Platform with Blockchain Technology. In *Berkeley Roundtable on the International Economy*, 2016.

[87] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai, Mike J. Hibbett, Giovani Estrada, Khaldun Ma'ruf, Florin Coras, Vina Ermagan, Hugo Latapie, Chris Cassar, John Evans, Fabio Maino, Jean Walrand, and Albert Cabellos. Knowledge-Defined Networking. *SIGCOMM Comput. Commun. Rev.*, 47(3):2–10, sep 2017.

[88] Matthias Mettler. Blockchain technology in healthcare: The revolution starts here. In *2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom)*, pages 1–3. IEEE, 2016.

[89] Gianfranco E Modoni and Marco Sacco. Discovering Critical Factors Affecting RDF Stores Success. In Rajic Pandey, Marcin Paprzycki, Nidhi Srivastava, Subhash Bhalla, and Katarzyna Wasielewska-Michniewska, editors, *Semantic IoT: Theory and Applications: Interoperability, Provenance and Beyond*, pages 193–206. Springer, 2021.

[90] Martín O. Moguillansky, Renata Wassermann, and Marcelo A. Falappa. An argumentation machinery to reason over inconsistent ontologies. In Guillermo R Simari Angel Kuri-Morales, editor, *Advances in Artificial Intelligence–IBERAMIA 2010*, pages 100–109, Berlin, Germany, 2010. Springer.

[91] Mehdi Montakhabi, Shenja van der Graaf, Akash Madhusudan, Roozbeh Sarenche, and Mustafa A. Mustafa. Fostering Energy Transition in Smart Cities: DLTs for Peer-to-Peer Electricity Trading. In *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 466–472, 2021.

[92] Boris Motik, Bijan Parsia, and Peter Patel-Schneider. OWL 2 Web Ontology Language XML Serialization (Second Edition), December 2012. `https://www.w3.org/TR/owl2-xml-serialization`.

[93] Dae-Hyeok Mun, Minh Le Dinh, and Young-Woo Kwon. An assessment of Internet of Things protocols for resource-constrained applications. In *Computer Software and Applications Conference*, pages 555–560. IEEE, 2016.

[94] Mark Alan Musen. The Protégé project: a look back and a look forward. *AI matters*, 1(4):4–12, 2015.

[95] Jer Shyuan Ng, Wei Yang Bryan Lim, Zehui Xiong, Xianbin Cao, Jiangming Jin, Dusit Niyato, Cyril Leung, and Chunyan Miao. Reputation-

Aware Hedonic Coalition Formation for Efficient Serverless Hierarchical Federated Learning. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):2675–2686, 2022.

[96] Alex Norta. Creation of smart-contracting collaborations for decentralized autonomous organizations. In *International Conference on Business Informatics Research*, pages 3–17. Springer, 2015.

[97] Martin J O'Connor, Christian Halaschek-Wiener, and Mark A Musen. Mapping master: a flexible approach for mapping spreadsheets to OWL. In *International Semantic Web Conference*, pages 194–208. Springer, 2010.

[98] Kelly Olson, Mic Bowman, James Mitchell, Shawn Amundson, Dan Middleton, and Cian Montgomery. Sawtooth: An Introduction. Whitepaper, 01 2018.

[99] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[100] Alfonso Panarello, Nachiket Tapas, Giovanni Merlino, Francesco Longo, and Antonio Puliafito. Blockchain and IoT Integration: A Systematic Survey. *Sensors*, 18(8):2575, 2018.

[101] Massimo Paolucci, Takahiro Kawamura, Terry R Payne, and Katia Sycara. Semantic Matching of Web Services Capabilities. In *International Semantic Web Conference*, pages 333–347. Springer, 2002.

[102] Bijan Parsia, Boris Motik, and Peter Patel-Schneider. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition), December 2012. `http://www.w3.org/TR/owl2-syntax/`.

[103] Bijan Parsia, Sebastian Rudolph, Markus Krötzsch, Peter Patel-Schneider, and Pascal Hitzler. OWL 2 Web Ontology Language

Primer (Second Edition), December 2012. http://www.w3.org/TR/owl2-primer.

[104] P. Patros, M. Ooi, V. Huang, M. Mayo, C. Anderson, S. Burroughs, M. Baughman, O. Almurshed, O. Rana, R. Chard, K. Chard, and I. Foster. Rural AI: Serverless-Powered Federated Learning for Remote Applications. *IEEE Internet Computing*, 27(02):28–34, mar 2023.

[105] Christoph Pinkel, Carsten Binnig, Ernesto Jiménez-Ruiz, Evgeny Kharlamov, Wolfgang May, Andriy Nikolov, Ana Sasa Bastinos, Martin G Skjæveland, Alessandro Solimando, Mohsen Taheriyan, et al. RODI: benchmarking relational-to-ontology mapping generation quality. *Semantic Web*, 9(1):25–52, 2018.

[106] Agnese Pinto, Saverio Ieva, Arnaldo Tomasino, Giuseppe Loseto, Floriano Scioscia, Michele Ruta, and Francesco De Feudis. A Framework for Automatic Knowledge Base Generation from Observation Data Sets. In Sven Casteleyn, Tommi Mikkonen, Alberto García Simón, In-Young Ko, and Giuseppe Loseto, editors, *Current Trends in Web Engineering*, pages 89–100, Cham, 2024. Springer Nature Switzerland.

[107] Behrouz Pourghebleh, Vahideh Hayyolalam, and Amir Aghaei Anvigh. Service discovery in the Internet of Things: review of current trends and research challenges. *Wireless Networks*, 26(7):5371–5391, 2020.

[108] Nihar Pradhan, Akhilendra Singh, Sahil Verma, Kavita ., Marcin Wozniak, Jana Shafi, and Muhammad Fazal Ijaz. A blockchain based lightweight peer-to-peer energy trading framework for secured high throughput micro-transactions. *Scientific Reports*, (2022) 12:14523, 08 2022.

[109] Eric Prud'hommeaux and Gavin Carothers. RDF 1.1 Turtle, February 2014. https://www.w3.org/TR/turtle/.

[110] Victor Casamayor Pujol, Praveen Kumar Donta, Andrea Morichetta, Ilir Murturi, and Schahram Dustdar. Edge Intelligence-Research Opportunities for Distributed Computing Continuum Systems. *IEEE Internet Computing*, 27(4):53–74, 2023.

[111] Ana Reyna, Cristian Martín, Jaime Chen, Enrique Soler, and Manuel Díaz. On blockchain and its integration with IoT. Challenges and opportunities. *Future Generation Computer Systems*, 88:179–190, 2018.

[112] Daniel Rosendo, Alexandru Costan, Patrick Valduriez, and Gabriel Antoniu. Distributed intelligence on the Edge-to-Cloud Continuum: A systematic literature review. *Journal of Parallel and Distributed Computing*, 166:71–94, 2022.

[113] Michele Ruta, Eugenio Di Sciascio, and Floriano Scioscia. Concept abduction and contraction in semantic-based P2P environments. *Web Intelligence and Agent Systems*, 9(3):179–207, 2011.

[114] Michele Ruta, T. D. Noia, Eugenio Di Sciascio, Floriano Scioscia, and Eufemia Tinelli. A Ubiquitous Knowledge-based System to Enable RFID Object Discovery in Smart Environments. *Pac. Asia J. Assoc. Inf. Syst.*, 2:4, 2010.

[115] Michele Ruta, Floriano Scioscia, Ivano Bilenchi, Filippo Gramegna, Giuseppe Loseto, Saverio Ieva, and Agnese Pinto. A multiplatform reasoning engine for the Semantic Web of Everything. *Journal of Web Semantics*, 73:100709, 2022.

[116] Michele Ruta, Floriano Scioscia, Filippo Gramegna, Ivano Bilenchi, and Eugenio Di Sciascio. Mini-ME Swift: the first OWL reasoner for iOS. In *16th Extended Semantic Web Conference (ESWC 2019)*, number 11503 in Lecture Notes in Computer Science, pages 298–313. Springer, 2019.

[117] Michele Ruta, Floriano Scioscia, Saverio Ieva, Giovanna Capurso, and Eugenio Di Sciascio. Semantic blockchain to improve scalability in the Internet of Things. *Open Journal of Internet Of Things (OJIOT)*, 3(1):46–61, 2017.

[118] Michele Ruta, Floriano Scioscia, Saverio Ieva, Giuseppe Loseto, Agnese Pinto, and Arnaldo Tomasino. Blockchain and knowledge representation for service-oriented smart mobility platforms. *Blockchain: Research and Applications*, 2025. in press.

[119] Michele Ruta, Floriano Scioscia, Giuseppe Loseto, Agnese Pinto, and Eugenio Di Sciascio. Machine learning in the Internet of Things: A semantic-enhanced approach. *Semantic Web*, 10(1):183–204, 2019.

[120] Eveline R Sacramento, Vânia MP Vidal, José Antonio F de Macêdo, Bernadette F Lóscio, Fernanda Lígia R Lopes, and Marco A Casanova. Towards automatic generation of application ontologies. *Journal of Information and Data Management*, 1(3):535–535, 2010.

[121] Guus Schreiber and Fabien Gandon. RDF 1.1 XML syntax, February 2014. `http://www.w3.org/TR/rdf-syntax-grammar/`.

[122] Guus Schreiber and Fabien Gandon. RDF-star and SPARQL-star, June 2023. `https://w3c.github.io/rdf-star/cg-spec/editors_draft.html`.

[123] Floriano Scioscia, Ivano Bilenchi, Michele Ruta, Filippo Gramegna, and Davide Loconte. A multiplatform energy-aware OWL reasoner benchmarking framework. *Journal of Web Semantics*, 72:100694, 2022.

[124] Floriano Scioscia, Giuseppe Loseto, Davide Loconte, Corrado Fasciano, Filippo Gramegna, Saverio Ieva, Agnese Pinto, and Michele Ruta. Osmotic Computing Platform for Smart City Applications in the Cloud-Edge Continuum. In *9th Italian Conference on ICT for Smart Cities And Communities (I-CiTies 2023)*, sep 2023.

[125] Floriano Scioscia, Giuseppe Loseto, Arnaldo Tomasino, Ivano Bilenchi, Filippo Gramegna, Saverio Ieva, Agnese Pinto, Eugenio Di Sciascio, and Michele Ruta. Embedded reasoning for UAV operations: towards real-time efficiency and trustworthy autonomy. In *9th Italian Conference on ICT for Smart Cities And Communities (I-CiTies 2023)*, sep 2023.

[126] Floriano Scioscia and Michele Ruta. Building a Semantic Web of Things: issues and perspectives in information compression. In *Proceedings of the 3rd IEEE International Conference on Semantic Computing*, pages 589–594. IEEE Computer Society, 2009.

[127] Floriano Scioscia, Michele Ruta, Giuseppe Loseto, Filippo Gramegna, Saverio Ieva, Agnese Pinto, and Eugenio Di Sciascio. Mini-ME matchmaker and reasoner for the Semantic Web of Things. In *Innovations, Developments, and Applications of Semantic Web and Information Systems*, pages 262–294. IGI Global, Hershey, PA, USA, 2018.

[128] Hossein Shafiei, Ahmad Khonsari, and Payam Mousavi. Serverless Computing: A Survey of Opportunities, Challenges, and Applications. *ACM Computing Surveys*, 54(11s), nov 2022.

[129] Maulshree Singh, Evert Fuenmayor, Eoin P Hinchy, Yuansong Qiao, Niall Murray, and Declan Devine. Digital Twin: Origin to future. *Applied System Innovation*, 4(2):36, 2021.

[130] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007.

[131] Steffen Staab and Rudi Studer. *Handbook on ontologies*. Springer Science & Business Media, 2010.

[132] Lanfang Sun, Xin Jiang, Huixia Ren, and Yi Guo. Edge-Cloud Computing and Artificial Intelligence in Internet of Medical Things: Architecture, Technology and Application. *IEEE Access*, 8:101079–101092, 2020.

[133] Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.

[134] John R Talburt, Lisa Ehrlinger, and Justin Magruder. Automated data curation and data governance automation. *Frontiers in Big Data*, 6, 2023.

[135] Gunnar Teege. Making the Difference: A Subtraction Operation for Description Logics. In *Proceedings of the Fourth International Conference on the Principles of Knowledge Representation and Reasoning (KR'94)*, pages 540–550, New York, NY, USA, 1994. ACM.

[136] Nikolay Teslya and Alexander Smirnov. Blockchain-based framework for ontology-oriented robots' coalition formation in cyberphysical systems. In *MATEC Web of Conferences*, volume 161, page 03018. EDP Sciences, 2018.

[137] Robert Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, 1953.

[138] Daniel R Torres, Cristian Martín, Bartolomé Rubio, and Manuel Díaz. An open source framework based on Kafka-ML for Distributed DNN inference over the Cloud-to-Things continuum. *Journal of Systems Architecture*, 118:102214, 2021.

[139] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *International Joint Conference on Automated Reasoning (IJCAR)*, pages 292–297, Berlin, Germany, 2006. Springer, Springer.

[140] Francesco Tusa and Stuart Clayman. End-to-end slices to orchestrate resources and services in the cloud-to-edge continuum. *Future Generation Computer Systems*, 141:473–488, 2023.

[141] Theo Van Veen. Wikidata: fron "an" identifier to "the" identifier. *Information technology and libraries*, 38(2):72–81, 2019.

[142] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan. Osmotic Computing: A New Paradigm for Edge/Cloud Integration. *IEEE Cloud Computing*, 3(6):76–83, 2016.

[143] Binh Vu, Craig Knoblock, and Jay Pujara. Learning semantic models of data sources using probabilistic graphical models. In *The World Wide Web Conference*, pages 1944–1953, 2019.

[144] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *International Workshop on Open Problems in Network Security*, pages 112–125. Springer, 2015.

[145] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview (Second Edition), December 2012. https://www.w3.org/TR/owl2-overview/.

[146] K. Wang, H. Yin, W. Quan, and G. Min. Enabling Collaborative Edge Computing for Software Defined Vehicular Networks. *IEEE Network*, 32(5):112–117, 2018.

[147] Qin Wang, Xinqi Zhu, Yiyang Ni, Li Gu, and Hongbo Zhu. Blockchain for the IoT and industrial IoT: A review. *Internet of Things*, 10:100081, 2020.

[148] Zehui Xiong, Yang Zhang, Dusit Niyato, Ping Wang, and Zhu Han. When mobile blockchain meets edge computing. *IEEE Communications Magazine*, 56(8):33–39, 2018.

[149] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[150] Jiangchao Yao, Shengyu Zhang, Yang Yao, Feng Wang, Jianxin Ma, Jianwei Zhang, Yunfei Chu, Luo Ji, Kunyang Jia, Tao Shen, et al. Edge-cloud polarization and collaboration: A comprehensive survey for AI. *IEEE Transactions on Knowledge and Data Engineering*, 35(7):6866–6886, 2022.

[151] Tsozen Yeh and Shengchieh Yu. Realizing dynamic resource orchestration on cloud systems in the cloud-to-edge continuum. *Journal of Parallel and Distributed Computing*, 160:100–109, 2022.

[152] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.

[153] Mirko Zichichi, Stefano Ferretti, and Gabriele D'Angelo. On the Efficiency of Decentralized File Storage for Personal Information Management Systems. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2020.

[154] Mirko Zichichi, Stefano Ferretti, and Víctor Rodríguez-Doncel. Decentralized Personal Data Marketplaces: How Participation in a DAO Can Support the Production of Citizen-Generated Data. *Sensors*, 22(16), 2022.

List of publications

Journal articles

1. Michele Ruta, Floriano Scioscia, Saverio Ieva, Giuseppe Loseto, Agnese Pinto, and Arnaldo Tomasino. Blockchain and knowledge representation for service-oriented smart mobility platforms. *Blockchain: Research and Applications*, 2025. in press

2. Ivano Bilenchi, Filippo Gramegna, Giuseppe Loseto, Saverio Ieva, Floriano Scioscia, and Michele Ruta. Cowl: Pushing OWL 2 over the Edge. *Elsevier Internet of Things Journal*, 29(101439):1–20, Jan 2025

3. Davide Loconte, Saverio Ieva, Filippo Gramegna, Ivano Bilenchi, Corrado Fasciano, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, Michele Ruta, and Eugenio Di Sciascio. Serverless Microservice Architecture for Cloud-Edge Intelligence in Sensor Networks. *IEEE Sensors Journal*, In press

4. Saverio Ieva, Davide Loconte, Giuseppe Loseto, Michele Ruta, Floriano Scioscia, Davide Marche, and Marianna Notarnicola. A Retrieval-Augmented Generation Approach for Data-Driven Energy Infrastructure Digital Twins. *Smart Cities*, 7(6):3095–3120, 2024

5. Davide Loconte, Saverio Ieva, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. Expanding the cloud-to-edge continuum to the IoT in serverless federated learning. *Future Gener. Comput. Syst.*, 155(C):447–462, July 2024

6. Michele Ruta, Floriano Scioscia, Ivano Bilenchi, Filippo Gramegna, Giuseppe Loseto, Saverio Ieva, and Agnese Pinto. A multiplatform reasoning

engine for the Semantic Web of Everything. *Journal of Web Semantics*, 73:100709, 2022

7. Giuseppe Loseto, Floriano Scioscia, Michele Ruta, Filippo Gramegna, Saverio Ieva, Corrado Fasciano, Ivano Bilenchi, and Davide Loconte. Osmotic Cloud-Edge Intelligence for IoT-based Cyber-Physical Systems. *Sensors*, 22(6):2166, 2022

Peer-reviewed conference papers

1. Saverio Ieva, Ivano Bilenchi, Filippo Gramegna, Floriano Scioscia, Michele Ruta, and Giuseppe Loseto. Integrating AI-based fleet optimization with mixed reality to enhance last-mile delivery. In *10th Italian Conference on ICT for Smart Cities and Communities (I-CiTies 2024)*, 2024

2. Agnese Pinto, Saverio Ieva, Arnaldo Tomasino, Giuseppe Loseto, Floriano Scioscia, Michele Ruta, and Francesco De Feudis. A Framework for Automatic Knowledge Base Generation from Observation Data Sets. In Sven Casteleyn, Tommi Mikkonen, Alberto García Simón, In-Young Ko, and Giuseppe Loseto, editors, *Current Trends in Web Engineering*, pages 89–100, Cham, 2024. Springer Nature Switzerland

3. Giuseppe Loseto, Floriano Scioscia, Michele Ruta, Filippo Gramegna, Saverio Ieva, Corrado Fasciano, Ivano Bilenchi, Davide Loconte, and Eugenio Di Sciascio. A Cloud-Edge Artificial Intelligence Framework for Sensor Networks. In *9th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI 2023)*, pages 149–154, 2023

4. Floriano Scioscia, Giuseppe Loseto, Arnaldo Tomasino, Ivano Bilenchi, Filippo Gramegna, Saverio Ieva, Agnese Pinto, Eugenio Di Sciascio, and Michele Ruta. Embedded reasoning for UAV operations: towards real-time efficiency and trustworthy autonomy. In *9th Italian Conference on ICT for Smart Cities And Communities (I-CiTies 2023)*, sep 2023

5. Floriano Scioscia, Giuseppe Loseto, Davide Loconte, Corrado Fasciano, Filippo Gramegna, Saverio Ieva, Agnese Pinto, and Michele Ruta. Osmotic Computing Platform for Smart City Applications in the Cloud-Edge Continuum. In *9th Italian Conference on ICT for Smart Cities And Communities (I-CiTies 2023)*, sep 2023

6. Floriano Scioscia, Giuseppe Loseto, Davide Loconte, Corrado Fasciano, Filippo Gramegna, Saverio Ieva, Agnese Pinto, and Michele Ruta. Osmotic Computing Platform for Smart City Applications in the Cloud-Edge Continuum. In *9th Italian Conference on ICT for Smart Cities And Communities (I-CiTies 2023)*, sep 2023

7. Ivano Bilenchi, Arnaldo Tomasino, Filippo Gramegna, Saverio Ieva, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. Knowledge Representation and Reasoning for Unmanned Aerial Vehicle Intelligence. In *7th Italian Workshop on Embedded Systems (IWES 2022)*, 2022

8. Filippo Gramegna, Arnaldo Tomasino, Saverio Ieva, Ivano Bilenchi, Agnese Pinto, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. RideMATCHain: a Semantic-enhanced Blockchain Marketplace for Ridesharing. In *8th Italian Conference on ICT for Smart Cities And Communities (I-CiTies 2022)*, 2022

9. Saverio Ieva, Davide Loconte, Agnese Pinto, Filippo Gramegna, Giuseppe Loseto, Floriano Scioscia, and Michele Ruta. Semantic-enhanced Network Orchestration to Support Smart City Services. In *8th Italian Conference on ICT for Smart Cities And Communities (I-CiTies 2022)*, 2022

Glossary

ABox Assertion Box. 65, 66, 70, 72, 132

AI Artificial Intelligence. 8, 10, 28, 29, 66, 73, 74, 83, 84

API Application Programming Interface. 39, 40, 55, 103, 105, 106

AST Abstract Syntax Tree. 31

AWS Amazon Web Services. 75, 89, 91– 93, 99– 102, 105, 106, 110– 112, 116, 118

BFT Byzantine Fault Tolerance. 15, 128

BLE Bluetooth Low Energy. 104

BOM Build OWL Model. 78

BTM Build Training Matrix. 77

CA Concept Abduction. 49, 51, 52, 54, 61

CB Concept Bonus. 49, 52

CC Concept Contraction. 47, 51, 52

CCov Concept Covering. 52, 53

CD Concept Contraction. 52

CE concept expression. 44

CMSIS Common Micro-controller Software Interface Standard. 105

CNF Conjunctive Normal Form. 25, 26, 44, 45, 48, 49, 51, 52, 54

COTS Commercial Of-The-Shelf. 75

CPS Cyber-Physical System. , 1– 5, 7– 12, 19, 29, 41– 43, 54, 65, 73, 74, 126, 130, 147, 148

CPU Central Processing Unit. 102, 116

DAG Directed Acyclic Graph. 18, 19

DAO Decentralized Autonomous Organization. 14, 16

DC Data Collection. 77

DL Description Logic. 20– 26, 42– 45, 60, 67, 77

DLT Distributed Ledger Technology. , 1, 2, 4, 12, 13, 18, 19, 29, 41– 43, 54, 66, 68, 74, 75, 147, 148

DM Data Modeling. 77

DNN Deep Neural Network. 83, 84

DSS Decision Support System. 30, 35– 39, 43

DT Digital Twin. 8– 11

EI Edge Intelligence. 7, 10, 11

EM Elbow Method. 69, 71

EPCIS Electronic Product Code Information Services. 16

ETSI European Telecommunications Standards Institute. 31

EU European Union. 57

FaaS Function-as-a-Service. 84

FD Field Device. 93, 95, 99, 101, 104, 105, 108, 110, 112, 114, 119

FL Federated Learning. 3, 74, 83, 84

FOL First Order Logic. 20

GNSS global navigation satellite system. 59

HTTP HyperText Transfer Protocol. 43, 105

IANA Internet Assigned Numbers Authority. 31

IETF Internet Engineering Task Force. 31

IIoT Industrial Internet of Things. 142

IoT Internet of Things. , 1, 3, 6– 8, 10, 15– 19, 28, 64, 73, 74, 81, 83– 88, 91– 94, 99– 101, 103– 106, 108– 110, 112, 118, 119, 123– 129, 131, 132, 134, 136, 138, 141, 148

IPC Inter-Process Communication. 92

IPFS InterPlanetary File System. 18

IRI Internationalized Resource Identifier. 21

JRE Java Runtime Environment. 111

JSON JavaScript Object Notation. 32– 34, 70, 105

KB Knowledge Base. 3, 6, 7, 21, 30, 32, 36, 39, 42, 53, 57– 66, 70– 72

KDN Knowledge-Defined Networking. 2, 29

KG Knowledge Graph. 3, 9, 11, 63, 64

KP Knowledge Plane. 29

KPI Key Performance Indicator. 35

KR Knowledge Representation. 5, 124

LEL Lower Explosion Limit. 58

LHS left-hand side. 26, 45

LLM Large Language Model. 9, 11

LOC Limiting Oxygen Concentration. 58

LOD Linked Open Data. 64

LTL Linear Temporal Logic. 17

MAFALDA MAtchmaking Features for mAchine Learning Data Analysis. 76, 77, 79– 81, 86, 88, 91, 93, 102, 105, 108, 111, 112, 116, 118, 119

MCU Micro Controller Unit. 104, 105, 107, 111

MEC Mobile Edge Computing. 17, 18, 123– 125, 127, 142

ML Machine Learning. 8, 10, 29, 63, 66, 73, 74, 76, 83, 88, 95, 119

MQTT Message Queuing Telemetry Transport. 88, 89, 91– 93, 101, 103, 105, 106

NA Network Analytics. 29

NETCONF Network Configuration Protocol. 30

NFC Near Field Communication. 104

NFV Network Function Virtualization. 2, 28

NFV-SOL Network Functions Virtualisation Solutions. 31

NSL Network Service Level. 40

OC Osmotic Computing. 11, 73, 83

ODbL Open Database License. 106

ONF Open Network Foundation. 31, 32

OWA Open World Assumption. 47, 49

OWL Web Ontology Language. 2, 6, 20– 24, 31– 33, 36, 39, 43, 58, 63, 65, 67, 68, 70, 71, 125, 128

PEV Plug-in Electric Vehicle. 4, 125, 134

PoET Proof of Elapsed Time. 130, 142

PoW Proof-of-Work. 13, 17, 18, 128

QoS Quality of Service. 37

R2RML Relational database-to-RDF. 64

RAG Retrieval-Augmented Generation. 9, 11

RAM Random Access Memory. 116

RAMI Reference Architecture Model Industry. 16

RDF Resource Description Framework. 6, 24, 63, 64

REST REpresentational State Transfer. 39, 129

RHS right-hand side. 26

RML RDF Mapping Language. 64

RTOS Real-Time Operating System. 42, 43, 56, 87, 88

SaaS Software as a Service. 91

SC Smart Contract. 14, 16, 17, 19, 68, 127, 131– 134, 136, 137, 139

SD Secure Digital. 102

SDK Software Development Kit. 138

SDN Software-Defined Networking. 2, 28, 29, 32, 34

SG Semantic Gateway. 138

SHFL Serverless Hierarchical Federated Learning. 84

SLA Service Level Agreement. 132

SOA Service-Oriented Architecture. 16, 19, 124, 125, 131, 142

SoC system-on-chip. 56

STF Semantic Transaction Family. 138

STP Semantic Transaction Processor. 138

SWoE Semantic Web of Everything. 42, 43, 63, 64, 67, 69, 74

SWoT Semantic Web of Things. , 1, 4, 6, 7, 19, 43, 44, 55, 62, 124, 147, 148

SWRL Semantic Web Rule Language. 30, 36, 37, 39

TAPI Transport API. 31, 32, 34– 36, 38– 40

TBox Terminological Box. 54, 65– 68, 70– 72

TM Training Matrix. 77, 79, 81, 88, 93, 94, 112

TP Transaction Processor. 129, 130, 142

TRL Technology Readiness Level. 148

u-KB ubiquitous Knowledge Base. 132

UAV Unmanned Aerial Vehicle. 42, 56, 57, 59– 61, 66

URI Universal Resource Identifier. 21, 131, 133, 139

UTXO Unspent Transaction Output. 15

VM Virtual Machine. 14

W3C World Wide Web Consortium. 5, 20, 21, 64

WI Web Interface. 129, 130

WWW World Wide Web. 5, 63

XML Extensible Markup Language. 31

YANG Yet Another Next Generation. 30– 34