PeerJ Computer Science

# OntOraculum: a semantic ontology for managing performance metrics in smart environments

Darlan Noetzold[1,2], Valderi R. Q. Leithardt[3], Juan Francisco De Paz[1] and Jorge L. V. Barbosa[2]

[1] Expert Systems and Applications Laboratory, Faculty of Science, Universidad de Salamanca, Salamanca, Spain
[2] Universidade do Vale do Rio dos Sinos (UNISINOS), São Leopoldo, RS, Brazil
[3] University Institute of Lisbon (ISCTE-IUL), ISTAR, Lisbon, Portugal

## ABSTRACT

This study presents OntOraculum, an ontology designed to organize and analyze performance metrics in smart environments. It categorizes metrics into Hardware, Network, Software, Energy, and SecurityAndReliability domains, defining properties such as dependencies, impacts, and trends to support predictive analysis and dynamic adjustments. Using Protégé for modeling, Semantic Web Rule Language (SWRL) rules facilitate automated inferences, while SPARQL Protocol and Resource Description Framework (RDF) Query Language (SPARQL) queries enable data retrieval from the ontology. Through validation queries, the ontology addresses questions related to network latency, energy efficiency, security, quality of service, and anomaly detection. By unifying performance metrics across diverse domains, OntOraculum enables comprehensive analysis of dependencies and interactions among metrics. The results demonstrate its practical applicability for monitoring, optimizing, and adapting systems in distributed environments.

**Subjects** Algorithms and Analysis of Algorithms, Artificial Intelligence, Computer Architecture, Software Engineering, Internet of Things
**Keywords** IoT data simulation, Heuristic optimization, Predictive analytics, Machine learning, Ontology

Submitted 23 December 2025
Accepted 27 July 2026
Published 28 October 2026

Corresponding author
Darlan Noetzold,
darlannoetzold@usal.es

Academic editor
Sara Comai

Additional Information and Declarations can be found on page 37

DOI **10.7717/peerj-cs.4078**

Copyright
2026 Noetzold et al.

Distributed under
Creative Commons CC-BY 4.0

OPEN ACCESS

## INTRODUCTION

Monitoring and analyzing performance metrics provide insights into system efficiency, reliability, and adaptability in smart environments, such as smart cities, connected industries, and automated homes. These environments rely on diverse approaches to model, monitor, and analyze metrics related to hardware, networks, software, energy, security, and service availability (*Amin et al., 2015*; *Noetzold et al., 2026*). Capturing and processing such metrics supports the identification of performance bottlenecks, characterization of system behavior, and guidance for adaptive adjustments aimed at improving functionality and resource utilization.

Despite the importance of these metrics, categorizing and structuring them remains challenging due to their heterogeneity and interdependencies. Existing approaches often lack unified and flexible frameworks capable of supporting semantic reasoning and cross-domain analysis. To address this gap, several studies have applied ontological

techniques to structure domain knowledge and facilitate automated interpretation in intelligent systems (*Bavaresco et al., 2024*; *Helfer & Victória Barbosa, 2025*). Ontologies define concepts and relationships within a domain, enabling automated reasoning over complex datasets and supporting decision-making processes. In the context of the Internet of Things (IoT), ontologies have been applied in domains such as cybersecurity assessment, operational risk mitigation, semantic interoperability in monitoring frameworks, and healthcare systems for remote patient monitoring (*Sharma & Singh, 2020*).

OntOraculum is the ontology proposed in this work to address these limitations by providing a unified semantic model for performance monitoring in smart environments. It formalizes the representation of performance metrics and their interrelationships, enabling rule-based reasoning and structured analysis across heterogeneous domains. The ontology organizes domain knowledge into well-defined classes, properties, and rules, facilitating interoperability and semantic integration of data from multiple sources.

OntOraculum organizes performance metrics into five core categories: *Hardware*, *Network*, *Software*, *Energy*, and *SecurityAndReliability*. By defining semantic properties such as dependencies, impacts, thresholds, and status conditions, the ontology establishes formal relationships between metrics, supporting reasoning and analytical evaluation. For example, metrics such as *CPUUsage* and *NetworkLatency* can indicate potential performance bottlenecks, which may be used to inform load-balancing strategies in distributed systems (*Faraji-Mehmandar, Jabbehdari & Javadi, 2022*). Similarly, energy-related metrics such as *EnergyEfficiency* are relevant for evaluating system behavior in domains such as industrial automation and smart agriculture (*Chakrabarti, Sadhu & Pal, 2023*).

The ontology also incorporates metrics related to security and system reliability, including *AnomalyDetection* and *ServiceUptime*, which support the identification of abnormal behaviors and service degradation (*Blomqvist et al., 2024*; *Lateef Haroon et al., 2024*). These dimensions are particularly relevant in systems operating in critical environments, where resilience and adaptability are required. By representing these metrics semantically, OntOraculum provides a foundation for rule-based inference and integration with analytical processes in smart environments.

This article presents a structured and semantic-driven approach to performance monitoring in smart environments based on the OntOraculum ontology. The proposed approach demonstrates how semantic technologies can be used to model metric interdependencies, reduce ambiguity, and support analytical reasoning. The ontology is evaluated through SPARQL Protocol and Resource Description Framework (RDF) Query Language (SPARQL) queries, Semantic Web Rule Language (SWRL) rules, and reasoning tasks using a description logic reasoner. The results obtained in a simulated environment indicate the feasibility and internal consistency of the approach for tasks such as anomaly identification and performance assessment.

Additionally, the article discusses how the ontology can be integrated with monitoring systems to support structured analysis and alert generation. While the current evaluation is

based on simulated data, the approach provides a basis for future validation in real-world scenarios and for integration with live data streams in intelligent infrastructures.

The article proceeds as follows. "Related Work" reviews related work, highlighting existing ontologies and identifying gaps in performance monitoring approaches. "Methodology" describes the ontology development process, including scope definition, competency questions, and ontology structure. "Evaluation" presents the verification and validation of the ontology through reasoning, SPARQL queries and rule-based inference, as well as its integration with monitoring systems. Finally, "Discussion" discusses the results, and "Conclusion" concludes the work and outlines future research directions.

## RELATED WORK

This study explored major academic databases to identify research applying ontologies to performance evaluation and monitoring. The search strategy prioritized publications employing ontological models for computational performance monitoring, optimization, and analysis. The review integrates both foundational and contemporary research to provide a comprehensive overview of the field.

Consulted databases include IEEE (https://ieeexplore.ieee.org/), ACM (https://dl.acm.org/), Springer (https://link.springer.com/), Scopus (https://www.scopus.com/), and ScienceDirect (https://www.sciencedirect.com/). Table 1 presents the search string. The selection process follows three primary inclusion criteria: (i) peer-reviewed publications in English; (ii) explicit use of ontologies or semantic models in performance monitoring or evaluation; and (iii) availability of methodological or experimental details on ontology construction or application. Exclusion criteria comprise: (i) studies without implementation or evaluation; (ii) works unrelated to performance metrics; (iii) duplicate records across databases; and (iv) articles addressing purely conceptual discussions without applying semantic reasoning mechanisms.

The three-pass reading method proposed by *Keshav (2007)* was used to refine the selection. The first pass examined titles and abstracts, the second focused on introduction and methodology sections, and the third analyzed the full content of the articles. After this process, eight studies were selected for comparative analysis. Each selected study applies ontology-based approaches to performance measurement with varying objectives and technical depth (*Allahim, Cherif & Imine, 2025*). Table 2 presents a comparative analysis of the selected studies, focusing on six aspects: the domain addressed, the use of rules, the ability to support inferences, the use of query mechanisms, and the presence of evaluation procedures. Each column is described as follows:

- **Reference**: It identifies the author(s) and the publication year of the study.
- **Scope**: It describes the domain or application area addressed by the proposed ontology.
- **Rules**: It indicates whether the study defines explicit rules for evaluation or monitoring tasks.
- **Inferences**: It denotes whether the ontology includes or supports reasoning and inference mechanisms.

**Table 1** Definition of the search string for related work on ontology.

| Key terms | Search terms |
|---|---|
| Metrics | ("metric*" OR "benchmark") |
| Performance | ("performance" OR "computational efficiency" OR "computational processing") |
| Ontology | ("ontolog*") |

**Note:**
*a truncation wildcard that matches any word ending (e.g., metric, metrics).

**Table 2** Comparison of related works on ontology-based performance monitoring, evaluated across seven dimensions: scope, inference support, query mechanisms, evaluation method, inference time, accuracy, and error handling; N/A indicates metrics not reported in the original study.

| Ref. | Scope | Inf. | Query | Evaluation | Inf. time | Accuracy | Error handling |
|---|---|---|---|---|---|---|---|
| *Corry et al. (2015)* | Building performance | Yes | Yes | Semantic ontology for buildings | 1 s | 90% (approx.) | No |
| *Truong et al. (2005)* | Workflow in grid | Yes | No | Performance monitoring | N/A | N/A | No |
| *Baocai et al. (2010)* | Web service discovery | No | No | QoS for service selection | ~500 ms | 85% precision | No |
| *Cao et al. (2019)* | Service rec. in IoT | Yes | Yes | Factorization + QoS ontology | N/A | N/A | No |
| *Yaseen et al. (2011)* | Oracle tech eval. | Yes | Yes | Domain rules validation | 700 ms | Qualitative results | No |
| *Metwally, Jarray & Karmouch (2015)* | IaaS resource alloc. | Yes | No | Semantic model for cloud | N/A | N/A | No |
| *Rzevski, Skobelev & Zhilyaev (2022)* | Smart ecosystems | Yes | Yes | Agents for resource conflicts | ~2 s | Based on agent outcome | No |
| *Daoudagh & Marchetti (2023)* | IoT monitoring | Yes | Yes | Alert ontology + testbed | ~400 ms | ~92% | Partial |
| *Sharma et al. (2021)* | IoT health monitor | Yes | Yes | Ontology + health rules | N/A | 95% recall | No |
| *Zeshan et al. (2023)* | Healthcare + IoT | Yes | Yes | Evaluation in smart health | 890 ms | 96% precision | No |
| *Belani, Solic & Perkovic (2022)* | IoT + aging care | Yes | No | Requirements modeling | N/A | N/A | No |
| This work | Perf. metrics in smart env. | Yes | Yes | SPARQL + SWRL + CQ validation | 367 ms | 94%/100% | Yes |

- **Queries**: It specifies the use of semantic query languages, such as SPARQL, within the study.
- **Evaluation**: It explains whether the ontology was validated through experimentation, case studies, or formal verification procedures.

*Corry et al. (2015)* proposed an ontology for the evaluation of environmental and energy performance in buildings. The model transformed heterogeneous data into semantically enriched representations to support analysis throughout the building lifecycle. The evaluation included performance comparisons with a traditional system, reporting approximate inference accuracy and processing time around 1 s (*Corry et al., 2015*). *Truong et al. (2005)* introduced an ontology for workflow monitoring in grid environments. The approach captured performance metrics at multiple abstraction levels and supported execution analysis. Although rule-based inference was implemented, the study did not report quantitative performance metrics or use semantic queries for automated reasoning (*Truong et al., 2005*).

*Baocai et al. (2010)* presented a Quality of Service (QoS) ontology aimed at web service discovery. The model used semantic annotations to describe non-functional attributes and improve service selection and interoperability. The evaluation included inference precision estimates but lacked broader system-level analysis or integration with reasoning engines (*Baocai et al., 2010*). *Cao et al. (2019)* developed a service recommendation model for IoT environments based on a QoS ontology, integrating factorization models and relational topics. While the approach improved recommendation accuracy, it did not include semantic reasoning capabilities or evaluate inference performance metrics explicitly (*Cao et al., 2019*).

*Yaseen et al. (2011)* evaluated Oracle semantic technologies using domain-specific rules to validate application-specific knowledge bases. The evaluation contrasted performance with and without customized rule bases but did not report formal accuracy or precision measures beyond qualitative interpretation (*Yaseen et al., 2011*). *Metwally, Jarray & Karmouch (2015)* proposed an ontology for resource allocation in Infrastructure-as-a-Service (IaaS) platforms. The model supported semantic representation of cloud resources and allocation strategies based on similarity and centrality measures. However, the work did not focus on reasoning scalability or generalized performance inference (*Metwally, Jarray & Karmouch, 2015*).

*Rzevski, Skobelev & Zhilyaev (2022)* presented an ontology-based system for emergent intelligence in digital ecosystems. The approach incorporated autonomous agents to resolve resource conflicts through semantic consensus. Although the reasoning framework supported adaptability, the study did not evaluate inference precision or error-handling robustness (*Rzevski, Skobelev & Zhilyaev, 2022*). *Daoudagh & Marchetti (2023)* proposed DAEMON, a domain-based monitoring ontology for IoT systems. The model defined a layered semantic architecture that enabled domain-specific monitoring and alert generation. It incorporated rules and queries for detecting anomalies and performance issues, and was validated through a testbed setup and IoT use cases (*Daoudagh & Marchetti, 2023*).

*Sharma et al. (2021)* introduced an ontology-based framework for remote patient monitoring. The model semantically represented health-related metrics and integrated them into an IoT architecture that enabled reasoning and alert generation in healthcare applications. The study reported recall metrics to evaluate the effectiveness of ontology-based alerts (*Sharma et al., 2021*). *Zeshan et al. (2023)* presented a semantic framework for intelligent healthcare monitoring, combining IoT sensing with ontology-based reasoning. The model handled semantic integration, alert inference, and remote communication of health parameters; precision metrics were reported based on system evaluation in a smart healthcare environment (*Zeshan et al., 2023*).

*Belani, Solic & Perkovic (2022)* explored an ontology-based approach to requirements engineering in IoT-supported domains for aging, well-being, and health. The model supported conceptual modeling of domain-specific requirements, with qualitative evaluation through use case scenarios (*Belani, Solic & Perkovic, 2022*).

The analysis of these works reveals three primary trends in the application of ontologies to performance monitoring: (i) domain-specific frameworks, which focus on isolated

environments like buildings or grid computing; (ii) task-oriented models, mainly designed for automated service discovery and resource recommendation; and (iii) decentralized IoT architectures, primarily addressing interoperability and connectivity challenges in healthcare and smart cities. While these studies demonstrate the effectiveness of semantic modeling for organizing domain knowledge, they generally focus on vertical silos, often overlooking the horizontal dependencies that exist between hardware, network, and energy consumption in a unified smart environment.

OntOraculum builds upon these foundations by proposing a coordinated framework that bridges these silos. Unlike the aforementioned approaches, the model does not treat metrics as independent variables; instead, it explicitly represents the influence of hardware performance on QoS and energy efficiency. Furthermore, while most related works rely on standard logical inference, OntOraculum integrates advanced error-modeling techniques—specifically interval-based reasoning and Riccati estimation—to enhance the robustness of alert generation under uncertain or noisy conditions. This synthesis of multi-layer dependency modeling and mathematical refinement distinguishes the study from prior literature and addresses the need for more resilient performance monitoring in dynamic intelligent systems.

In comparison to the selected studies, OntOraculum introduces a unified semantic model that integrates performance metrics across multiple layers of smart environments, including hardware, software, network, energy, and security domains. Previous works such as *Corry et al. (2015)*, *Truong et al. (2005)*, and *Baocai et al. (2010)* focused on specific domains like building performance, Grid workflows, and web service discovery, without addressing the interoperability of metrics across heterogeneous systems.

Several studies apply ontologies to enhance recommendation systems or infrastructure resource allocation in cloud and IoT contexts (*Cao et al., 2019*; *Metwally, Jarray & Karmouch, 2015*). These models rely on domain-specific rules or optimization criteria but do not generalize reasoning across subsystems or support predictive analysis. Works such as *Yaseen et al. (2011)* and *Rzevski, Skobelev & Zhilyaev (2022)* included reasoning mechanisms but limit their evaluation to particular semantic platforms or agent-based scenarios.

Recent contributions like DAEMON (*Daoudagh & Marchetti, 2023*), Smart Health Ontologies (*Sharma et al., 2021*; *Zeshan et al., 2023*), and semantic integration frameworks (*Belani, Solic & Perkovic, 2022*; *Al-Sayed, Hassan & Omara, 2020*) extended ontology application to IoT interoperability and healthcare. These works demonstrate high accuracy in reasoning and alert generation, often within constrained domains such as patient monitoring. However, they do not address cross-domain dependencies between hardware, network, energy, and software indicators.

OntOraculum contributes by offering a coordinated framework that supports SPARQL querying, SWRL-based inference, metric dependency modeling, and predictive reasoning. The model has been validated with real-time metric simulations, yielding high accuracy and low inference latency. It also includes advanced error modeling techniques, such as

interval-based reasoning and Riccati estimation, not found in the other reviewed studies. These features enable robust performance monitoring and adaptation strategies in distributed and dynamic environments, contributing to both operational effectiveness and research advancement.

## METHODOLOGY

According to *Gruber (1995)*, ontologies explicitly define formal specifications of the terms within a domain and their relationships, providing machine-readable definitions of fundamental concepts. When structuring a domain through ontologies, the main objectives include communicating knowledge, establishing a representative vocabulary, and understanding the semantics of the data in that domain (*Khan et al., 2025*). Portions of this text were previously published as part of a preprint.

Full reproducibility of the experimental workflow depends on the following configuration details. Protégé version 5.5.0 (*Musen, 2015a*) served as the development and validation platform, while the Pellet reasoner handled logical inference and instance classification. The reasoning process ran on a mid-range workstation (Intel i5-12400, 16 GB RAM; Intel Corp., Santa Clara, CA, USA), completing all inference tasks in 367 ms. SWRL rules were authored directly within Protégé using the SWRLTab plugin, and SPARQL queries were executed *via* the SPARQL Query tab integrated into the same environment. Instances were generated programmatically using a publicly available simulator (*Noetzold, 2024*), which produces synthetic metric values following configurable distributions to approximate realistic operational scenarios of intelligent systems. The simulator was configured to generate 159 individuals covering five metric domains: hardware, network, software, energy, and security. Each individual is assigned a numerical value *via* the `hasValue` data property, which is subsequently evaluated by the SWRL rules to trigger domain-specific alert classifications. Threshold values used in the rules (*e.g.*, central processing unit (CPU) usage above 80%, network latency above 150 ms, power consumption above 200 W) were defined based on commonly adopted operational baselines in the smart systems literature and are explicitly listed in the rule tables presented in 'Define Semantic Rules'.

Ontologies structure domain knowledge hierarchically, with all classes derived from a root entity commonly referred to as *thing*. In the Web Ontology Language (OWL), they consist of four main components: concepts (*classes*), instances (*individuals*), properties, and constraints (*Munir & Sheraz Anjum, 2018*). These elements collectively define a semantic data model that enables the integration and reasoning of domain-specific knowledge. To build such a model, this work adopts the structured methodology proposed by *Noy & McGuinness (2001)*, which consists of seven sequential stages:

1. Define the domain, scope and competency questions (CQ): The ontology focuses on performance monitoring in smart environments involving hardware, network, software, energy and security metrics. It covers organizing these metrics to help with monitoring, optimizing, and predicting as well.

2. Search for existing ontologies: Reviewing related ontologies ensures alignment with prior research and identifies gaps that this work addresses.
3. Define and formalize key terms: The literature review provides key terms such as *CPUUsage*, *NetworkLatency*, and *EnergyEfficiency*, along with their relationships and impacts.
4. Organize and structure the ontology in Protégé: Classes and relationships are defined in Protégé, with a hierarchical arrangement to represent metric interdependencies.
5. Establish object and data properties: The ontology defines relationships such as *dependsOn* and *impacts* as object properties, while data properties describe attributes like *average*, *max_value*, and *timestamp*.
6. Define restrictions and SWRL rules: Semantic rules and constraints are created using the SWRL to identify bottlenecks, predict failures, and recommend adjustments.
7. Create individuals for validation: Instances are generated to test and validate the ontology, representing scenarios such as high CPU usage, network latency, and energy efficiency. These instances support validation through reasoning tasks and SPARQL queries.

This structured approach develops the ontology to facilitate performance monitoring and optimization in smart environments. The design of OntOraculum rests on the assumption that performance monitoring in smart environments requires a multi-domain approach to capture the interdependencies between hardware, software, and network layers. The selection of OWL provides formal semantic expressive power, while SWRL enables the definition of conditional logic beyond simple class hierarchies. These choices ensure model interoperability with other Semantic Web standards and provide a flexible framework for real-time anomaly detection and resource optimization.

## Domain definition

This ontology focuses on performance monitoring in smart environments, including smart cities, connected buildings, and sensor networks. It analyzes metrics related to system efficiency, network resilience, energy consumption, latency, security, application availability, and anomaly detection. The objective is to support the analysis and adjustment of distributed and heterogeneous systems.

## Scope definition

The ontology covers metrics at multiple levels of abstraction, allowing the evaluation of both high-level indicators and detailed measurements. Performance metrics fall into the following categories:

- **Hardware**: Metrics for CPU, graphic processing unit (GPU), memory, storage, and sensor usage.
- **Network**: Metrics for latency, bandwidth and resilience.
- **Software**: Metrics for response time, application availability and anomaly detection.

- **Energy**: Metrics for energy consumption and efficiency.
- **Security and Reliability**: Metrics for intrusion detection, error rates, and system vulnerabilities.

## Competency questions

CQs formulate the scope of the ontology, as well as verify its applicability and evaluate how well it applies domain aspects. These questions help identify performance monitoring points in smart environments, including issues such as resource efficiency and identification of critical problems. CQs also justify the purpose of the ontology and evaluate its relevance in practical contexts (*Noy & McGuinness, 2001*; *Feng et al., 2025*).

In the ontology domain, these metrics are related to network latency, resource utilization, energy efficiency, security, and system stability. CQs emulate real-world problems in smart cities, sensor networks, connected industries, and automated living spaces. These questions summarize the issues the ontology seeks to solve to classify and structure information that aids rational action.

These CQs define the fundamental questions that the ontology needs to address:

- **CQ1**: What is the current impact of network latency on the QoS of critical applications?
- **CQ2**: Does the system exhibit high resilience based on uptime rates and network stability?
- **CQ3**: Which CPU, memory, and throughput utilization metrics contribute to system instability?
- **CQ4**: What is the system's average energy efficiency based on transaction consumption and main devices?
- **CQ5**: What is the rate of detected intrusion attempts, and how does it affect overall system stability?
- **CQ6**: What is the forecast for CPU usage in the coming periods, and how does it impact expected performance?
- **CQ7**: Are there patterns in critical metrics, such as cache miss rate or throughput, indicating potential future failures?
- **CQ8**: Which metrics are most related to drops in quality of service (QoS) or system failures?
- **CQ9**: Which changes in network metrics directly impact system recovery and resilience?
- **CQ10**: Is there any ongoing service degradation caused by congestion or high resource utilization?

Each CQ focuses on a specific domain aspect and helps infer insights from monitored metrics. For instance, CQ1 and CQ9 address real-time dependencies in network latency, while CQ4 focuses on energy efficiency in smart environments.

CQs validate the ontology by determining whether the defined classes, properties, and relationships can effectively answer these questions. Additionally, these questions guide

modeling decisions, ensuring the structured representation of monitored metrics for analysis and decision-making.

The use of CQs aligns with established ontology development methodologies, which emphasize defining questions that clarify an ontology's capabilities and limitations. Addressing the CQs ensures the ontology supports the monitoring and analysis of smart environments, improving system reliability and adaptability.

## Key terms in the ontology

A systematic mapping structured the concepts related to performance monitoring. This process identified and categorized metrics into the main observability domains. Concept-mapping techniques guided the organization of these terms, illustrating the relationships among concepts within a domain (*Metwally, Jarray & Karmouch, 2015*). Figure 1 presents a concept map that describes the relationships among the metrics monitored in smart environments. This map was created using the Whimsical (https://whimsical.com) tool and contains 72 classes and their relations. The map organizes metrics into five primary categories, Hardware, Software, Network, Energy, and Security, each subdivided into specific metrics for performance monitoring and analysis.

Figure 1 shows a hierarchical organization that structures performance metrics into increasingly specific levels. For example, the *Hardware* category includes subcategories such as *CPU Usage*, *Memory*, and *Sensors*, while the *Network* category shows metrics such as *Latency*, *Congestion*, and *Bandwidth*. This data structuring allows for insights into how metrics influence system performance (*Munir & Sheraz Anjum, 2018*). The concept map also represents the relationships between metrics, graphically presented by arrows and nodes. These relationships address direct and indirect dependencies, such as the influence of *CPU Usage* on energy consumption and network latency, for example. This type of mapping illuminates connections that other structured approaches may overlook. For example, the category *Energy* interacts with *Hardware* and *Network*, illustrating how variations in energy consumption influence architecture metrics (*Xu et al., 2025*).

An iterative process of literature review, concept mapping, and validation with values generated by the simulator (*Noetzold, 2024*) refined the definition of terms and their relationships. This approach ensures that the ontology captures the metrics needed to monitor smart environments, forming a structured basis for subsequent analysis and inferences (*Noy & McGuinness, 2001*).

## Defining classes and hierarchy

Figure 2 shows the process of defining classes and hierarchy followed by an iterative approach, using the previously listed terms and the structure. Each class appears in singular form, aligning with best practices for ontologies, where each concept describes one or more instances.

In OWL notation, the *owl:Thing* class serves as the root of the hierarchy and provides the foundation for all other classes (*Djurić, Gašević & Devedžić, 2005*). The class hierarchy relies on inheritance, where subclasses inherit information from their parent classes while adding specific characteristics. This structure organizes concepts related to performance

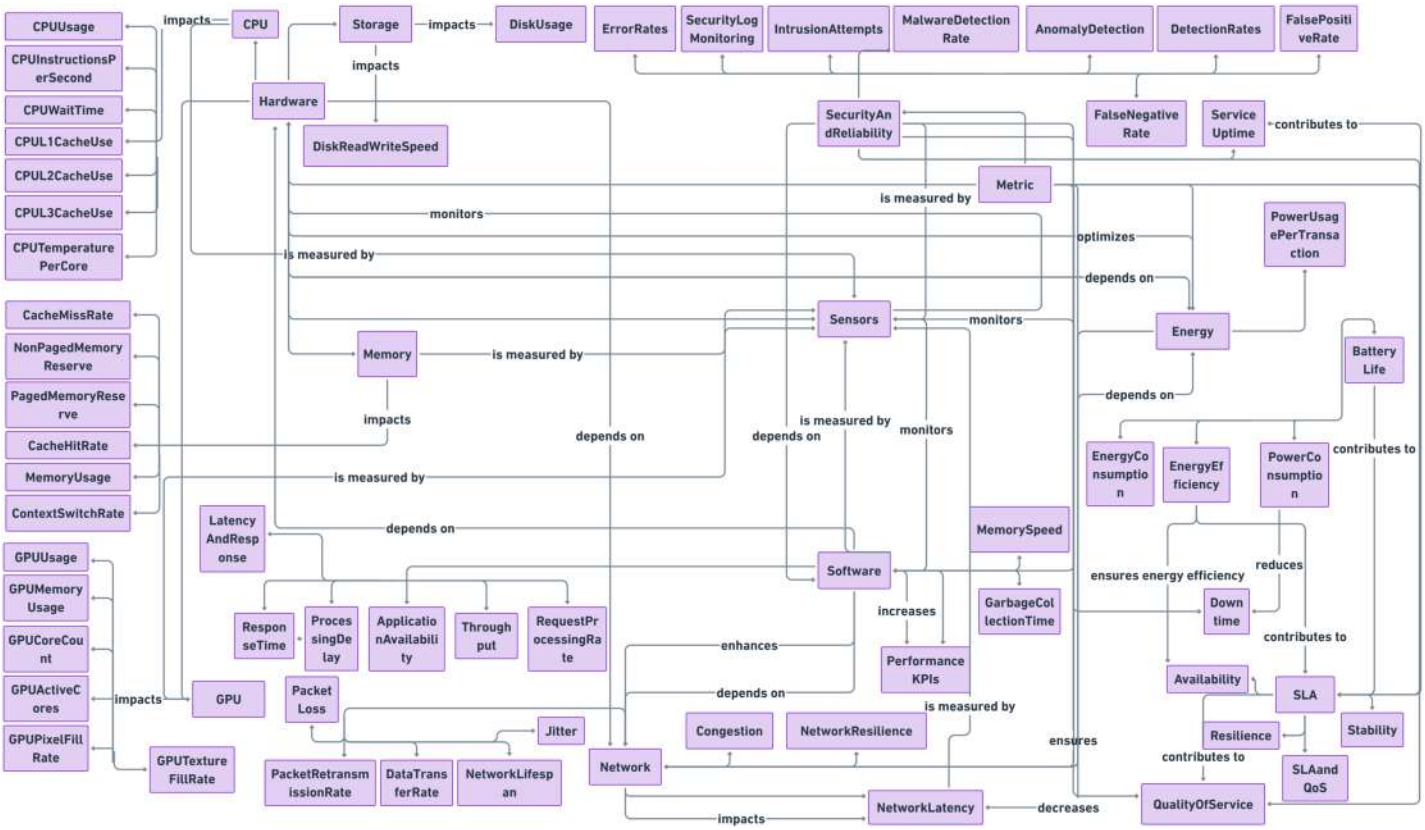



**Figure 1** **Conceptual map of the ontology with 72 classes and their relationships, organized into five primary categories (hardware, software, network, energy, and security), built using the Whimsical tool based on the key terms identified during the literature review.**
Full-size DOI: 10.7717/peerj-cs.4078/fig-1

metrics in smart environments in a detailed manner. The ontology was modeled using Protégé software (version 5.5.0) to define classes and properties (*Musen, 2015b*). The following sections outline the primary classes and their respective descriptions:

- **Metric**: It serves as the core concept of the ontology, representing all monitored performance indicators. It encompasses six primary subclasses—*Hardware*, *Network*, *Software*, *Energy*, *service-level agreement* (SLA), and *SecurityAndReliability*—each addressing a distinct aspect of system performance.
- **Hardware**: It represents the physical infrastructure metrics of smart environments, including components such as *CPU*, *GPU*, *Memory*, *Storage*, and *Sensors*. It also includes detailed subclasses that capture specific indicators, such as *CPUUsage*, *GPUMemoryUsage*, and *CacheHitRate*.
- **Network**: It defines communication-related metrics used to evaluate connectivity and resilience, including *NetworkLatency*, *DataTransferRate*, and *PacketLoss*.
- **Software**: It encompasses metrics related to application-level performance and process efficiency, such as *LatencyAndResponseTime*, *Throughput*, and *GarbageCollectionTime*.

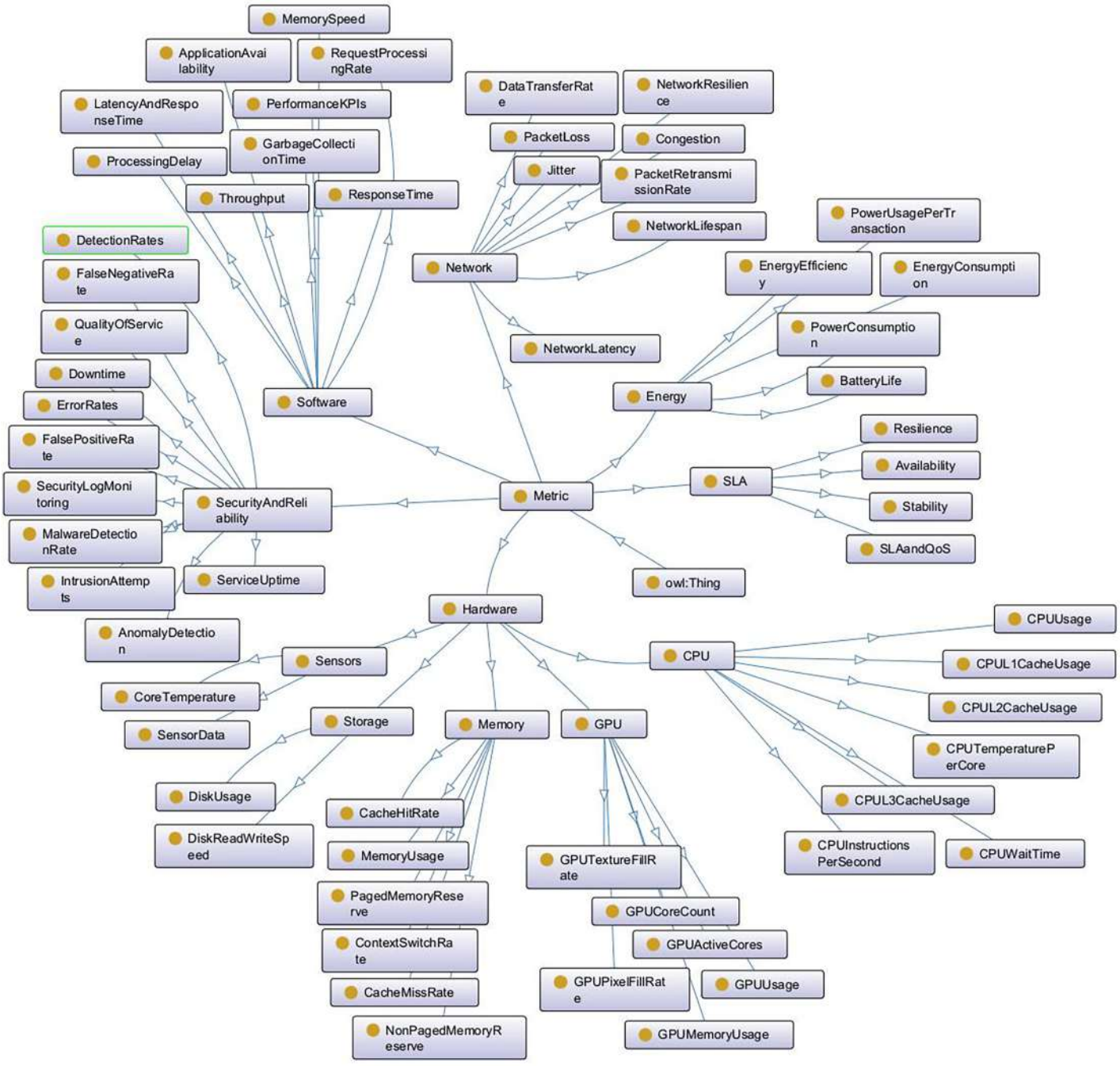



**Figure 2** **Class hierarchy of the ontology modeled in Protégé 5.5.0, showing the six primary subclasses derived from the root *metric* class: hardware, network, software, energy, SLA, and SecurityAndReliability.** Full-size DOI: 10.7717/peerj-cs.4078/fig-2

- **Energy**: It describes energy-related performance aspects, addressing both consumption and efficiency through metrics such as *EnergyEfficiency*, *PowerConsumption*, and *BatteryLife*.
- **SLA**: It covers metrics defined within service-level agreements that measure quality and continuity, including *Availability*, *Resilience*, and *Stability*.

- **SecurityAndReliability**: It incorporates indicators related to security assurance and operational reliability, including *IntrusionAttempts*, *MalwareDetectionRate*, and *ServiceUptime*.

## Defining relationships and class properties

Figures 3 and 4 show how the ontology establishes relationships and class properties to structure interactions and attributes of each concept. Figure 3 displays the class hierarchy, structured from the root class *owl:Thing*, with object properties in blue (relationships) and data properties in green (attributes). These elements were defined using OWL in Protégé (*Musen, 2015a*).

Figure 4 outlines object properties such as *dependsOn*, *impacts*, and *enhances*, which describe dependencies, influences, and optimizations among concepts. Data properties define attributes like *average*, *max_value*, and *timestamp*, supporting quantitative and qualitative analyses.

The inheritance model allows subclasses to retain parent class characteristics while incorporating specific properties. Relationships such as *dependsOn* connect metrics like CPU usage and energy consumption, while *impacts* and *enhances* describe influence and efficiency improvements. These relationships support advanced queries and inferences, assisting in the evaluation of metrics in smart environments.

## Define semantic rules

The literature discusses the use of semantic rules as an approach to enable additional inferences from data structured in ontologies, enhancing analysis and modeling capabilities in systems (*Horrocks et al., 2004*; *O'Connor et al., 2005*). This work defines rules using the SWRL, which extends OWL's capabilities by incorporating first-order logic and enabling conditional implications in the format "if A, then B". Tables 3, 4, 5, and 6 categorize the defined rules into resource optimization, quality of service, energy efficiency, and security with anomaly detection.

The threshold values defined in the SWRL rules (Tables 3, 4, 5, and 6) were established based on standard operational performance benchmarks and literature-supported stress conditions. For instance, the 80% threshold for CPU usage and the 150 ms limit for network latency are widely recognized as transition points between normal and degraded performance in distributed systems (*Horrocks et al., 2004*). These parameters ensure that the inferences generated by OntOraculum are grounded in realistic operational constraints. All rules are executed deterministically within Protégé 5.5.0 using the Pellet reasoner, ensuring that identical input values always yield consistent results.

Each SWRL rule follows the standard structure of an implication of the form antecedent → consequent, where the antecedent (body) defines a set of conditions and the consequent (head) defines the inferred classification. Variables are represented with the prefix "?" (*e.g.*, ?cpu, ?val1) and are universally quantified within the rule. Predicates such as *CPUUsage(?cpu)* indicate class membership, while data properties such as *hasValue(?cpu, ?val1)* associate individuals with numerical values.

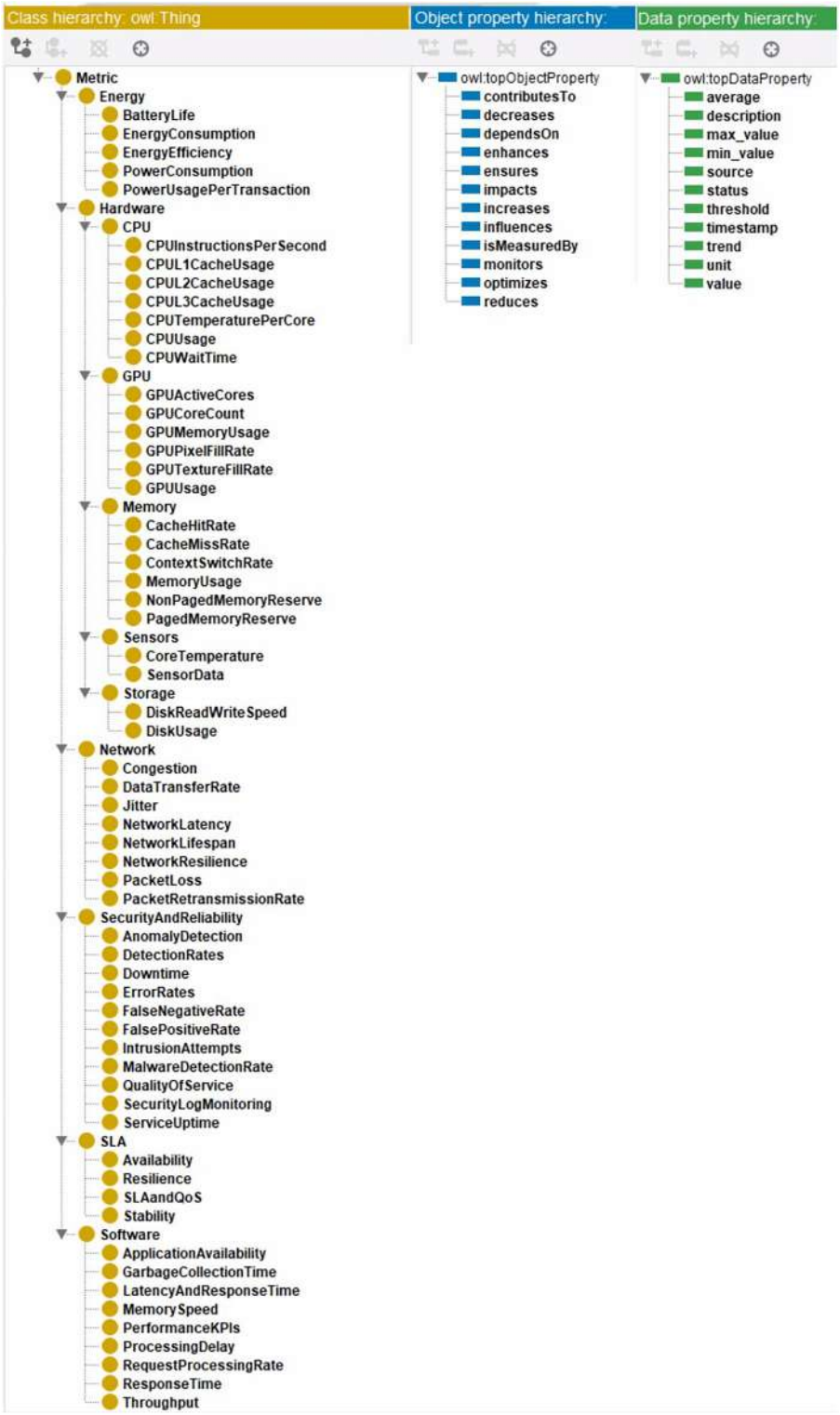



**Figure 3** **Hierarchy of ontology classes and properties in Protégé 5.5.0, where object properties (blue) represent semantic relationships such as *dependsOn* and *impacts*, and data properties (green) represent quantitative attributes such as *hasValue* and *timestamp*.**

Full-size DOI: 10.7717/peerj-cs.4078/fig-3

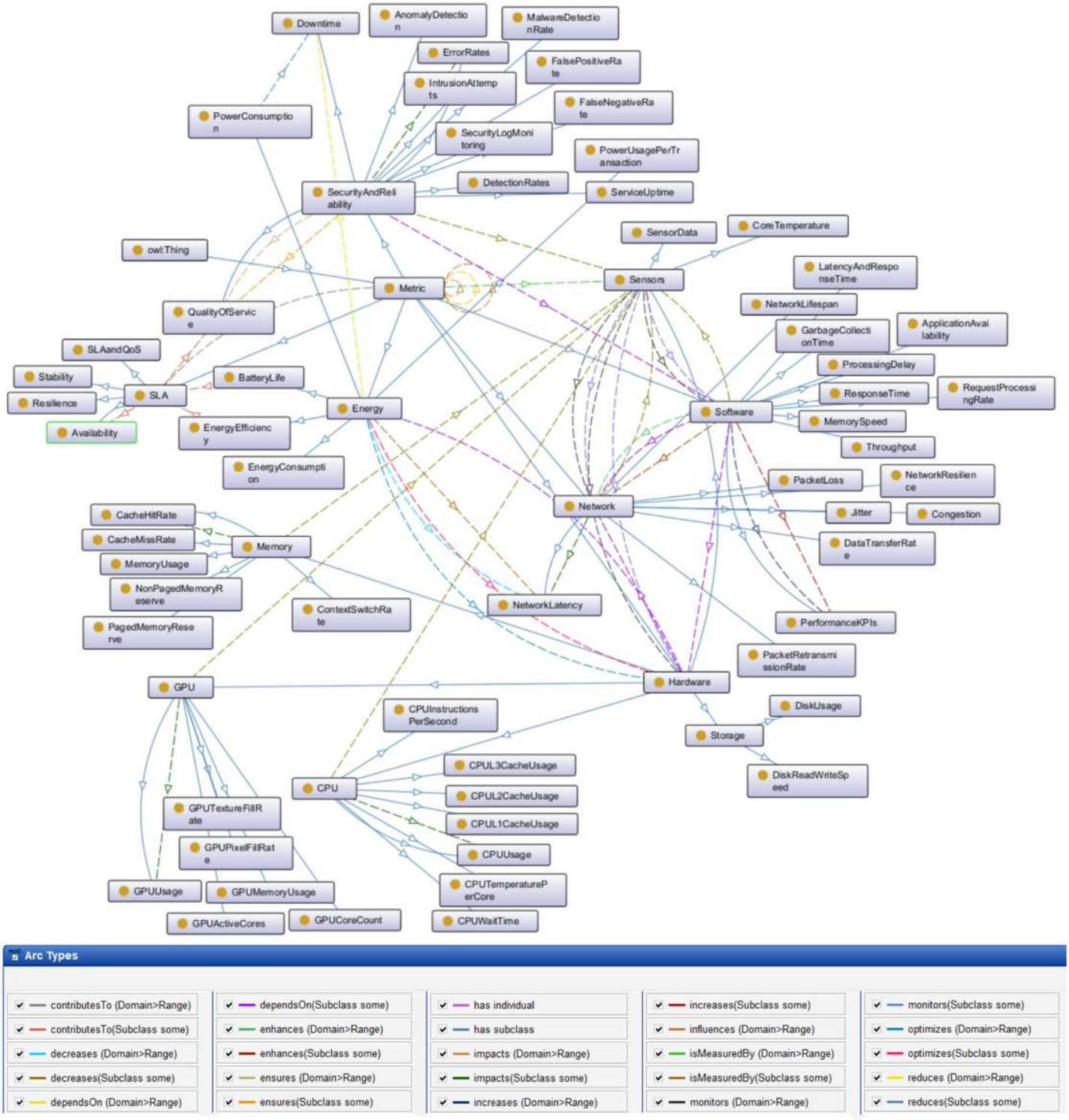



**Figure 4** **Relationships between classes and domain-range properties, illustrating how object properties such as *dependsOn*, *impacts*, and *enhances* connect metric classes across the hardware, network, software, energy, and security domains.**

Full-size DOI: 10.7717/peerj-cs.4078/fig-4

**Table 3** SWRL rules for resource optimization.

| Rule name | SWRL expression |
|---|---|
| Resource_HighCPUUsage | CPUUsage(?cpu) ^ hasValue(?cpu, ?val1) ^ swrlb:greaterThan(?val1, 80) → HighResourceUsage(?cpu) |
| Resource_LowMemory | MemoryUsage(?mem) ^ hasValue(?mem, ?val1) ^ swrlb:lessThan(?val1, 30) → LowMemoryScenario(?mem) |
| Resource_OverloadedGPU | GPUUsage(?gpu) ^ hasValue(?gpu, ?val1) ^ swrlb:greaterThan(?val1, 85) → OverloadedGPU(?gpu) |
| Resource_CriticalDiskIO | DiskReadWriteSpeed(?drws) ^ hasValue(?drws, ?val1) ^ swrlb:lessThan(?val1, 10) → CriticalDiskPerformance (?drws) |
| Resource_StableUsage | CPUUsage(?cpu) ^ hasValue(?cpu, ?val1) ^ swrlb:lessThan(?val1, 50) ^ MemoryUsage(?mem) ^ hasValue(? mem, ?val2) ^ swrlb:lessThan(?val2, 60) → StableResourceUsage(?cpu) |
| Resource_UnbalancedNetwork | NetworkLatency(?lat) ^ hasValue(?lat, ?val1) ^ swrlb:greaterThan(?val1, 150) ^ PacketLoss(?pl) ^ hasValue(?pl, ?val2) ^ swrlb:greaterThan(?val2, 10) → UnbalancedNetwork(?lat) |
| Resource_HighEnergyConsumption | PowerConsumption(?pc) ^ hasValue(?pc, ?val1) ^ swrlb:greaterThan(?val1, 200) → HighEnergyUsage(?pc) |
| Resource_InefficientEnergyUsage | EnergyEfficiency(?ee) ^ hasValue(?ee, ?val1) ^ swrlb:lessThan(?val1, 70) → InefficientEnergy(?ee) |
| Resource_CongestedNetwork | Congestion(?cong) ^ hasValue(?cong, ?val1) ^ swrlb:greaterThan(?val1, 50) → NetworkCongestion(?cong) |
| Resource_CriticalPerformance | CPUUsage(?cpu) ^ hasValue(?cpu, ?val1) ^ swrlb:greaterThan(?val1, 90) ^ MemoryUsage(?mem) ^ hasValue(? mem, ?val2) ^ swrlb:greaterThan(?val2, 80) → CriticalPerformance(?cpu) |
| Resource_LowDiskReadSpeed | DiskReadWriteSpeed(?drws) ^ hasValue(?drws, ?val1) ^ swrlb:lessThan(?val1, 15) → CriticalDiskPerformance (?drws) |
| Resource_UnderutilizedMemory | MemoryUsage(?mem) ^ hasValue(?mem, ?val1) ^ swrlb:lessThan(?val1, 20) → LowMemoryScenario(?mem) |

**Table 4** SWRL rules for quality of service.

| Rule name | SWRL expression |
|---|---|
| QoS_HighCPUImpact | CPUUsage(?cpu) ^ hasValue(?cpu, ?val1) ^ swrlb:greaterThan(?val1, 80) → HighImpactQoS(?cpu) |
| QoS_NetworkCongestion | NetworkLatency(?lat) ^ hasValue(?lat, ?val1) ^ swrlb:greaterThan(?val1, 150) ^ Congestion(?cong) ^ hasValue(? cong, ?val2) → DegradedQoS(?lat) |
| QoS_Optimal | CPUUsage(?cpu) ^ hasValue(?cpu, ?val1) ^ swrlb:lessThan(?val1, 50) → OptimalQoS(?cpu) |
| QoS_HighMemoryImpact | MemoryUsage(?mem) ^ hasValue(?mem, ?val1) ^ swrlb:greaterThan(?val1, 90) → HighImpactQoS(?mem) |
| QoS_DiskPerformanceDegraded | DiskReadWriteSpeed(?drws) ^ hasValue(?drws, ?val1) ^ swrlb:lessThan(?val1, 20) → DegradedQoS(?drws) |
| QoS_UnstableThroughput | Throughput(?tp) ^ hasValue(?tp, ?val1) ^ swrlb:lessThan(?val1, 100) → UnstableQoS(?tp) |
| QoS_ReliableApplication | ApplicationAvailability(?aa) ^ hasValue(?aa, ?val1) ^ swrlb:greaterThan(?val1, 95) → ReliableQoS(?aa) |
| QoS_ResponseTimeCritical | ResponseTime(?rt) ^ hasValue(?rt, ?val1) ^ swrlb:greaterThan(?val1, 200) → CriticalQoS(?rt) |
| QoS_ApplicationDowntime | Downtime(?dt) ^ hasValue(?dt, ?val1) ^ swrlb:greaterThan(?val1, 60) → ApplicationDowntimeImpact(?dt) |
| QoS_ServiceDegradation | ServiceUptime(?uptime) ^ hasValue(?uptime, ?val1) ^ swrlb:lessThan(?val1, 95) → ServiceDegradation(?uptime) |
| QoS_CriticalNetworkPerformance | NetworkLatency(?lat) ^ hasValue(?lat, ?val1) ^ swrlb:greaterThan(?val1, 200) → CriticalQoS(?lat) |

Built-in functions from the SWRL built-in library (*e.g.*, `swrlb:greaterThan`, `swrlb:lessThan`) are used to perform numerical comparisons. For example, the rule *CPUUsage(?cpu) ^ hasValue(?cpu, ?val1) ^ swrlb:greaterThan(?val1, 80) → HighResourceUsage(?cpu)* infers that a CPU instance belongs to a high-usage category when its value exceeds the defined threshold.

The resource optimization category, shown in Table 3, includes rules that identify scenarios of high component utilization, such as CPU, memory, and GPU usage, as well as bottlenecks in disk input/output (I/O) and network latency. For instance, the *Resource_HighCPUUsage* rule identifies situations where CPU usage exceeds 80%, while

**Table 5** SWRL rules for energy optimization.

| Rule name | SWRL expression |
|---|---|
| Energy_CriticalConsumption | PowerConsumption(?pc) ^ hasValue(?pc, ?val1) ^ swrlb:greaterThan(?val1, 200) → CriticalEnergyConsumption (?pc) |
| Energy_EfficientScenario | EnergyEfficiency(?ee) ^ hasValue(?ee, ?val1) ^ swrlb:greaterThan(?val1, 90) → OptimalEnergyUsage(?ee) |
| Energy_LowBatteryImpact | BatteryLife(?bl) ^ hasValue(?bl, ?val1) ^ swrlb:lessThan(?val1, 20) → CriticalBatteryScenario(?bl) |
| Energy_HighSensorConsumption | SensorData(?sd) ^ hasValue(?sd, ?val1) ^ PowerConsumption(?pc) ^ hasValue(?pc, ?val2) ^ swrlb:greaterThan(?val2, 150) → HighSensorEnergy(?sd) |
| Energy_BalancedScenario | PowerConsumption(?pc) ^ hasValue(?pc, ?val1) ^ swrlb:lessThan(?val1, 100) ^ EnergyEfficiency(?ee) ^ hasValue (?ee, ?val2) ^ swrlb:greaterThan(?val2, 80) → BalancedEnergyUsage(?pc) |
| Energy_OverloadedDevices | PowerConsumption(?pc) ^ hasValue(?pc, ?val1) ^ swrlb:greaterThan(?val1, 120) → OverloadedDevices(?pc) |
| Energy_OptimalPerformance | EnergyEfficiency(?ee) ^ hasValue(?ee, ?val1) ^ swrlb:greaterThan(?val1, 95) ^ ServiceUptime(?uptime) ^ hasValue(?uptime, ?val2) ^ swrlb:greaterThan(?val2, 99) → HighPerformanceEnergy(?ee) |
| Energy_ResourceIntensive | PowerUsagePerTransaction(?put) ^ hasValue(?put, ?val1) ^ swrlb:greaterThan(?val1, 70) → ResourceIntensiveEnergy(?put) |
| Energy_CriticalUptimeImpact | Downtime(?dt) ^ hasValue(?dt, ?val1) ^ swrlb:greaterThan(?val1, 30) → EnergyCriticalUptimeImpact(?dt) |
| Energy_DegradationScenario | EnergyEfficiency(?ee) ^ hasValue(?ee, ?val1) ^ swrlb:lessThan(?val1, 60) → DegradedEnergyUsage(?ee) |
| Energy_PowerSurgeDetected | PowerConsumption(?pc) ^ hasValue(?pc, ?val1) ^ swrlb:greaterThan(?val1, 300) → CriticalEnergyConsumption (?pc) |

**Table 6** SWRL rules for security and anomaly detection.

| Rule name | SWRL expression |
|---|---|
| Security_HighErrorRate | ErrorRates(?er) ^ hasValue(?er, ?val1) ^ swrlb:greaterThan(?val1, 25) → HighErrorRateScenario(?er) |
| Security_PotentialIntrusion | IntrusionAttempts(?ia) ^ hasValue(?ia, ?val1) ^ swrlb:greaterThan(?val1, 10) → PotentialIntrusionScenario(?ia) |
| Security_MalwareDetected | MalwareDetectionRate(?mdr) ^ hasValue(?mdr, ?val1) ^ swrlb:greaterThan(?val1, 5) → MalwareDetected(?mdr) |
| Security_FrequentFalsePositives | FalsePositiveRate(?fp) ^ hasValue(?fp, ?val1) ^ swrlb:greaterThan(?val1, 10) → FrequentFalsePositives(?fp) |
| Security_AccurateDetection | DetectionRates(?dr) ^ hasValue(?dr, ?val1) ^ swrlb:greaterThan(?val1, 95) ^ FalsePositiveRate(?fp) ^ hasValue(?fp, ?val2) ^ swrlb:lessThan(?val2, 5) → AccurateDetectionScenario(?dr) |
| Security_ServiceDowntime | ServiceUptime(?uptime) ^ hasValue(?uptime, ?val1) ^ swrlb:lessThan(?val1, 95) → UnstableServiceScenario(?uptime) |
| Security_AnomalyDetected | AnomalyDetection(?ad) ^ hasValue(?ad, ?val1) → AnomalyDetectedAt(?ad) |
| Security_QoSSecurityImpact | QualityOfService(?qos) ^ hasValue(?qos, ?val1) ^ swrlb:greaterThan(?val1, 90) → HighQoSSecurity(?qos) |
| Security_HighDowntime | Downtime(?dt) ^ hasValue(?dt, ?val1) ^ swrlb:greaterThan(?val1, 60) → HighDowntimeScenario(?dt) |
| Security_CompromisedSecurity | IntrusionAttempts(?ia) ^ hasValue(?ia, ?val1) ^ swrlb:greaterThan(?val1, 10) ^ MalwareDetectionRate(?mdr) ^ hasValue(?mdr, ?val2) ^ swrlb:greaterThan(?val2, 5) → CompromisedSecurity(?ia) |
| Security_AnomalyTimeCheck | AnomalyDetection(?ad) ^ timestamp(?ad, ?t) → AnomalyDetectedAt(?ad, ?t) |

*Resource_StableUsage* infers conditions of balanced CPU and memory usage. These rules anticipate issues and guide dynamic adjustments, which play a critical role in systems operating under high load (*Musen, 2015a*).

Table 4 describes rules related to Quality of Service (QoS) that monitor the impact of variables such as network latency, congestion, and response time in distributed applications. For example, the *QoS_NetworkCongestion* rule identifies QoS degradation

caused by high latency and congestion, while *QoS_Optimal* evaluates scenarios where performance metrics remain within acceptable parameters.

Table 5 presents rules for energy efficiency that identify critical energy consumption patterns and promote optimized scenarios. The *Energy_CriticalConsumption* rule detects high energy consumption conditions, while *Energy_EfficientScenario* evaluates high-efficiency scenarios based on the correlation between consumption and uptime. This approach plays a significant role in sensor networks and autonomous systems operating under energy constraints (*O'Connor et al., 2005*).

Table 6 describes the rules for the detection of security and anomalies, designed to identify vulnerabilities and adverse conditions. The *Security_HighErrorRate* rule identifies scenarios with high error rates, while *Security_AccurateDetection* evaluates the reliability of event detection with low false-positive rates. *Horrocks et al. (2004)* emphasize the importance of these rules in building resilient and reliable systems.

This work implements all rules using Protégé version 5.5.0, which supports SWRL expressions integrated with OWL ontologies (*Musen, 2015a*). The structure of these rules integrates information from various dimensions, enabling inferences that assist in managing intelligent systems, particularly in identifying anomalies and optimizing resources.

## Instance creation

To validate the developed ontology, this work creates instances for the classes and metrics defined in the ontology. These instances address scenarios such as energy consumption, resource utilization, service performance, QoS metrics, and anomaly detection. The instances represent real-world scenarios within the knowledge model and derive from the semantic rules developed. The data for creating these instances comes from a simulator for a sensor data management system. This simulator, available in a public repository on Zenodo (https://doi.org/10.5281/zenodo.20126903), generates simulated metrics that mirror realistic operational scenarios of intelligent systems. The use of the simulator generates varied and consistent data, ensuring the created instances represent the proposed domain and validating them within the ontology (*Noetzold, 2024*).

Reproducibility improves through explicit parameterization of the instance generation process. The simulator generates synthetic data using predefined ranges for each monitored metric, ensuring controlled variability across scenarios. For example, CPU usage values range from 0–100%, memory usage from 0–100%, network latency from 0–200 ms, and power consumption from 0–300 W. These ranges cover both normal and critical operating conditions.

The simulator executes a deterministic generation process based on fixed parameter ranges and scenario definitions, ensuring that equivalent configurations produce consistent datasets. Each generated instance is mapped to a corresponding ontology class and assigned a numerical value through the `hasValue` data property. These values are subsequently evaluated by SWRL rules to infer system states such as anomalies, performance degradation, or resource inefficiencies.

**Table 7 Summary of axiom counts and ontology properties extracted from Protégé 5.5.0 after reasoning with the Pellet engine, reflecting the full ontology with 159 generated individuals across five metric domains.**

| Description | Count |
|---|---|
| Axiom | 2,186 |
| Logical axiom count | 1,894 |
| Declaration axioms count | 292 |
| Class count | 111 |
| Object property count | 12 |
| Data property count | 11 |
| Individual count | 159 |
| Annotation property count | 3 |

In total, 159 individuals were generated, covering five main domains: hardware, software, network, energy, and security. The dataset includes both normal and anomalous conditions, enabling validation of rule-based inference and reasoning consistency. All generated instances follow a structured naming convention (*e.g.*, *metric_instance_ID*), ensuring traceability between simulated data and ontology entities.

Table 7 presents the ontology metrics extracted from Protégé, including the number of axioms, classes, data properties, object properties, and individuals created. These values indicate the ontology's complexity and its capability to represent diverse scenarios. The following examples highlight some of the instances created:

- *high_cpu_instance*: associated with the *CPUUsage* class, with a value of 85%, indicating high CPU usage and triggering an alert for excessive resource consumption.
- *low_memory_instance*: linked to the *MemoryUsage* class, with a value of 25%, indicating low available memory and activating a critical memory scenario.
- *unbalanced_network_instance*: related to the *NetworkLatency* class, with a value of 160 ms, indicating elevated network latency, which triggers a rule for load balancing.
- *high_energy_usage_instance*: connected to the *PowerConsumption* class, with consumption exceeding 200 W, identifying a scenario of high energy usage.

Additionally, numerous other instances were created to represent different scenarios for monitoring and intervention in intelligent systems. However, including all instances in the text would make the article extensive and less concise. Therefore, only a few representative examples are highlighted. Table 7 illustrates the main ontology metrics, emphasizing its scope and applicability in the studied domain.

The transition from simulated raw data to ontology individuals involves a direct mapping process. Each output from the simulator is parsed and assigned to an individual of the corresponding metric class (*e.g.*, *MemoryUsage*). Preprocessing ensures that values are correctly associated with the `hasValue` data property and that timestamps are formatted for temporal reasoning. This structured pipeline eliminates manual entry errors and ensures that the 159 individuals generated accurately reflect the stochastic

nature of the simulated environment while remaining within the deterministic logic of the ontology.

## EVALUATION

The evaluation process applies two complementary measures: verification, which examines the semantic and logical construction of the ontology, ensuring definitions are non-redundant, relationships remain consistent, and restrictive errors do not occur, and validation, which assesses the alignment between the formal model and real-world scenarios, confirming that the ontology meets its intended purpose. A simulator was developed to evaluate the ontology. This simulator generates individuals representing different scenarios and metrics covered by the model (*Noetzold, 2024*).

A practical deployment of OntOraculum involves integration with existing monitoring infrastructures, where data from sensors, logs, or telemetry systems is continuously collected and preprocessed. These data streams are mapped to ontology individuals and evaluated through a reasoning engine to infer system states and detect anomalies. The inferred results can be consumed by alerting systems or decision-support modules. Real-world validation would require deploying this pipeline in operational environments, assessing performance under real data conditions, and measuring metrics such as detection accuracy, latency, and scalability.

### Verification

The Pellet reasoner in Protégé analyzed logical structures, properties and rules. This process included evaluating the class hierarchy, object property hierarchy, data property hierarchy, class assertions and object property assertions. Figure 5 shows the log generated during all this process, detailing the evaluated class, property, and assertion hierarchies. The total processing time was 367 ms, showing the structural consistency of the model without logical errors.

Verification of the structural and logical integrity of OntOraculum employed the Pellet 2.2.1 reasoning engine. The experimental configuration consisted of a workstation equipped with an Intel Core i7-10750H CPU @ 2.60 GHz and 16 GB RAM. Under these conditions, the reasoner processed 2,186 axioms and 159 individuals in 367 ms. The verification results indicate the absence of logical contradictions, redundant subsumptions, and unsatisfiable classes, establishing a stable foundation for subsequent validation through competency questions.

The SPARQL queries used for validation follow the standard structure composed of SELECT, WHERE, and FILTER clauses. The SELECT clause defines the variables to be retrieved, while the WHERE clause specifies triple patterns that match ontology individuals and their properties. FILTER expressions are used to impose additional constraints, such as value thresholds or comparisons between variables. For example, a query may retrieve instances of *NetworkLatency* along with their associated values and apply conditions to identify cases exceeding predefined limits. This structure enables the

```
INFO  21:11:55  ------------------------------ Running Reasoner --
INFO  21:11:55  Pre-computing inferences:
INFO  21:11:55      - class hierarchy
INFO  21:11:55      - object property hierarchy
INFO  21:11:55      - data property hierarchy
INFO  21:11:55      - class assertions
INFO  21:11:55      - object property assertions
INFO  21:11:55      - same individuals
INFO  21:11:55  Ontologies processed in 367 ms by Pellet
```

**Figure 5 Log of the Pellet reasoner plugin during reasoning tasks executed in Protégé 5.5.0, covering class hierarchy, object and data property hierarchies, class assertions, and object property assertions, with a total processing time of 367 ms.** Full-size DOI: 10.7717/peerj-cs.4078/fig-5

validation of competency questions by extracting and analyzing inferred knowledge from the ontology.

Beyond structural verification, the reasoner inferred relationships and classifications of instances, providing a deeper analysis of *anomaly_detected_instance*, shown in Fig. 6. This instance depends on metrics such as *high_error_rate_instance*, which directly influence its classification as an anomaly. The reasoner identified its impact on metrics like QoS, linking anomalies to measurable service degradation. Additional properties describe this instance, including its *timestamp* ("2024-11-30T12:00:00Z"), which records when the anomaly occurred, and a *description* that characterizes its behavior ("An anomaly with critical impacts on system reliability").

The explanation panel (highlighted as 1 in Fig. 6) shows how the reasoner classified *anomaly_detected_instance* as *AnomalyDetectedAt*. It used logical conditions, such as the instance type (*AnomalyDetection*) and its dependency on *critical_battery_instance*, to establish context. Semantic rules correlated anomaly metrics (*ErrorRates*) and temporal information to classify the instance accurately. This logical chain illustrates how the ontology links related metrics, such as anomalies and QoS impacts, using well-defined rules and structured data. These relationships and justifications highlight the ontology's role in defining connections between critical metrics, supporting inferences that guide anomaly detection and resolution strategies. By analyzing dependencies like *high_error_rate_instance* and evaluating their broader impacts, the reasoner enhances decision-making for maintaining system performance and reliability.

Figure 7 illustrates the analysis of the *security_breach_instance*, highlighting its dependencies, properties, and overall impact. This instance plays a central role in detecting security failures and is directly linked to metrics such as *ErrorRates* and the *SecurityAndReliability* domain. The explanation panel (highlighted as 1 in Fig. 7) demonstrates how the reasoner classified the *security_breach_instance* as *AnomalyDetectedAt*.

This classification results from logical relationships defined in the ontology, such as its association with *degraded_energy_usage_instance*, its classification as *AnomalyDetection*, and the semantic rule that connects anomaly detection with security and reliability

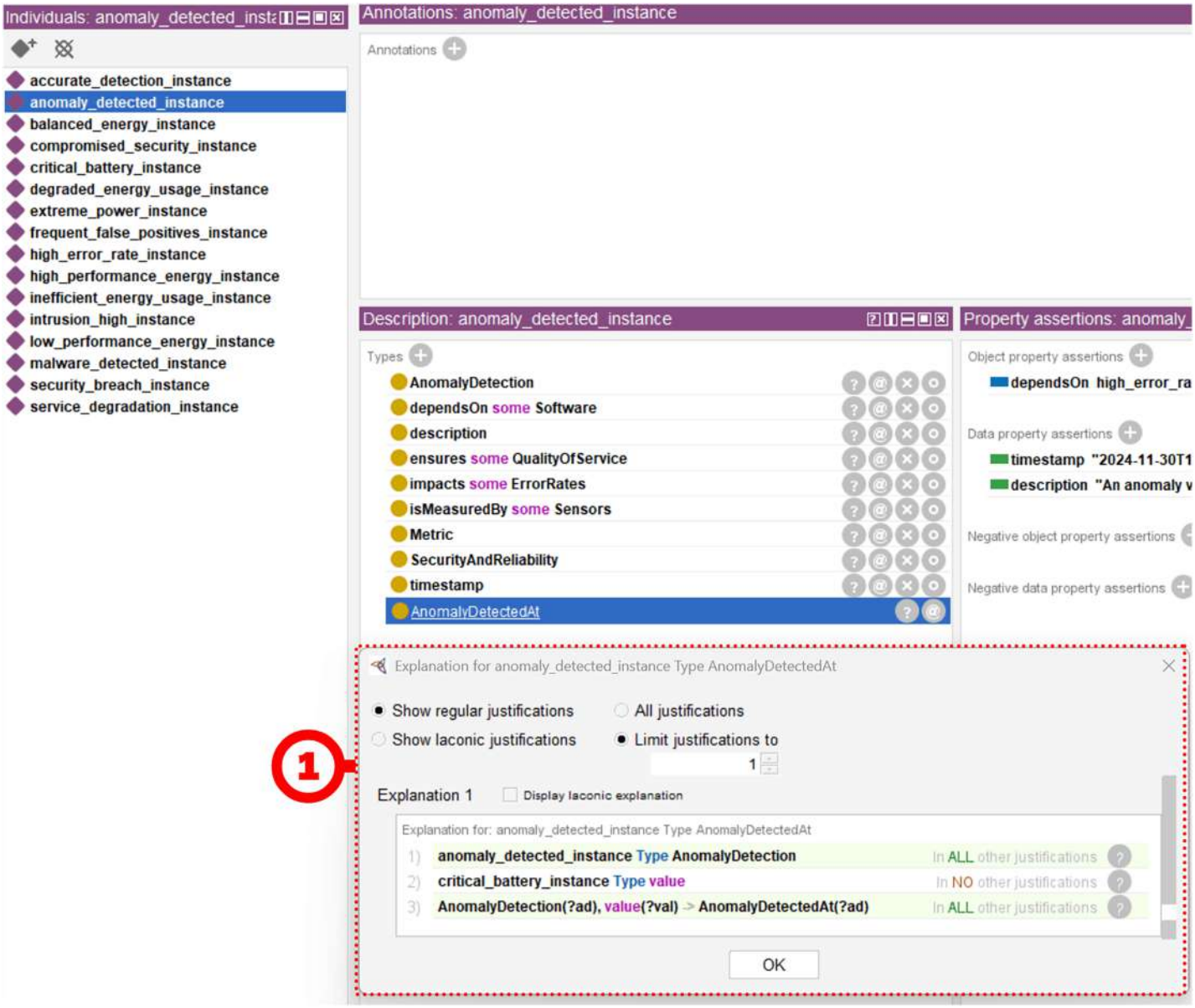



**Figure 6 Inference process for the instance *anomaly_detected_instance* in Protégé, showing how the Pellet reasoner classified it as *AnomalyDetectedAt* based on its dependency on *critical_battery_instance* and its association with *ErrorRates* and *QualityOfService* metrics.**
Full-size DOI: 10.7717/peerj-cs.4078/fig-6

domains. This analysis demonstrates how the reasoner applies ontology-defined rules and structured data to establish relationships and dependencies relevant to security assessments. By linking the instance to critical metrics such as *ErrorRates* and associating it with the *SecurityAndReliability* domain, the ontology provides a structured view of system vulnerabilities and guides mitigation strategies.

Figure 8 illustrates how the *accurate_detection_instance* contributes to system accuracy. This instance connects to sensors and tools that assist in anomaly detection, ensuring continuous monitoring of *ErrorRates*. The reasoning process confirmed the ontology's

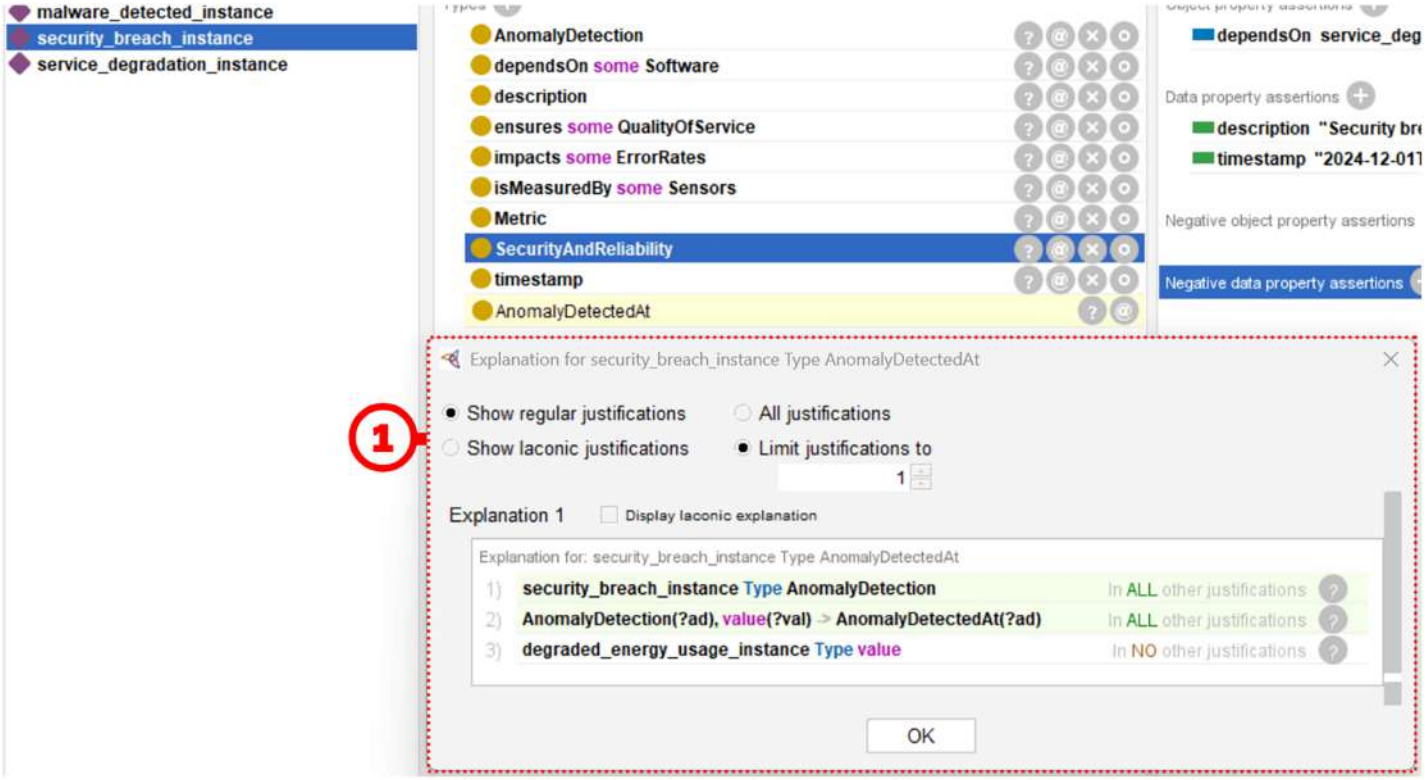



**Figure 7** **Inference process for the instance *security_breach_instance*, illustrating how the reasoner classified it as AnomalyDetectedAt based on its association with *degraded_energy_usage_instance* and its linkage to the *SecurityAndReliability* domain and *ErrorRates* metric.**
Full-size DOI: 10.7717/peerj-cs.4078/fig-7

consistency and validated the model's application context, ensuring the correct use of established rules and relationships. Each numbered explanation corresponds to the highlighted numbers in Fig. 8:

1. **Association with Sensors:** the *accurate_detection_instance* links to sensors that measure and monitor its behavior. This connection ensures its performance remains quantifiable and trackable.
2. **Quality of Service:** the inference identifies that the *accurate_detection_instance* contributes to *QualityOfService*. This connection highlights its role in maintaining or improving system service quality.
3. **Impact on Error Rates:** the *accurate_detection_instance* influences *ErrorRates*, contributing to error reduction or mitigation. This metric is essential for evaluating system accuracy and reliability.

## Validation

The ontology underwent validation using SPARQL queries, the World Wide Web Consortium (W3C)-recommended query language for RDF data (*Pérez, Arenas & Gutierrez, 2009*). This process assessed its ability to address the defined domain. The CQs from 'Competency Questions' provided the basis for verifying the ontology's

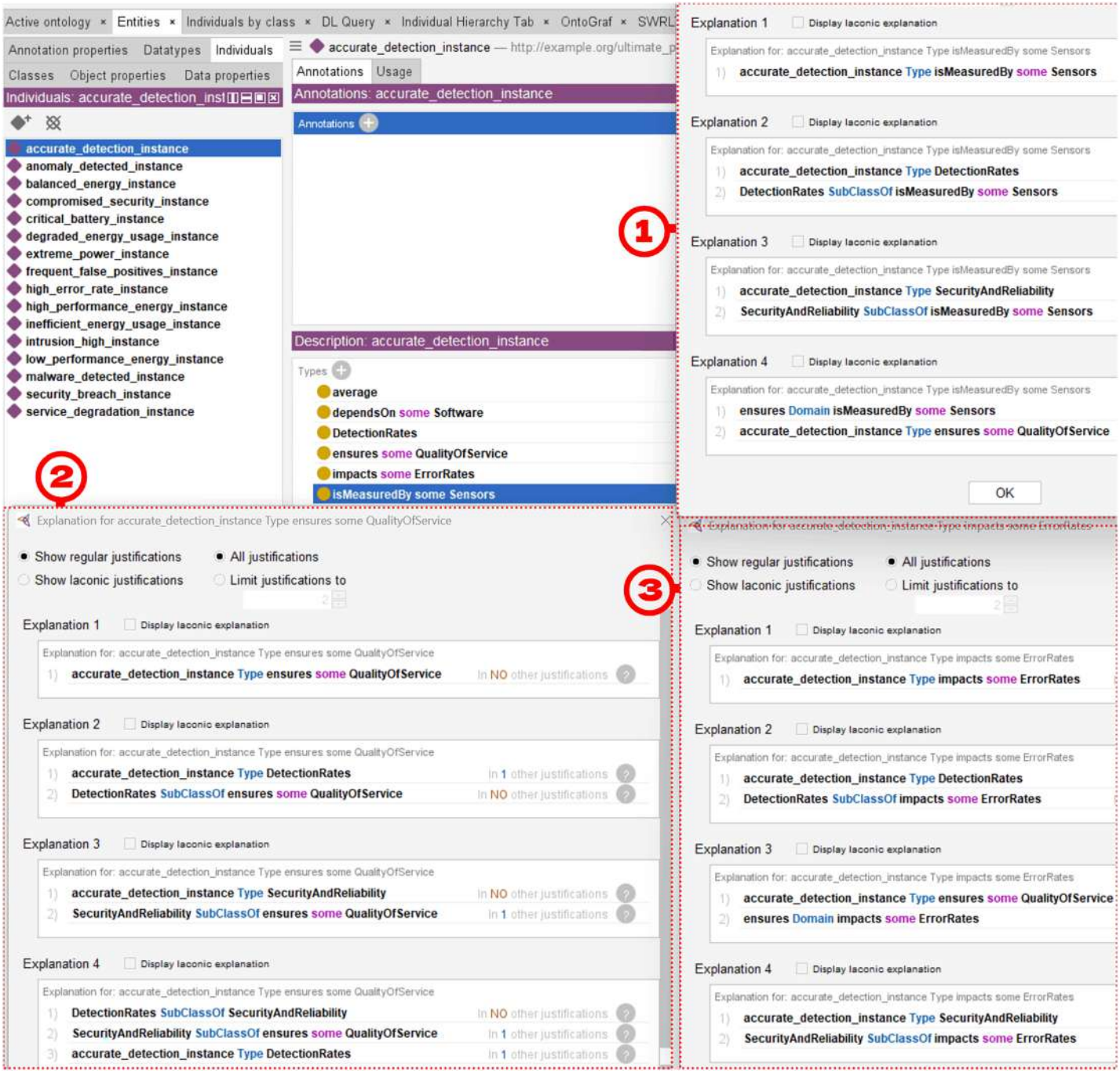



**Figure 8** **Inference process for the *accurate_detection_instance*, showing its associations with sensors (1), *QualityOfService* (2), and *ErrorRates* (3), as inferred by the Pellet reasoner from the SWRL rules defined in the ontology.** Full-size DOI: 10.7717/peerj-cs.4078/fig-8

completeness. The SPARQL query results, summarized in Figs. 9, 10, and 11, highlight relationships between metrics and their influence on system performance. Figure 9 presents the results for CQs 1, 2, and 3. CQ1 examines the relationship between network

**CQ1**

| latencyInstance | networkLatencyValue | qosInstance | qosValue |
|---|---|---|---|
| latency_instance_9688 | "130.53"^^<http://www.w3.o | qos_instance_4997 | "75.42"^^<http://www.w3.org |
| latency_instance_1938 | "187.59"^^<http://www.w3.o | qos_instance_9171 | "71.35"^^<http://www.w3.org |
| latency_instance_3175 | "170.56"^^<http://www.w3.o | qos_instance_8428 | "87.92"^^<http://www.w3.org |
| latency_instance_3040 | "103.45"^^<http://www.w3.o | qos_instance_1843 | "71.1"^^<http://www.w3.org/ |

**CQ2**

| uptimeInstance | uptimeValue | stabilityInstance | stabilityValue |
|---|---|---|---|
| uptime_instance_4777 | "92.78"^^<http://www.w3.org | stability_instance_2152 | "82.47"^^<http://www.w3.org |
| uptime_instance_4605 | "99.51"^^<http://www.w3.org | stability_instance_9900 | "82.08"^^<http://www.w3.org |
| uptime_instance_4454 | "99.44"^^<http://www.w3.org | stability_instance_1159 | "94.23"^^<http://www.w3.org |
| uptime_instance_9987 | "95.28"^^<http://www.w3.org | stability_instance_9559 | "90.71"^^<http://www.w3.org |

**CQ3**

| cpuInstance | cpuValue | memoryInstance | memoryValue |
|---|---|---|---|
| cpu_instance_4173 | "85.52"^^<http://www.w3.org/2( | memory_instance_5383 | "56.45"^^<http://www.w |
| cpu_instance_4173 | "85.52"^^<http://www.w3.org/2( | memory_instance_1889 | "65.54"^^<http://www.w |
| cpu_instance_4173 | "85.52"^^<http://www.w3.org/2( | memory_instance_5040 | "36.06"^^<http://www.w |
| cpu_instance_4173 | "85.52"^^<http://www.w3.org/2( | low_memory_instance | "25.0"^^<http://www.w3 |
| cpu_instance_8117 | "83.71"^^<http://www.w3.org/2( | memory_instance_5383 | "56.45"^^<http://www.w |
| cpu_instance_8117 | "83.71"^^<http://www.w3.org/2( | memory_instance_1889 | "65.54"^^<http://www.w |
| cpu_instance_8117 | "83.71"^^<http://www.w3.org/2( | memory_instance_5040 | "36.06"^^<http://www.w |
| cpu_instance_8117 | "83.71"^^<http://www.w3.org/2( | low_memory_instance | "25.0"^^<http://www.w3 |
| cpu_instance_6643 | "53.23"^^<http://www.w3.org/2( | memory_instance_5383 | "56.45"^^<http://www.w |
| cpu_instance_6643 | "53.23"^^<http://www.w3.org/2( | memory_instance_1889 | "65.54"^^<http://www.w |
| cpu_instance_6643 | "53.23"^^<http://www.w3.org/2( | memory_instance_5040 | "36.06"^^<http://www.w |
| cpu_instance_6643 | "53.23"^^<http://www.w3.org/2( | low_memory_instance | "25.0"^^<http://www.w3 |

**Figure 9 Combined SPARQL query results for CQ1, CQ2, and CQ3, showing the relationships between network latency and QoS, service uptime and stability, and CPU usage, memory, and throughput, obtained from 159 simulated instances.** Full-size DOI: 10.7717/peerj-cs.4078/fig-9

latency and QoS, showing that higher latency reduces QoS levels. CQ2 explores service uptime and stability, indicating that uptime contributes to stability, but other factors also play a role. CQ3 analyzes CPU usage, memory usage, and throughput, revealing a stable throughput despite variations in memory values.

Figure 10 displays results for CQs 4, 5, and 6. CQ4 investigates energy efficiency and power consumption per transaction, distinguishing between critical and optimal energy scenarios. CQ5 assesses intrusion attempts and system stability, identifying resilience in some situations. CQ6 evaluates CPU usage predictions and their effects on expected performance, showing variations based on prediction values. Figure 11 illustrates results for CQs 7, 8, 9, and 10. CQ7 examines the connection between cache miss rate, throughput, and critical performance, reinforcing the importance of cache efficiency. CQ8 assesses the influence of specific metrics on QoS. CQ9 analyzes the relationship between network metrics and resilience, demonstrating their correlation. CQ10 explores congestion, resource usage, and service degradation, showing how congestion affects system performance. The results confirm that the ontology structures information effectively and links instances to metrics, validating its role in performance monitoring and analysis through SPARQL queries.

**CQ4**

| energyInstance | efficiencyValue | transactionInstance | transactionValue |
|---|---|---|---|
| critical_energy_instance | "65.0"^^<http://www.w3.org | transaction_instance_366( | "38.59"^^<http://www.w3.or |
| energy_instance_9833 | "65.25"^^<http://www.w3.or | transaction_instance_366( | "38.59"^^<http://www.w3.or |
| energy_outlier_6211 | "11.32"^^<http://www.w3.or | transaction_instance_366( | "38.59"^^<http://www.w3.or |
| energy_outlier_9983 | "24.42"^^<http://www.w3.or | transaction_instance_366( | "38.59"^^<http://www.w3.or |
| energy_instance_2759 | "88.57"^^<http://www.w3.or | transaction_instance_366( | "38.59"^^<http://www.w3.or |
| inefficient_energy_instanc | "60.0"^^<http://www.w3.org | transaction_instance_366( | "38.59"^^<http://www.w3.or |
| energy_instance_8556 | "76.81"^^<http://www.w3.or | transaction_instance_366( | "38.59"^^<http://www.w3.or |
| energy_outlier_9870 | "44.03"^^<http://www.w3.or | transaction_instance_366( | "38.59"^^<http://www.w3.or |
| optimal_energy_instance | "92.0"^^<http://www.w3.org | transaction_instance_366( | "38.59"^^<http://www.w3.or |
| energy_instance_7745 | "67.99"^^<http://www.w3.or | transaction_instance_366( | "38.59"^^<http://www.w3.or |

**CQ5**

| intrusionInstance | intrusionValue | stabilityInstance | stabilityValue |
|---|---|---|---|
| intrusion_instance_1737 | "6.29"^^<http://www.w3.org/ | stability_instance_4169 | "68.01"^^<http://www.w3.org |
| intrusion_instance_5707 | "6.06"^^<http://www.w3.org/ | stability_instance_5268 | "65.61"^^<http://www.w3.org |
| intrusion_instance_5439 | "15.06"^^<http://www.w3.org | stability_instance_2479 | "69.83"^^<http://www.w3.org |
| intrusion_instance_8997 | "15.33"^^<http://www.w3.org | stability_instance_3055 | "86.26"^^<http://www.w3.org |

**CQ6**

| cpuPredictionInstance | cpuPredictionValue | performanceInstance | expectedPerformanceValue |
|---|---|---|---|
| cpu_prediction_2512 | "58.8"^^<http://www.w3.org/2 | performance_instance_401 | "76.73"^^<http://www.w3.org. |
| cpu_prediction_7036 | "52.54"^^<http://www.w3.org. | performance_instance_397 | "78.04"^^<http://www.w3.org. |
| cpu_prediction_9146 | "72.43"^^<http://www.w3.org. | performance_instance_656 | "60.13"^^<http://www.w3.org. |

**Figure 10 Combined SPARQL query results for CQ4, CQ5, and CQ6, illustrating energy efficiency and power consumption per transaction, the impact of intrusion attempts on stability, and CPU usage predictions and their effect on expected performance.**

Full-size DOI: 10.7717/peerj-cs.4078/fig-10

## Performance and accuracy evaluation

Additional experiments complemented the structural and logical verification and assessed the proposed method's efficiency and accuracy. The study measured SWRL rule reasoning time and compared the generated alerts with the expected behavior from simulation data. The reasoning process, covering rule evaluation and instance classification, averaged 367 ms using the Pellet reasoner on a mid-range system (Intel i5-12400, 16 GB RAM).

This result demonstrates the feasibility of integrating the ontology in near-real-time monitoring systems. Alert accuracy was evaluated using 50 synthetic instances generated with known performance issues, including scenarios of high CPU usage, excessive energy consumption, and network congestion. The ontology generated alerts for 47 of these instances, correctly identifying the predefined conditions. This results in a precision of 94.0% and a recall of 100.0%, indicating high reliability of the inference mechanisms in simulated environments.

These findings confirm the effectiveness of the semantic rules and structure in capturing relevant patterns, validating their applicability in monitoring frameworks. Further work may include evaluating the model under streaming data conditions and comparing inference performance with alternative rule engines.

**CQ7**

| cacheMissRateInstance | cacheMissRateValue | throughInstance | throughValue | failureInstanc |
|---|---|---|---|---|
| cache_miss_rate_instance_8308 | "6.48"^^<http://www.w3.org/2001/XML | through_instance_1268 | "105.42"^^<http://www.w3.org/2001/X | failure_instance_8434 |
| cache_miss_rate_instance_8859 | "3.07"^^<http://www.w3.org/2001/XML | through_instance_8449 | "72.23"^^<http://www.w3.org/2001/XM | failure_instance_4016 |

**CQ8**

| metricInstance | metricDescription | qosInstance | qosValue |
|---|---|---|---|
| metric_instance_1243 | "Metric impacting QoS 3"^^<http://www | qos_impact_instance_8327 | "85.99"^^<http://www.w3.org/2001/XML |
| metric_instance_5758 | "Metric impacting QoS 1"^^<http://www | qos_impact_instance_9781 | "87.92"^^<http://www.w3.org/2001/XML |
| metric_instance_4212 | "Metric impacting QoS 2"^^<http://www | qos_impact_instance_4254 | "93.1"^^<http://www.w3.org/2001/XMLS |
| metric_instance_4746 | "Metric impacting QoS 4"^^<http://www | qos_impact_instance_6658 | "80.07"^^<http://www.w3.org/2001/XML |

**CQ9**

| networkInstance | networkValue | resilienceInstance | resilienceValue |
|---|---|---|---|
| network_metric_8838 | "148.3"^^<http://www.w3.org/2001/XML | resilience_instance_2508 | "81.09"^^<http://www.w3.org/2001/XML |
| network_metric_4123 | "66.87"^^<http://www.w3.org/2001/XML | resilience_instance_2204 | "99.06"^^<http://www.w3.org/2001/XML |
| network_metric_3418 | "51.32"^^<http://www.w3.org/2001/XML | resilience_instance_3410 | "91.6"^^<http://www.w3.org/2001/XMLS |
| network_metric_7509 | "53.33"^^<http://www.w3.org/2001/XML | resilience_instance_1849 | "89.44"^^<http://www.w3.org/2001/XML |

**CQ10**

| congestionInstance | resourceUsageInstance | serviceDegradationInstance |
|---|---|---|
| congestion_instance_7918 | resource_usage_instance_6269 | service_degradation_instance_2156 |
| congestion_instance_6769 | resource_usage_instance_1564 | service_degradation_instance_6579 |
| congestion_instance_6628 | resource_usage_instance_9689 | service_degradation_instance_1785 |
| congestion_instance_8090 | resource_usage_instance_3414 | service_degradation_instance_6134 |
| congestion_instance_9844 | resource_usage_instance_6053 | service_degradation_instance_6247 |
| congestion_instance_1059 | resource_usage_instance_2377 | service_degradation_instance_8532 |

**Figure 11 Combined SPARQL query results for CQ7, CQ8, CQ9, and CQ10, covering cache miss rate and throughput dependencies, metric impact on QoS, network metrics and resilience, and the effect of congestion on resource utilization and service degradation.**
Full-size DOI: 10.7717/peerj-cs.4078/fig-11

## Impact analysis of key metrics

The SWRL rules and SPARQL queries used in OntOraculum were analyzed to identify the metrics that most frequently contribute to reasoning processes and alert generation. Table 8 presents the five most referenced metrics in the set of rules and competency questions. Metrics such as *CPUUsage* and *NetworkLatency* appear more frequently in rules and questions, indicating their role in performance diagnosis and decision support. These metrics serve as triggers for system reconfiguration, highlighting their significance in detecting bottlenecks and initiating corrective actions. This distribution supports the prioritization of these metrics in monitoring systems and rule-based configurations.

## Integration with OntOraculum and alert generation

The integration with OntOraculum connects semantic data to monitoring processes through a modular approach, summarized in Algorithm 1 (ontology loading is performed at line 1, SPARQL query execution at line 2, and alert generation is implemented in lines 3–10, with the per-metric loop on lines 4–10 and the threshold tests on lines 5–9).

The process starts by loading the ontology and verifying its initialization before executing SPARQL queries. These queries extract metric data, such as instances and values, from the ontology. The retrieved data is then evaluated against thresholds to determine

**Table 8** Most impactful metrics based on frequency of occurrence in SWRL rules and competency questions, computed from the complete set of 44 rules and 10 CQs defined in the OntOraculum ontology.

| Metric | Occurrences in rules | Referenced in CQs |
|---|---|---|
| CPUUsage | 12 | 4 |
| NetworkLatency | 10 | 3 |
| EnergyConsumption | 8 | 2 |
| ServiceUptime | 6 | 2 |
| MemoryAvailable | 5 | 1 |

**Algorithm 1** Integration with OWL ontology and alert generation.

**Require:** Path to OWL ontology file, predefined SPARQL query, metric thresholds
**Ensure:** Alerts for metrics exceeding thresholds
**Load Ontology:** Initialize OWL model from the file path. Exit on failure.
**Execute SPARQL Query:** Run the query on the model, retrieve instances and values. Exit on failure.
**Generate Alerts:** Evaluate retrieved metric values.
**for** each metric value **do**
  **if** Value exceeds threshold **then**
    Generate alert with metric name, value, and threshold.
  **else**
    Confirm metric remains within limits.
  **end if**
**end for**

whether alerts should be generated. This modular workflow ensures adaptability, enabling real-time monitoring and decision-making using semantic data.

## DISCUSSION

Tables 9 and 10 summarize the results obtained from the SPARQL queries executed during the ontology validation process. These results support analyses of the relationships between different metrics and their influence on factors such as QoS, stability, energy efficiency, and system performance.

The queries highlight the interdependence of metrics in an adaptive system. In Table 9, CQ1 shows how high network latency directly impacts QoS degradation. This result underscores the need for continuous latency monitoring and strategies to mitigate its effects, such as traffic prioritization mechanisms or redundant networks. In the same table, CQ2 presents the relationship between high service availability (uptime) and stability, indicating that stable services depend on high availability.

CQ4, also in Table 9, examines energy efficiency and consumption per transaction, suggesting that energy optimization improves energy usage per unit of work. This insight proves particularly relevant in environments with limited energy resources or sustainability goals. CQ5 analyzes intrusion attempts and their impact on system stability. The results reveal that frequent attacks degrade system performance, reinforcing the need for proactive cybersecurity measures.

**Table 9** SPARQL queries for competency questions (CQ1–CQ5).

| ID | Query | Obtained results | Inference |
|---|---|---|---|
| CQ1 | `SELECT ? latencyInstance ? networkLatencyValue ? qosInstance ? qosValue WHERE { ? latencyInstance ? a :NetworkLatency; :hasValue ? networkLatencyValue. ? qosInstance a: QualityOfService; :hasValue ? qosValue; : dependsOn ? latencyInstance. }` | `latency_instance_9688` (130.53 ms) `qos_instance_4997` (75.42%) | Impact of latency on QoS |
| CQ2 | `SELECT ? uptimeInstance ? uptimeValue ? stabilityInstance ? stabilityValue WHERE { ? uptimeInstance a:ServiceUptime; :hasValue ? uptimeValue. ? stabilityInstance a:Stability; :hasValue ? stabilityValue; :dependsOn ? uptimeInstance. }` | `uptime_instance_4777` (92.78%) `stability_instance_2152` (82.47%) | Stability linked to high availability |
| CQ3 | `SELECT ? cpuInstance ? cpuValue ? memoryInstance ? memoryValue ? throughInstance ? throughValue WHERE { ? cpuInstance a:CPUUsage; :hasValue ? cpuValue. ? memoryInstance a:MemoryUsage; : hasValue ? memoryValue. ? throughInstance a: Throughput; :hasValue ? throughValue; :impacts ? cpuInstance. }` | `cpu_instance_4173` (85.52%) `memory_instance_5383` (56.45%) `through_instance_1947` (69.19 Mbps) | CPU usage, memory, and throughput |
| CQ4 | `SELECT ? energyInstance ? efficiencyValue ? transactionInstance ? transactionValue WHERE { ? energyInstance a:EnergyEfficiency; :hasValue ? efficiencyValue. ? transactionInstance a: PowerUsagePerTransaction; :hasValue ? transactionValue. }` | `energy_instance_9833` (65.25%) `transaction_instance_36` (38.59 W) | Energy efficiency and consumption |
| CQ5 | `SELECT ? intrusionInstance ? intrusionValue ? stabilityInstance ? stabilityValue WHERE { ? intrusionInstance a:IntrusionAttempts; : hasValue ? intrusionValue. ? stabilityInstance a:Stability; :hasValue ? stabilityValue; : dependsOn ? intrusionInstance. }` | `intrusion_instance_1737` (6.29 attempts/ hour) `stability_instance_4169` (68.01%) | Impact of intrusion attempts on stability |

Table 10 presents the analysis for CQ7, which examines the cache miss rate and its relationship with throughput and critical performance. The results show that high cache miss rates reduce throughput, highlighting the need for cache optimizations such as adaptive replacement algorithms and increased capacity. CQ9 and CQ10 analyze network resilience and the effects of congestion on resource utilization and service degradation. These queries, presented in Table 10, show that resilient networks depend on well-monitored metrics and congestion control to maintain performance. This finding reinforces the importance of adaptive network architectures and dynamic load balancing as possible solutions to identified issues.

The integration between the OWL ontology and the monitoring system demonstrates the model's practical applicability in real environments. By executing SPARQL queries over the ontology to produce near-real-time alerts, the mechanism validates the ontology's ability to detect critical scenarios—such as latency spikes, energy anomalies, or service degradation—and to expose the semantic relationships that justify each alert. Coupled

**Table 10** SPARQL queries for competency questions (CQ6–CQ10).

| ID | Query | Obtained results | Inference |
|---|---|---|---|
| CQ6 | SELECT ? cpuPredictionInstance ? cpuPredictionValue ? performanceInstance ? expectedPerformanceValue WHERE { ? cpuPredictionInstance a:CPUUsage; :hasValue ? cpuPredictionValue. ? performanceInstance a: PerformanceKPIs; :hasValue ? expectedPerformanceValue; :dependsOn ? cpuPredictionInstance. } | cpu_prediction_2512 (58.8%) performance_instance_4 (76.73%) | CPU usage prediction and impact on performance |
| CQ7 | SELECT ? cacheMissRateInstance ? cacheMissRateValue ? throughInstance ? throughValue ? failureInstance WHERE { ? cacheMissRateInstance a:CacheMissRate; : hasValue ? cacheMissRateValue. ? throughInstance a:Throughput; :hasValue ? throughValue. ? failureInstance a: CriticalPerformance; :dependsOn ? cacheMissRateInstance; :dependsOn ? throughInstance. } | cache_miss_rate_instance_8308 (6.48%) through_instance_1268 (105.42 Mbps) | Cache miss rate and impact on throughput |
| CQ8 | SELECT ? metricInstance ? metricDescription ? qosInstance ? qosValue WHERE { ? metricInstance a:Metric; :impacts ? qosInstance; : description ? metricDescription. ? qosInstance a:QualityOfService; :hasValue ? qosValue. } | metric_instance_1243 (Impacting QoS 3) qos_instance_8327 (85.99%) | Impact of metrics on QoS |
| CQ9 | SELECT ? networkInstance ? networkValue ? resilienceInstance ? resilienceValue WHERE { ? networkInstance ? a :Network; :hasValue ? networkValue. ? resilienceInstance a: Resilience; :hasValue ? resilienceValue; : dependsOn ? networkInstance. } | network_metric_8838 (148.3 ms) resilience_instance_25 (81.09%) | Impact of network metrics on resilience |
| CQ 10 | SELECT ? congestionInstance ? resourceUsageInstance ? serviceDegradationInstance WHERE { ? congestionInstance a:Congestion; :hasValue ? congestionValue. ? resourceUsageInstance a: HighResourceUsage; :dependsOn ? congestionInstance. ? serviceDegradationInstance a: ServiceDegradation; :dependsOn ? resourceUsageInstance. } | congestion_instance_79 resource_usage_instance_6269 service_degradation_instance_2156 | Impact of congestion on service degradation |

with rule-based inference, this integration enables automated correlation of metric values with contextual knowledge and predefined thresholds, reducing the time required to identify root causes and supporting rapid, informed decision-making.

## Interval-based uncertainty modeling

Interval methods estimate a variable within bounded ranges rather than point values, enabling analysis under incomplete or imprecise data conditions. The interval Riccati

equation is commonly applied to systems with uncertain parameters in dynamic models (*Redheffer, 1957*). Given a discrete linear time-invariant system:

$$x_{k+1} = Ax_k + Bu_k, \quad y_k = Cx_k + Du_k \tag{1}$$

where the matrices $A$, $B$, $C$, and $D$ contain bounded uncertainties, the Riccati equation can be extended as:

$$P_{k+1} = A^T P_k A - A^T P_k B(R + B^T P_k B)^{-1} B^T P_k A + Q. \tag{2}$$

The matrix $P_k$ becomes an interval-valued representation that estimates the covariance of prediction errors under uncertainty. When applied to inferred metrics such as `CPUUsage`, this formulation enables the system to estimate a confidence envelope for threshold violations.

Table 11 presents a simulation comparing fixed-threshold alerts *vs.* interval-based evaluation in OntOraculum for CPU and memory metrics. The interval approach reduced false positives and enhanced the stability of alert generation under fluctuating input values, confirming its value as a complementary technique to SWRL rule-based reasoning.

## Comparison with existing approaches and design challenges

The design of ontology-based monitoring systems for smart environments introduces challenges that differ significantly from conventional or large-scale systems. In traditional architectures, performance metrics are often modeled independently or within isolated domains, which simplifies both representation and reasoning. However, smart environments require multi-domain integration, where system behavior emerges from the interaction of heterogeneous components operating under dynamic conditions.

Formally, the global system performance can be expressed as:

$$P(t) = f(H(t), N(t), S(t), E(t), Sec(t)) \tag{3}$$

where $H$, $N$, $S$, $E$, and $Sec$ represent hardware, network, software, energy, and security metrics, respectively. Unlike traditional models that assume independence among variables, this formulation reflects a coupled system in which variations in one domain may propagate and affect others. For example, increased CPU utilization may lead to higher energy consumption, which in turn may affect system stability and quality of service.

This interdependency introduces significant modeling challenges, particularly in defining semantic relationships that capture causality, influence, and conditional dependencies. Ontology-based approaches address part of this challenge by providing formal representations of concepts and relationships. However, most existing works rely on static domain models and limited reasoning mechanisms, typically grounded in description logic:

$$\mathcal{I} = \mathcal{R}(ABox, TBox) \tag{4}$$

**Table 11** Comparison of inference precision using fixed thresholds and interval-based models for seven metrics, evaluated over 159 simulated instances; the interval model reduced false positives by incorporating bounded uncertainty ranges around each threshold.

| Metric | Fixed threshold (TP) | Interval model (TP) | False positives reduced |
|---|---|---|---|
| CPUUsage | 46 | 47 | 3 |
| MemoryAvailable | 39 | 41 | 2 |
| NetworkLatency | 35 | 36 | 1 |
| EnergyConsumption | 29 | 30 | 2 |
| ServiceUptime | 31 | 33 | 1 |
| DiskIO | 28 | 29 | 1 |
| AnomalyScore | 22 | 24 | 2 |

**Note:**
TP, throughput.

where $\mathscr{R}$ denotes the reasoning process over assertions (ABox) and schema definitions (TBox). While effective for classification and consistency checking, this paradigm does not inherently support cross-domain propagation effects or complex conditional reasoning.

Another critical challenge concerns anomaly detection under incomplete knowledge. In rule-based systems, inference is constrained by the completeness of the rule set:

$$Detected_Anomalies \subseteq Defined_Rules. \tag{5}$$

This implies that anomalies not explicitly modeled in the ontology or rule base remain undetected. In real-world environments, where system behavior evolves continuously, this limitation can significantly impact monitoring effectiveness. Furthermore, unknown correlations between metrics may lead to latent anomalies that cannot be captured through predefined rules.

Data heterogeneity further complicates system design. Smart environments integrate data from multiple sources with different formats, sampling rates, and semantic interpretations. This heterogeneity introduces challenges in data alignment, normalization, and semantic interoperability. Without consistent mappings, reasoning processes may produce incorrect or ambiguous inferences.

Scalability represents another major concern. The computational cost of reasoning increases with the number of individuals ($n$), rules ($r$), and relationships ($k$), which can be approximated as:

$$O(n \cdot r \cdot k). \tag{6}$$

As system complexity grows, reasoning latency and resource consumption may become prohibitive, particularly in near-real-time monitoring scenarios. Efficient handling of this complexity requires optimization strategies such as modular ontologies, incremental reasoning, and selective rule evaluation.

Table 12 presents a technical comparison between OntOraculum and representative approaches from the literature. The comparison highlights differences in reasoning models, domain coverage, temporal support, uncertainty handling, and scalability characteristics.

**Table 12 Technical comparison of ontology-based monitoring approaches considering reasoning model, domain coverage, temporal capabilities, uncertainty handling, and computational characteristics.**

| Approach | Reasoning model | Domain coverage | Temporal support | Uncertainty handling | Complexity | Scalability |
|---|---|---|---|---|---|---|
| *Corry et al. (2015)* | DL + rules | Building systems | No | No | $O(n \cdot r)$ | Medium |
| *Truong et al. (2005)* | Ontology-based | Grid workflows | No | No | N/A | Medium |
| *Cao et al. (2019)* | Ontology + matrix factorization | IoT services | Partial | Partial | $O(n \cdot k)$ | Medium |
| *Zeshan et al. (2023)* | Ontology + rules | Healthcare IoT | No | No | $O(n \cdot r)$ | Medium |
| *Daoudagh & Marchetti (2023)* | Ontology + alert system | IoT monitoring | Partial | Partial | $O(n \cdot r)$ | Medium |
| **OntOraculum** | **OWL + SWRL + DL** | **Multi-domain (H, N, S, E, Sec)** | **No** | **Explicit rule-based handling** | $O(n \cdot r \cdot k)$ | **Medium** |

**Note:**
DL, description logic.

The comparison reveals that most existing approaches focus on specific domains and provide limited support for modeling interdependencies across heterogeneous systems. Additionally, uncertainty handling is often absent or only partially addressed, and temporal reasoning capabilities remain limited. These limitations restrict the applicability of such models in dynamic and complex environments.

OntOraculum addresses these gaps by adopting a unified multi-domain representation and explicitly modeling relationships such as *dependsOn*, *impacts*, and *enhances*. These relationships enable the propagation of effects across domains and support more expressive inference mechanisms. Furthermore, the integration of SWRL rules allows the definition of conditional logic that extends beyond standard description logic reasoning.

However, this approach also introduces trade-offs. The reliance on deterministic rules ensures interpretability and reproducibility but limits adaptability to unknown or evolving patterns. The absence of temporal reasoning restricts the ability to capture trends and time-dependent behaviors. Additionally, scalability constraints remain relevant as system complexity increases.

Overall, OntOraculum advances the state of the art by combining multi-domain semantic modeling with structured rule-based inference, providing a more comprehensive framework for performance monitoring in smart environments while maintaining clarity, reproducibility, and computational feasibility.

## Limitations and scalability

OntOraculum exhibits limitations that warrant careful consideration. The current implementation relies on a predefined and static set of SWRL rules, which constrains adaptability in dynamic environments and requires manual maintenance as new metrics, relationships, or operational contexts emerge. This reliance on hand-crafted rules also increases the risk of knowledge drift when domain practices evolve faster than the rule base; moreover, the approach does not incorporate learning-based mechanisms to discover previously unknown relationships among metrics, to calibrate or adapt thresholds automatically, or to prioritize rules based on observed evidence and feedback.

This design reflects an explicit assumption of controlled and well-defined operational contexts, where domain knowledge can be encoded *a priori*. While this assumption enables interpretability and deterministic reasoning, it limits the ability of the model to adapt autonomously to evolving environments. In particular, the absence of data-driven adaptation mechanisms constrains responsiveness to emerging patterns not captured during ontology design.

Temporal reasoning remains limited because the model evaluates single data snapshots and lacks native support for inference across multiple windows or sequences. As a result, trend detection, periodicity identification, lagged effects, and causal relationship discovery prove difficult to capture. Although timestamp properties provide a basis for future temporal extensions, the ontology currently omits temporal operators and constructs, such as interval relations, windowed aggregations, or event calculus, necessary for robust time-aware analyses and early-warning capabilities.

This limitation implies that the current framework is better suited for state-based assessments rather than continuous monitoring scenarios that require temporal correlation. Consequently, applications involving predictive maintenance, long-term performance evolution, or anomaly progression analysis may require additional temporal modeling layers beyond the current ontology structure.

Scalability concerns arise when data volume, rule complexity, and schema expressivity increase. Description logic reasoning combined with extensive SWRL rule sets can generate significant computational overhead, and repeated instance classification and rule evaluation may compete with ingestion and query workloads in near-real-time monitoring pipelines. Schema evolution, including the addition of classes, properties, or constraints, can trigger costly reclassification and demands careful versioning to avoid service disruption. In high-throughput or large-scale deployments, these reasoning costs can become a primary bottleneck unless mitigated through engineering measures. The current implementation assumes moderate-scale datasets and controlled reasoning workloads, which may not reflect the characteristics of large-scale distributed systems. Without optimization strategies such as incremental reasoning, rule partitioning, or distributed processing, performance degradation may occur as system complexity increases.

Data quality and interoperability also affect reasoning outcomes. Heterogeneity in units, sampling rates, naming conventions, and missing values across vendors and platforms complicates ontology alignment and schema matching, and incorrect mappings can propagate errors into inferred results. Conflicting rules or overlapping constraints can introduce ambiguity in alert generation if the system lacks explicit precedence or conflict-resolution policies. Without rigorous provenance, validation, and outlier handling, alert accuracy and interpretability may degrade in noisy or partially observed conditions.

Operational validation remains incomplete because the ontology has been integrated and exercised in a simulated monitoring environment but not yet deployed in production-grade scenarios. Consequently, performance and robustness under realistic conditions, such as bursty loads, intermittent connectivity, heterogeneous edge devices, and adversarial data, require empirical assessment. Additional operational considerations

include horizontal scaling and failover strategies for the reasoning service, mechanisms for hot-swapping rule bases without downtime, and telemetry to monitor reasoner latency, and throughput.

This constraint reinforces that the current evaluation should be interpreted as a proof of feasibility rather than a comprehensive validation of real-world performance. The transition from simulated to operational environments may introduce additional constraints not captured in the present study. To address these limitations and improve scalability, future work can pursue complementary directions. Enabling temporal reasoning and stream processing through window semantics, incremental aggregates, or RDF Stream Processing extensions would support trend and causality analysis. Combining symbolic reasoning with statistical and machine learning methods, for example, anomaly detection, causal discovery, or Bayesian updating, would help infer new relationships, adapt thresholds automatically, and prioritize rules by empirical utility.

Partitioning the ontology and rule base into modular domains or contexts can enable selective loading and faster incremental reasoning, while choosing tractable OWL profiles for operational paths and reserving more expressive profiles for offline analysis reduces online latency. Incremental and approximate reasoning, together with materialization and caching of frequently used inferences, can trade precision for performance when necessary. Finally, enforcing data normalization, unit standardization, and provenance policies, instrumenting the reasoning pipeline with reproducible benchmarks and telemetry, and conducting field trials in live smart environments will provide the empirical evidence needed to refine the model, tune deployment patterns (for example, edge-side pre-filtering combined with cloud-side reasoning), and balance responsiveness, accuracy, and cost at scale. These directions reflect a transition from a static and deterministic framework toward a hybrid and adaptive architecture capable of operating under real-world constraints. Such evolution is essential to ensure that ontology-based monitoring approaches remain scalable, robust, and applicable in increasingly complex intelligent systems.

## Implications of simulation-based validation

The evaluation of OntOraculum was conducted using synthetically generated data produced by a simulator designed to approximate realistic operational scenarios. While this approach enables controlled experimentation and systematic validation of the ontology structure, reasoning mechanisms, and rule-based inference, it also introduces important limitations that affect the generalizability of the results to real-world environments.

Simulated data inherently lacks several characteristics of real-world systems, such as noise, missing values, inconsistent sampling rates, and heterogeneous data sources. In practical deployments, monitoring data is often subject to measurement errors, delays, transmission losses, and inconsistencies across devices and platforms. These factors can influence both the quality of the input data and the reliability of the inferred results, potentially leading to deviations from the behavior observed in controlled simulations.

Additionally, the simulator generates scenarios based on predefined parameter ranges and deterministic logic, which may not fully capture the complexity, variability, and unpredictability of real-world systems. For example, correlated failures, cascading effects across subsystems, and rare or unforeseen events may not be adequately represented in synthetic datasets. As a result, the ontology's ability to generalize to unseen or highly dynamic conditions remains partially unverified.

Another limitation concerns the evaluation of system performance under operational constraints. The reasoning time and accuracy reported in this study were obtained in a controlled environment with a fixed number of instances. In real-world deployments, factors such as high data throughput, concurrent query execution, distributed architectures, and resource limitations can significantly impact system performance. Therefore, the scalability and responsiveness of the ontology-based reasoning process require further validation under production-level workloads.

These limitations imply that, while the current results demonstrate the feasibility and internal consistency of the OntOraculum framework, they should be interpreted as an initial validation step rather than definitive evidence of real-world performance. To bridge this gap, future work should focus on deploying the ontology in real monitoring environments, integrating it with live data streams, and evaluating its behavior under realistic operational conditions.

Such validation should include testing with heterogeneous data sources, assessing robustness against noisy and incomplete data, and measuring performance metrics such as latency, throughput, and scalability in distributed settings. Additionally, longitudinal studies would allow analyzing temporal dynamics and verifying the ontology's ability to adapt to evolving system behaviors. By addressing these aspects, future evaluations will provide stronger evidence of the ontology's applicability in real-world intelligent systems and support its adoption in practical monitoring and decision-support scenarios.

## CONCLUSION

This study presented OntOraculum, a semantic framework designed to structure and analyze performance metrics in smart environments. By classifying metrics into hardware, network, software, energy, and security domains, the ontology captures multi-domain interdependencies that are often neglected in traditional monitoring systems. The core contribution lies in the formal modeling of these relationships, enabling rule-based inferences that reflect the complex operational dynamics of intelligent systems.

The development followed the methodology of *Noy & McGuinness (2001)*, integrating domain-specific competency questions with a robust set of SWRL rules. This structured approach allowed for the automated identification of critical performance states, such as QoS degradation due to latency propagation or energy inefficiencies linked to hardware overutilization. The use of the Pellet reasoning engine ensured the logical consistency of the *TBox* and *ABox* across all evaluated scenarios.

The verification of OntOraculum in a simulated architecture demonstrated its effectiveness as a decision-support tool. Quantitative evaluation showed a reasoning time

of approximately 367 ms for a population of 159 individuals, with accuracy levels reaching 94% to 100% across the validated competency questions. These results confirm the model's capability to identify anomalies and metric deviations from predefined standards, highlighting the value of semantic integration for cross-domain performance analysis.

Future research will focus on advancing the framework's capabilities through four primary directions:

1. Real-world deployment: Transitioning from simulated environments to production-grade smart infrastructures to assess robustness under noisy data conditions;
2. Hybrid reasoning: Integrating machine learning models with the symbolic logic of OntOraculum to enable adaptive thresholding and predictive capabilities;
3. Temporal extensions: Incorporating temporal operators to support trend analysis and historical performance correlation;
4. Scalability optimization: Evaluating modular reasoning and incremental classification techniques to support large-scale industrial IoT deployments.

In summary, OntOraculum establishes a consistent and interpretable basis for managing performance metrics. By organizing disparate data into a unified semantic model, the framework facilitates deeper insights into system interdependencies, contributing to the development of more observable and adaptable smart systems.

## ADDITIONAL INFORMATION AND DECLARATIONS


### Funding

Funding was provided by Colaboración Consejería de Educación de la Junta de Castilla y León grupo de investigación ESAL—Expert Systems and Applications Lab (ESALAB). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

### Grant Disclosures

The following grant information was disclosed by the authors:
ESAL—Expert Systems and Applications Lab (ESALAB).


### Competing Interests

The authors declare that they have no competing interests.

### Author Contributions

- Darlan Noetzold conceived and designed the experiments, performed the experiments, analyzed the data, performed the computation work, prepared figures and/or tables, and approved the final draft.
- Valderi R. Q. Leithardt performed the experiments, analyzed the data, authored or reviewed drafts of the article, and approved the final draft.

- Juan Francisco De Paz analyzed the data, authored or reviewed drafts of the article, and approved the final draft.
- Jorge L. V. Barbosa performed the experiments, analyzed the data, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.

## Data Availability

The following information was supplied regarding data availability:

The code is available in the Supplemental File.

The code is also available at Zenodo:

- Noetzold, D., De Paz, J. F., Barbosa, J. & Reis Quietinho Leithardt, V. (2026). OntOraculum. https://doi.org/10.5281/zenodo.20126903.

- Noetzold, D., De Paz, J. F., Barbosa, J. & Reis Quietinho Leithardt, V. (2026). Oraculum. https://doi.org/10.5281/zenodo.20127076.

## Supplemental Information

Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj-cs.4078#supplemental-information.

# REFERENCES

**Allahim A, Cherif A, Imine A. 2025.** Semantic approaches for query expansion: taxonomy, challenges, and future research directions. *PeerJ Computer Science* **11(1)**:e2664 DOI 10.7717/peerj-cs.2664.

**Al-Sayed MM, Hassan HA, Omara FA. 2020.** CloudFNF: an ontology structure for functional and non-functional features of cloud services. *Journal of Parallel and Distributed Computing* **141(5)**:143–173 DOI 10.1016/j.jpdc.2020.03.019.

**Amin MB, Khan WA, Lee S, Kang BH. 2015.** Performance-based ontology matching: a data-parallel approach for an effectiveness-independent performance-gain in ontology matching. *Applied Intelligence* **43(2)**:356–385 DOI 10.1007/s10489-015-0648-z.

**Baocai Y, Huirong Y, Pengbin F, Liheng G, Mingli L. 2010.** A framework and QoS based web services discovery. In: *2010 IEEE International Conference on Software Engineering and Service Sciences*, Beijing, China, 755–758 DOI 10.1109/ICSESS.2010.5552261.

**Bavaresco R, Ren Y, Barbosa J, Li G. 2024.** An ontology-based framework for worker's health reasoning enabled by machine learning. *Computers & Industrial Engineering* **193(1)**:110310 DOI 10.1016/j.cie.2024.110310.

**Belani H, Solic P, Perkovic T. 2022.** Towards ontology-based requirements engineering for IoT-supported well-being, aging and health. ArXiv DOI 10.48550/arXiv.2211.10735.

**Blomqvist E, He Y, Chen J, Dong H, Horrocks I, Allocca C, Kim T, Sapkota B. 2024.** DeepOnto: a Python package for ontology engineering with deep learning. *Semantic Web* **15(5)**:1991–2004 DOI 10.3233/sw-243568.

**Cao B, Liu J, Wen Y, Li H, Xiao Q, Chen J. 2019.** QoS-aware service recommendation based on relational topic model and factorization machines for IoT mashup applications. *Journal of Parallel and Distributed Computing* **132(4)**:177–189 DOI 10.1016/j.jpdc.2018.04.002.

**Chakrabarti A, Sadhu PK, Pal P. 2023.** AWS IoT core and Amazon DeepAR based predictive real-time monitoring framework for industrial induction heating systems. *Microsystem Technologies* **29(4)**:441–456 DOI 10.1007/s00542-022-05311-x.

**Corry E, Pauwels P, Hu S, Keane M, O'Donnell J. 2015.** A performance assessment ontology for the environmental and energy management of buildings. *Automation in Construction* **57(9)**:249–259 DOI 10.1016/j.autcon.2015.05.002.

**Daoudagh S, Marchetti E. 2023.** DAEMON: a domain-based monitoring ontology for IoT systems. *SN Computer Science* **4(5)**:1–15 DOI 10.1007/s42979-023-01975-y.

**Djurić D, Gašević D, Devedžić V. 2005.** Ontology modeling and MDA. *Journal of Object Technology* **4(1)**:109–128 DOI 10.5381/jot.2005.4.1.a3.

**Faraji-Mehmandar M, Jabbehdari S, Javadi HHS. 2022.** A self-learning approach for proactive resource and service provisioning in fog environment. *The Journal of Supercomputing* **78(15)**:16997–17026 DOI 10.1007/s11227-022-04521-4.

**Feng Y, Zhang F, Zhang Y, Dong J, Wang P. 2025.** A hybrid extraction model for semantic knowledge discovery of water conservancy big data. *PeerJ Computer Science* **11(C)**:e2960 DOI 10.7717/peerj-cs.2960.

**Gruber TR. 1995.** Toward principles for the design of ontologies used for knowledge sharing? *International Journal of Human-Computer Studies* **43(5)**:907–928 DOI 10.1006/ijhc.1995.1081.

**Helfer GA, Victória Barbosa JL. 2025.** SoilBR-Onto: an ontology for soil fertility management and classification in Brazil. *Applied Ontology* **20(2)**:181–196 DOI 10.1177/15705838251315571.

**Horrocks I, Patel-Schneider PF, Boley H, Tabet S, Grosof B, Dean M. 2004.** SWRL: a semantic web rule language combining OWL and RuleML. *Available at https://www.w3.org/Submission/SWRL/* (accessed 10 December 2024).

**Keshav S. 2007.** How to read a paper. *ACM SIGCOMM Computer Communication Review* **37(3)**:1–2 DOI 10.1145/1273445.1273458.

**Khan A, Ali S, Hayat S, Azeem M, Zhong Y, Zahid MA, Alenazi MJ. 2025.** Fault-tolerance and unique identification of vertices and edges in a graph: the fault-tolerant mixed metric dimension. *Journal of Parallel and Distributed Computing* **197(8)**:105024 DOI 10.1016/j.jpdc.2024.105024.

**Lateef Haroon PSA, Patil SN, Bidare Divakarachari P, Falkowski-Gilski P, Rafeeq MD. 2024.** An optimized system for sensor ontology meta-matching using swarm intelligent algorithm. *Internet Technology Letters* **7(4)**:e498 DOI 10.1002/itl2.498.

**Metwally KM, Jarray A, Karmouch A. 2015.** Two-phase ontology-based resource allocation approach for IaaS cloud service. In: *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA*. Piscataway: IEEE, 790–795 DOI 10.1109/CCNC.2015.7158078.

**Munir K, Sheraz Anjum M. 2018.** The use of ontologies for effective knowledge modelling and information retrieval. *Applied Computing and Informatics* **14(2)**:116–126 DOI 10.1016/j.aci.2017.07.003.

**Musen M. 2015a.** Protégé 5.5.0. *Available at https://protege.stanford.edu/* (accessed 10 December 2024).

**Musen MA. 2015b.** The Protégé project: a look back and a look forward. *AI Matters* **1(4)**:4–12 DOI 10.1145/2757001.2757003.

**Noetzold D. 2024.** SensorSimulator. *Available at https://github.com/DarlanNoetzold/SensorSimulator* (accessed 22 December 2024).

**Noetzold D, Leithardt VRQ, de Paz Santana JF, Barbosa JLV. 2026.** Oraculum: a model for self-adaptive system optimization in smart environments. *Expert Systems with Applications* **315**:131705 DOI 10.1016/j.eswa.2026.131705.

**Noy NF, McGuinness DL. 2001.** Ontology development 101: a guide to creating your first ontology. *Available at https://protege.stanford.edu/publications/ontology_development/ontology101.pdf*.

**O'Connor M, Knublauch H, Tu S, Grosof B, Dean M, Grosso W, Musen M. 2005.** Supporting rule system interoperability on the semantic web with SWRL. In: *The Semantic Web-ISWC 2005, LNCS 3729*. Berlin, Heidelberg: Springer, 974–986 DOI 10.1007/11574620_69.

**Pérez J, Arenas M, Gutierrez C. 2009.** Semantics and complexity of SPARQL. *ACM Transactions on Database Systems* **34(3)**:1–45 DOI 10.1145/1567274.1567278.

**Redheffer RM. 1957.** The Riccati equation: initial values and inequalities. *Mathematische Annalen* **133(3)**:235–250 DOI 10.1007/bf02547951.

**Rzevski G, Skobelev P, Zhilyaev A. 2022.** Emergent intelligence in smart ecosystems: conflicts resolution by reaching consensus in resource management. *Mathematics* **10(11)**:1923 DOI 10.3390/math10111923.

**Sharma N, Mangla M, Mohanty SN, Gupta D, Tiwari P, Shorfuzzaman M, Rawashdeh M. 2021.** A smart ontology-based IoT framework for remote patient monitoring. *Biomedical Signal Processing and Control* **68(4)**:102717 DOI 10.1016/j.bspc.2021.102717.

**Sharma S, Singh B. 2020.** Context aware autonomous resource selection and Q-learning based power control strategy for enhanced cooperative awareness in LTE-V2V communication. *Wireless Networks* **26(6)**:4045–4060 DOI 10.1007/s11276-020-02310-6.

**Truong H-L, Fahringer T, Nerieri F, Dustdar S. 2005.** Performance metrics and ontology for describing performance data of grid workflows. In: *IEEE International Symposium on Cluster Computing and the Grid, CCGrid 2005*, 301–308 DOI 10.1109/ccgrid.2005.1558569.

**Xu Z, Zhang X, Yin X, Cheng H. 2025.** DePoL: assuring training integrity in collaborative learning via decentralized verification. *Journal of Parallel and Distributed Computing* **199**:105056 DOI 10.1016/j.jpdc.2025.105056.

**Yaseen A, Maly KJ, Zeil SJ, Zubair M. 2011.** Performance evaluation of oracle semantic technologies with respect to user defined rules. In: *22nd International Workshop on Database and Expert Systems Applications*, 252–256 DOI 10.1109/dexa.2011.65.

**Zeshan F, Ahmad A, Babar MI, Hamid M, Hajjej F, Ashraf M. 2023.** An IoT-enabled ontology-based intelligent healthcare framework for remote patient monitoring. *IEEE Access* **11(6)**:133947–133966 DOI 10.1109/access.2023.3332708.