NebulOuS

A META OPERATING SYSTEM FOR BROKERING
HYPER-DISTRIBUTED APPLICATIONS ON
CLOUD COMPUTING CONTINUUMS

D2.3
NEBULOUS SEMANTIC MODELS AND RESOURCE DISCOVERY MECHANISM

[31/10/2025]

Funded by
the European Union

Grant Agreement No.	101070516
Project Acronym/ Name	NebulOuS - A META OPERATING SYSTEM FOR BROKERING HYPER DISTRIBUTED APPLICATIONS ON CLOUD COMPUTINGCONTINUUMS
Topic	HORIZON-CL4-2021-DATA-01-05
Type of action	HORIZON-RIA
Service	CNECT/E/04
Duration	36 months (starting date 1 September 2022)
Deliverable title	Nebulous Semantic Models and Resource Discovery Mechanism
Deliverable number	D2.3
Deliverable version	1.0
Contractual date of delivery	31 October 2025
Actual date of delivery	4 November 2025
Nature of deliverable	Report
Dissemination level	Public
Work Package	WP2
Deliverable lead	SEERC
Author(s)	Simeon Veloudis

Abstract	This deliverable presents the refined semantic foundations and resource discovery mechanisms of NebulOuS. Building on D2.2, it consolidates three complementary semantic models describing (i) the capabilities of cloud–edge resources, (ii) the QoS requirements of application components, and (iii) the meta-quality constraints governing service delivery. It also presents the enhanced Resource Discovery Mechanism, which has been extended to support edge device registration with the Scheduling Abstraction Layer (SAL) and dynamic integration into deployment scenarios.
Keywords	Fog computing, Cloud Continuum, application deployment, resource discovery, declarative description, semantic models, brokerage, QoS, quality assurance

DISCLAIMER

COPYRIGHT

CONTRIBUTORS

Name	Organization
Simeon Veloudis	SEERC
Andreas Tsagkaropoulos	ICCS

PEER REVIEWERS

Name	Organization
Dimitris Apostolou	ICCS
Paweł Skrzypek	7Bulls

REVISION HISTORY

Version	Date	Owner	Author(s)	Comments
1.0	06/10/2025	SEERC	Simeon Veloudis	Preliminary draft
1.1	09/10/2025	SEERC	Simeon Veloudis	Preliminary draft
1.2	09/10/2025	ICCS	Andreas Tsagkaropoulos	Preliminary draft
1.3	17/10/2025	SEERC	Simeon Veloudis	Preliminary draft
1.4	29/10/2015	SEERC	Simeon Veloudis	Preliminary draft
1.5	03/11/2015	ICCS	Andreas Tsagkaropoulos	Final draft

Abbreviation/Acronym	Open form
ABox	Assertional Box
AMQP	Advanced Message Queuing Protocol
CC	Cloud Continuum
CoCoOn	Cloud Computing Ontology
OAM	Open Application Model
ODRL	Open Digital Rights Language
OWL	Web Ontology Language
Q-SLA	Quality-based Service Level Agreement
SAL	Scheduling Abstraction Layer
SL	Service Level
SLO	Service Level Objective
DL	Description Logic
TBox	Terminological Box
QoS	Quality of Service

TABLE OF ABBREVIATIONS AND ACRONYMSTABLE OF CONTENTS

EXECUTIVE SUMMARY 6
1 INTRODUCTION 7
2 RESOURCE DISCOVERY MECHANISM 8
3 SEMANTIC MODELLING 11
3.1 Capabilities of cloud-edge resources 11
3.2 requirements of application components 11
3.2.1 Service Levels and Service Level Objectives 11
3.2.2 QoS Metrics and Literals 12
3.2.3 Settlement 13
3.3 Meta-quality constraints 14
3.3.1 ODRL 14
3.3.2 Meta-Quality Constraints in ODRL 14
4 CONCLUSIONS 16
5 REFERENCES 17

LIST OF FIGURES

Figure 1: Sequence diagram 8
Figure 2: Updated registration form 9
Figure 3: Updated registration form 9
Figure 4: Overview of Q-SLA 12
Figure 5: SLA example 13
Figure 6: ODRL 14
Figure 7: Meta-quality constraints in ODRL 15

EXECUTIVE SUMMARY

This deliverable presents the consolidated outcomes of Work Package 2 of the NebulOuS project, focusing on the evolution and operational maturity of the semantic and discovery components that underpin intelligent service brokerage across the Cloud–Edge Continuum CEC.

Building on the foundations established in Deliverable 2.2, this iteration introduces:

- **An enhanced Resource Discovery Mechanism**, enabling the registration, management, and utilisation of heterogeneous edge devices through the Scheduling Abstraction Layer (SAL). Key improvements include the integration of device metadata (e.g., geolocation, provider), support for non-standard communication ports, a nonce-based security token for application-specific device registration, and the inclusion of a MongoDB sidecar for persistent device management within Kubernetes.
- **A refined suite of NebulOuS semantic models**, encompassing three complementary ontological representations:
    1. **The CoCoOn-based resource capability model**, semantically lifting operational resource descriptions from SAL.
    2. **The Q-SLA model**, capturing application QoS requirements and enabling ontology-driven reasoning over Service Level Agreements.
    3. **The ODRL-based meta-quality constraint model**, defining higher-level governance rules that constrain SLA formulation and enforce compliance with overarching quality policies.

  These models collectively support automated reasoning for SLA validation, forming the semantic foundation for the NebulOuS Brokerage Quality Assurance mechanism, described in Deliverable 3.2, for enabling formal verification of SLA consistency and compliance with meta-quality constraints.

1 INTRODUCTION

NebulOuS approaches resource discovery within the **Cloud Continuum** (**CC**) through **model-driven** and **semantics-based techniques** that enable the identification of **optimal resources** for deploying hyper-distributed applications. Building on the foundations laid in Deliverable 2.2, *"Initial Semantic Models and Resource Discovery Mechanism"*, this deliverable presents the **matured NebulOuS semantic models** and the **refined resource discovery mechanism** developed to support intelligent, user-centric, and quality-assured application deployment across heterogeneous cloud and edge environments.

NebulOuS enables users to **declaratively** define hyper-distributed applications and their deployment requirements through the Open Application Model (OAM) [1] and its Kubernetes-native implementation, KubeVela, as detailed in Deliverable 2.2. The **Quality of Service** (QoS) requirements of each application component are specified using a NebulOuS-specific **metric model** inspired by CAMEL, which allows metrics to be defined over arbitrary, user-defined QoS attributes and their associated data sources (see Deliverable 2.2). Furthermore, **optimisation objectives governing application placement and execution** are expressed through an AMPL-based optimisation model, also introduced in Deliverable 2.2, which determines deployment configurations that best satisfy both QoS constraints and user preferences.

This model-driven foundation is complemented by an **ontological layer** that formally captures the knowledge embedded in **Service Level Agreements** (**SLAs**). This layer underpins NebulOuS' **Brokerage Quality Assurance mechanism**, reasoning about the consistency and feasibility of user-defined QoS requirements, as well as their compliance with higher-level application policies—the so-called *meta-quality constraints*. This mechanism is detailed in Deliverable 3.2, *"NebulOuS Brokerage, Optimization, and Resource Management"*.

In this context, the present deliverable focuses on two core elements:

- **Section 2** introduces the evolved **Resource Discovery Mechanism**, describing the functional and architectural improvements that enable the seamless integration of heterogeneous edge devices into the NebulOuS Scheduling Abstraction Layer (SAL).

- **Section 3** presents the **NebulOuS semantic models**, encompassing (i) the **capabilities of cloud–edge resources**, (ii) the **QoS requirements** of application components, and (iii) the representation of **meta-quality constraints** governing SLA formulation and compliance, formally expressed through the Open Digital Rights Language (ODRL) [3].

Together, these models and mechanisms form the semantic backbone of NebulOuS, supporting policy-driven SLA governance, and automated reasoning across the Cloud–Edge Continuum.

It should be noted that sections related to the Open Application Model (OAM), its Kubernetes-native implementation KubeVela [4], the NebulOuS metric model, and the optimisation mechanisms are **omitted in this iteration, as no additional work has been performed beyond what was already reported in Deliverable 2.2.** The present document therefore focuses exclusively on the components that have undergone substantial technical and conceptual evolution—namely, the Resource Discovery Mechanism and the NebulOuS semantic models.

2 RESOURCE DISCOVERY MECHANISM

The **NebulOuS Resource Discovery Mechanism** plays a crucial role in enabling the distribution of processing tasks across edge devices and independent computing resources throughout the continuum. Through this mechanism, NebulOuS is able to register such devices with the **Scheduling Abstraction Layer (SAL)**, thereby allowing them to supplement—or even replace—traditional cloud resources in deployment scenarios.

The primary goal of the component is to allow NebulOuS users to easily register computing resources – edge devices or isolated processing nodes – to be used in future deployments of their applications. Registration to the Resource Discovery Mechanism is being accomplished by first completing a form on device characteristics and then combining this information with appropriate diagnostics to provide a wider view of its processing capacity. Once a computing resource is registered to the Resource Discovery Mechanism, registration details are being used to inform the Scheduling Abstraction Layer about the new resource – which can then be used in new deployments by NebulOuS.

In this second iteration of the Resource Discovery Mechanism, a number of **functional and structural improvements** have been introduced, alongside the resolution of several issues identified in earlier releases. The latest version of the component is publicly available at: https://github.com/eu-nebulous/resource-manager

The most significant enhancement concerns the **registration of edge device information** to SAL. This functionality has been coupled with supporting utilities that allow this information to be **published via the AMQP broker** employed by NebulOuS. As a result, NebulOuS can now seamlessly integrate edge devices—or any independent computing nodes not tied to a particular cloud provider—into its deployments. This enables operational scenarios where such resources can be dynamically leveraged to improve flexibility and efficiency.

Another important improvement introduces the ability to **restrict the use of specific edge devices to a single NebulOuS application**. This feature ensures that resources registered for one application are not automatically visible to others (while still retaining the option to register a device for use by all applications, if desired). The restriction mechanism is implemented through a **nonce-based security token** generated by the user interface and transmitted via the broker to the Resource Discovery server. Upon receiving the token, the server validates it through the NebulOuS Middleware, retrieves the associated application name, and constructs a **unique device identifier** comprising both the device and application names. This identifier is stored in SAL and used during resource optimization to filter out devices that do not belong to the current application context.

Figure 1 illustrates the sequence of interactions between an external actor and the Resource Discovery Mechanism during device registration.

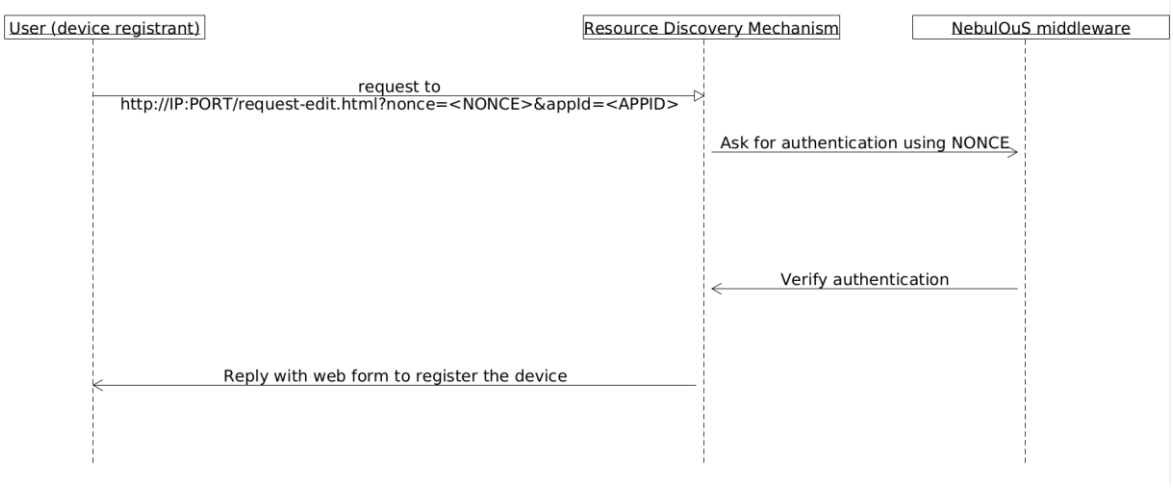


Figure 1: Sequence diagram

Depending on the login method, the updated registration form appears pre-filled with different data in the 'Device Ref' field used by SAL (see Figures 2 and 3). Both registration forms adhere to the following **naming convention** when populating the *Device Ref* field:

$$application\_id| < APP\_ID > | < DEV\_HASH >$$

In this convention, $< DEV\_HASH >$ represents a randomly generated device identifier, while $< APP\_ID >$ corresponds either to an alphanumeric string (for devices registered to a specific application) or to the literal string "all_applications" (for globally available devices).

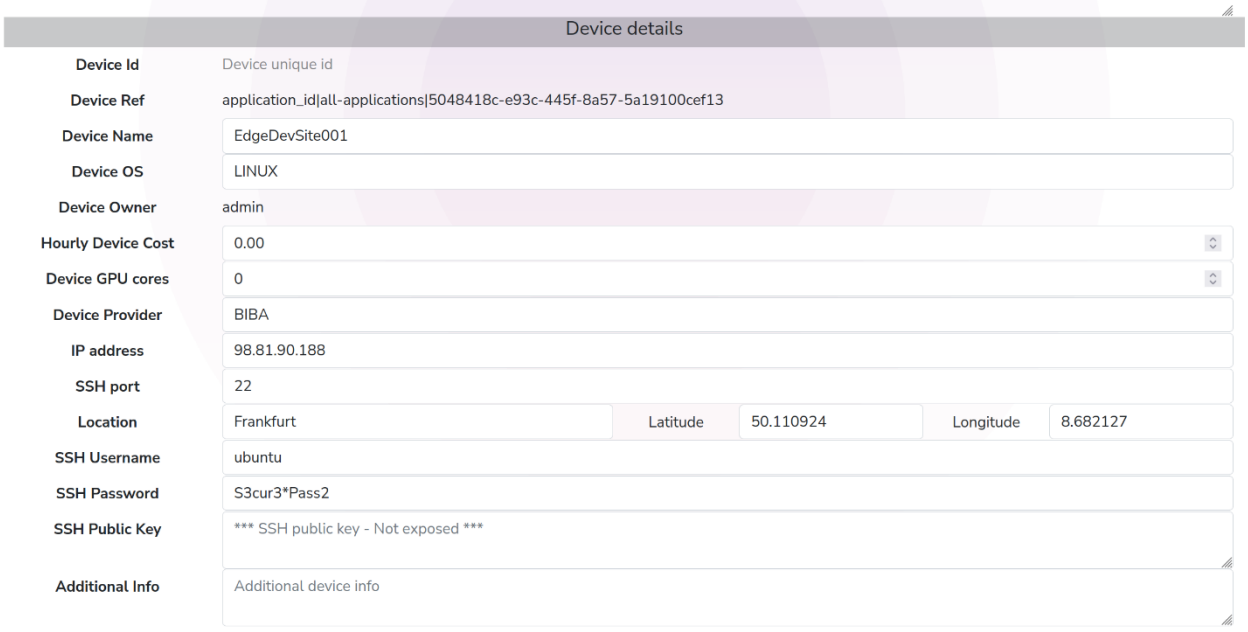


Figure 2: Updated registration form

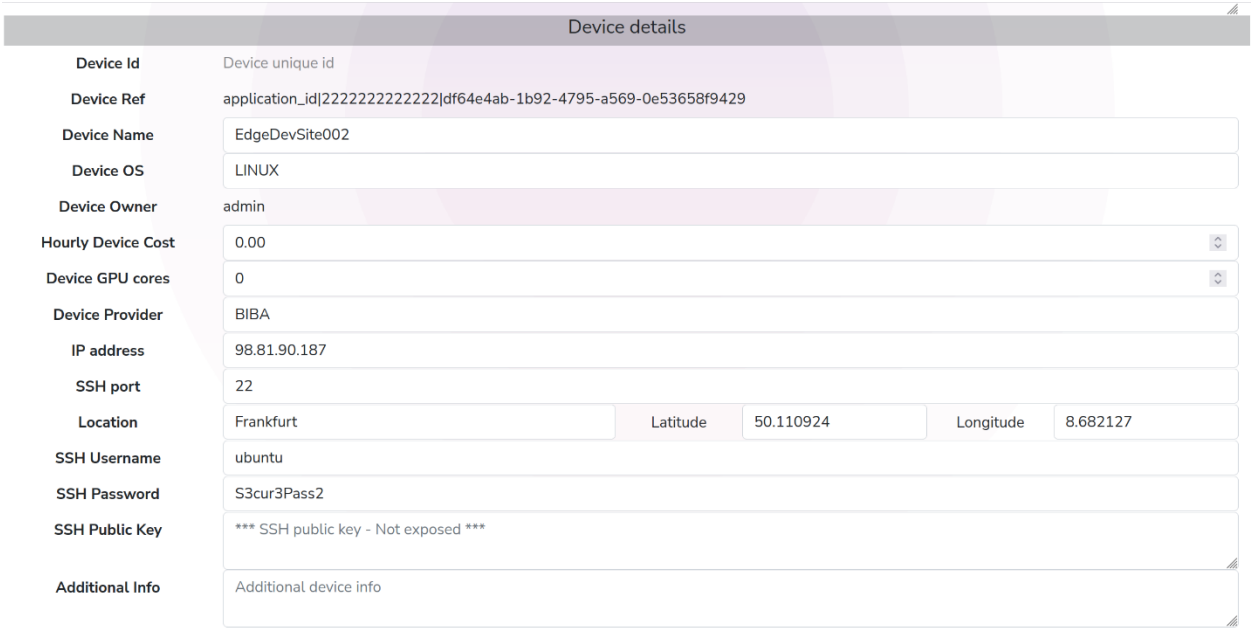


Figure 3: Updated registration form

As an example, let us consider the registration form presented in Figure 4: An edge device belonging to provider 'BIBA', having a public IP address of 98.81.90.187, situated in Frankfurt, is being made accessible through the standard SSH port (22) with the username-password pair illustrated in the figure. The device does not have a GPU, and its cost is zero since it is assumed to be an edge device belonging to the company.

In the 'Device Ref' field, we can observe the middle part of the string (2222222222222) which reflects the identifier of the particular application that the device will only be available for.

The Resource Discovery Mechanism has also been extended to support **device registration using custom port numbers**, enabling communication through non-standard ports and the circumvention of restrictive firewall configurations. In addition, users can now specify further metadata through the user interface, including **device provider, geographic longitude, and latitude**.

Finally, the component's **codebase and deployment structure** have been refined to favour **configuration over hard-coded options**, allowing parameters such as platform IP, username, and password to be externally configured. Structurally, the mechanism now operates as a **Kubernetes pod** with an integrated **MongoDB sidecar**, simplifying the management and persistence of registered device information.

3 SEMANTIC MODELLING

NebulOuS relies on **three distinct semantic models**. The first captures the **capabilities of CC resources**, based on the **CoCoOn information schema** [5], describing compute, storage, network, and other infrastructural characteristics. The second model articulates the **QoS requirements of application components**, leveraging the **Q-SLA ontology** [6] to represent performance, reliability, monitoring, and other service-level attributes. The third model captures **meta-quality constraints**, expressing higher-order policies and rules that govern application deployments, and is based on the **Open Digital Rights Language** (**ODRL**) [3].

3.1 CAPABILITIES OF CLOUD-EDGE RESOURCES

In NebulOuS, resource capabilities are modelled following the approach outlined in Deliverable 2.2: they are represented in terms of the same concepts and properties as IaaS cloud services using the **CoCoOn ontology**.

Building on the initial CoCoOn-based model introduced in Deliverable 2.2, subsequent work has focused on integrating operational resource descriptions through the **Scheduling Abstraction Layer** (**SAL**) and establishing a semantic mapping between SAL and CoCoOn. To operationalise and maintain an up-to-date view of available resources, NebulOuS leverages SAL, which provides a REST API and associated schemas for describing and managing CC nodes and services. SAL defines a set of resource descriptors—including CPU, memory, storage, network bandwidth, location, and price—that capture the capabilities of heterogeneous cloud and edge nodes in a machine-actionable form.

A **mapping is established from SAL to CoCoOn** to semantically lift these operational descriptions into the ontology. In this process, only SAL concepts and attributes that may participate as **first arguments of provider-facing SLA SLOs** are mapped, namely: *compute*, *memory*, *storage*, *network capacity*, *location*, and *price*. Each resource instance described through SAL is translated into a corresponding CoCoOn instance (e.g., cocoon:ComputeService, cocoon:StorageService, cocoon:NetworkService) with its quantitative attributes preserved. Attributes that are operational, temporal, or procedural in SAL (e.g., resource status, scheduling policies, runtime metrics) are excluded from this mapping, as they do not directly contribute to SLO definitions.

This mapping provides the following benefits. First, it establishes a **semantic bridge** between operational resource descriptions and formal resource models, enabling **interoperability** across heterogeneous CC resources. Second, it allows SLA SLOs to be **grounded in CoCoOn**, meaning that each SLO's first argument directly references a semantically defined resource, which supports **automated reasoning** about feasibility, consistency, and policy compliance. In this way, the semantic mapping of SAL to CoCoOn forms a foundational step in ensuring that NebulOuS's **provider-facing SLAs** are **trustworthy**, **precise**, and semantically consistent.

3.2 REQUIREMENTS OF APPLICATION COMPONENTS

The second NebulOuS semantic model focuses on capturing the **Quality of Service (QoS)** requirements associated with the components of hyper-distributed applications. This model builds upon **Q-SLA**, a comprehensive semantic QoS description language developed on top of **OWL**, which provides the formal expressiveness required to represent and reason about **Service Level Objectives (SLOs)** within the CC. The following section outlines the role and usage of Q-SLA within NebulOuS.

3.2.1 Service Levels and Service Level Objectives

Building upon the semantic foundations for representing SLAs established in Deliverable 2.2, the present work extends this model towards **operationalisation and reasoning-driven automation**. While Deliverable 2.2 focused on defining the conceptual structure of the usage of Q-SLA within NebulOuS—including the definition of Service Levels (SLs), SLOs, compensations, and settlements—this section concentrates on how these concepts are now **instantiated, interconnected, and dynamically used in reasoning processes**. It introduces mechanisms for **automatic SLA transitions and settlements**, enabling policy-driven brokerage across the Cloud–Edge Continuum.

In NebulOuS SLAs are treated as aggregations of smaller-scale QoS specifications called **Service Levels (SLs)**. SLs are used for modelling the fluctuating level of resources that is often provisioned in near-edge environments affecting the quality of workload consumption. They may also be used for defining compensating actions that transition between different SLs when certain performance-related threshold values are superseded or violated, thus implementing service degradation schemes.

Ontologically, SLs take the form of **complex QoS constraints** i.e., of logical combinations of simple and/or other complex constraints. **Simple constraints** are boolean expressions that represent SLOs. They invariably comprise two arguments and an operator for comparing them. The first argument (specified through the object property `firstArgument` – see Figure 4) is a **QoS metric**, and the second argument (specified through the data property `secondArgument`) is a **literal**; operators are attached to the constraints through the object property `operator`. In the example SLA shown in Figure 5, two SLs are defined: a **high** and a **low**. Each SL is expressed as a conjunction of two SLOs – **response time** and **availability**. Specifically, the high SL combines the high-tier versions of these SLOs (`ResponseTime_H` and `Availability_H`), while the low SL combines their low-tier counterparts (`ResponseTime_L` and `Availability_L`). The response time SLOs are linked to the average response time metric (`Avg_RT`) via the object property `firstArgument` which they constrain through the data property `secondArgument` (`100ms` for the high SL and `200ms` for the low SL). Similarly, the availability SLOs constrain the average workload uptime metric, setting a `99.999%` threshold for the high SL and `99.99%` for the low SL. Metrics and literals are further elaborated below.

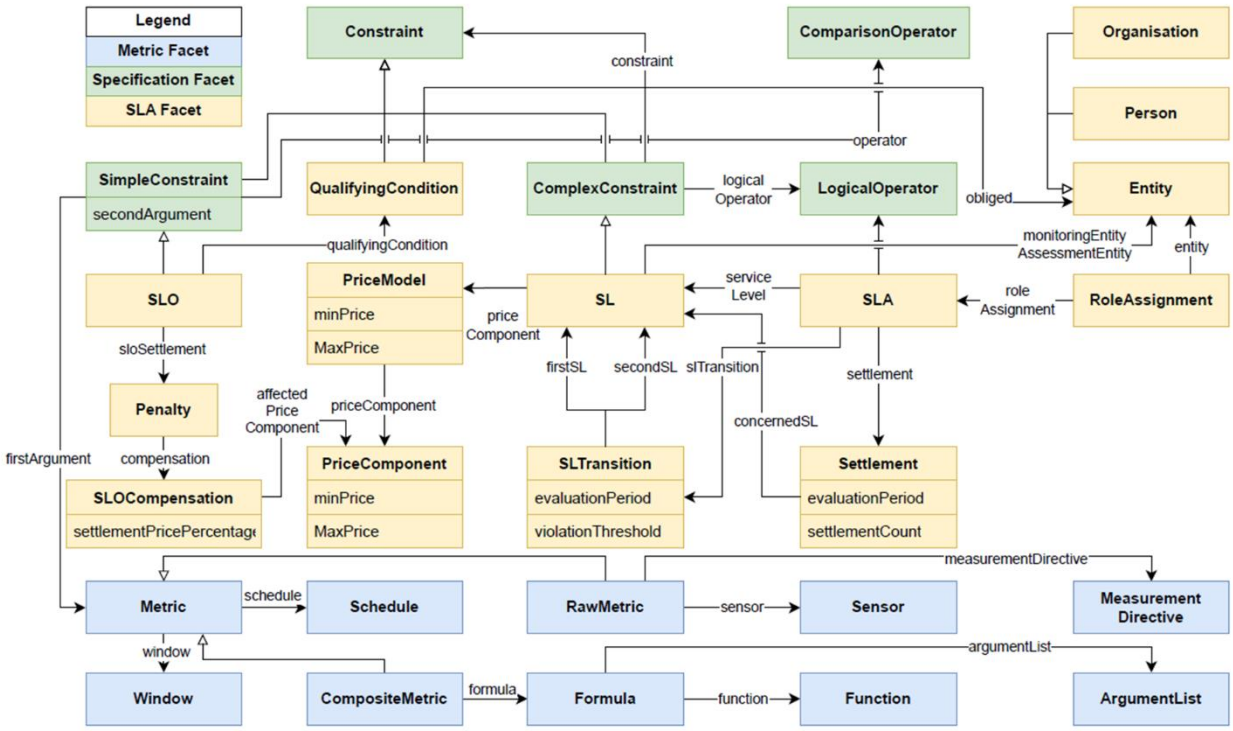


Figure 5: Overview of Q-SLA

3.2.2 QoS Metrics and Literals

QoS metrics in NebulOuS are represented using the **metric model** (or facet – see Figure 4) and may be either **simple** or **composite**. Simple metrics (e.g., "raw response time") are directly monitored from sensors and are associated with **measurement directives** specifying how monitoring should be performed, including scheduling parameters (start/end times and measurement frequency), selection of measurement values (e.g., all measurements after a certain timestamp or within a recent time window), access type ("push" or "pull"), and the scripts used, if any. Each measurement is further annotated with a value type (e.g., integer) and a unit of measurement (e.g., seconds). Users can also define custom simple metrics, provided that monitoring endpoints are available.

Composite metrics (e.g., "average response time") are derived from one or more simple or other composite metrics, enabling the creation of higher-order metrics that bridge the gap between low-level device

measurements and application-level QoS concepts. For example, a workload may be associated with a composite metric "workload availability," represented as a vector combining simple metrics for "uptime" and "response time." In highly dynamic near-edge environments, additional simple metrics—such as "error rate"—can be incorporated into the composition to capture variability and potential service disruptions.

This operational focus extends the original Q-SLA model from Deliverable 2.2 by providing concrete guidance for automated monitoring, dynamic metric composition, and handling of near-edge deployment variability, supporting reasoning-driven SLA management in the Cloud–Edge Continuum.

3.2.3 Settlement

SLA **settlement** refers to the set of automated corrective actions and compensations applied when Service Level Objectives (SLOs) are violated. It formalises how SLA violations are handled, including penalties, service level transitions, or workload termination, and is defined ontologically to support reasoning-driven enforcement across the Cloud–Edge Continuum. Settlement in NebulOuS occurs at three different levels:

1. **SLO level**: Individual SLO violations trigger penalties that take the form of instances of the `Penalty` class (see Figure 4). These instances are linked to instances of the `SLOCompensation` class which specify —via the data property `settlementPricePercentage`— a discount percentage on the cost of workload consumption. In Figure 5, for example, a response time SLO is associated with a penalty that enforces a 5% price discount per SLO violation. Notably, penalties apply regardless of the SL at which the violation occurs.
2. **SL level**: Cumulative SLO violations surpass a preset threshold, prompting transitions between service levels. Transitions are modelled through the `SLTransition` concept (see Figure 4). Each transition connects two SLs using the object properties `firstSL` and `secondSL`, while the data properties evaluationPeriod and `violationThreshold` define its activation conditions. For example, in Figure 5, the `H_L_Transition` instance specifies a downgrade from the high to the low SL when the number of SLO violations exceeds 4 within any 1-hour period.

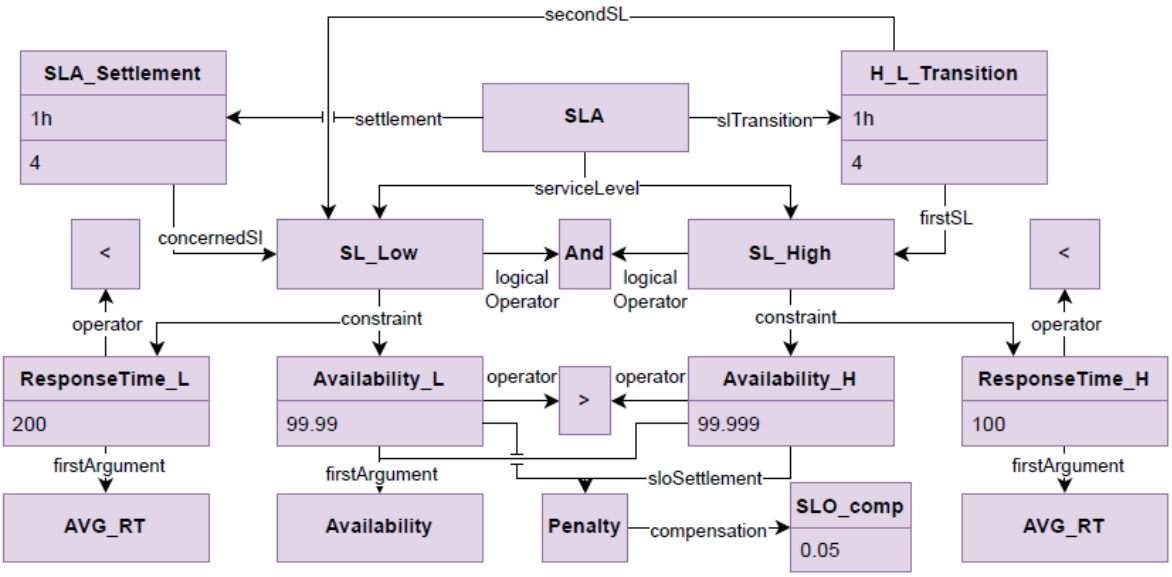


Figure 6: SLA example

3. **SLA level**: Critical situations occur and workload consumption must be terminated. Critical situations are captured through the Settlement concept, using the data properties `evaluationPeriod` and `settlementCount`. In Figure 5, the `SLA_Settlement` instance mandates termination if 4 or more SLO violations occur within any 1-hour period. Notably, this condition applies only when the workload is already operating at the low SL, which is indicated by the association between `SLA_Settlement` and the low SL via the object property `concernedSLs`. Settlement actions may be automatically triggered through ontology-based reasoning. Depending on the occasion, this reasoning may be simpler —e.g., entailing a mere evaluation of the conditions governing the activation of a transition— or more complex —e.g., entailing semantic inferencing over an external geospatial ontology to determine whether the current deployment location is near-edge and hence whether settlement actions should ensue.

3.3 META-QUALITY CONSTRAINTS

Meta-quality constraints impose **governance rules** that constrain how SLAs are formulated. More specifically, they define **higher-level compliance conditions** that govern individual SLOs, or entire SLAs, ensuring that service brokerage decisions adhere to **overarching quality principles**. They may restrict the values of **individual SLOs**—for example, "*the response time SLO should invariably impose a threshold below 10ms*"—or they may apply more broadly, focusing on **workload consumption as a whole** e.g., "*if any SLO is violated more than n times in an hour, workload consumption must be downgraded to a lower QoS (and thus cheaper) level, and if no such level exists, it must be terminated altogether*". In NebulOuS, meta-quality constraints are formally expressed using the **Open Digital Rights Language** (**ODRL**) [3].

3.3.1 ODRL

ODRL is a W3C recommendation designed for **ontologically** representing policies. It provides several normative subtypes for modelling both **policies** and the **rules** that they comprise. Policies can be classified as **Offers**, **Agreements**, or **Sets**, representing proposals, commitments, claims that do not grant any permissions/prohibitions, or collections of rules with no additional semantics, respectively. Similarly, rules can be **Prohibitions**, **Permissions**, or **Duties**, indicating actions that a party cannot, can, or must perform, respectively. A rule is semantically associated with: an **action** i.e., the operation to be performed on the resource; an **asset** or **asset collection** i.e., the resource or entity whose usage the rule governs; a **party** i.e., the entities involved in the rule, such as the **actor** of the action; one or more **constraints** (optional) i.e., additional conditions that further restrict the rule's execution. Actions, assets, and parties can be further **refined** through the `refinement` object property, which links these elements to one or more constraints that specify additional conditions governing their applicability. **Refinements** enable **context-aware** policy expressions, allowing constraints to be applied dynamically based on external factors. For instance, a meta-quality constraint may restrict workload response times to under `100ms`, but with a **temporal refinement** restricting the application of this constraint to certain day times or to certain deployment locations.

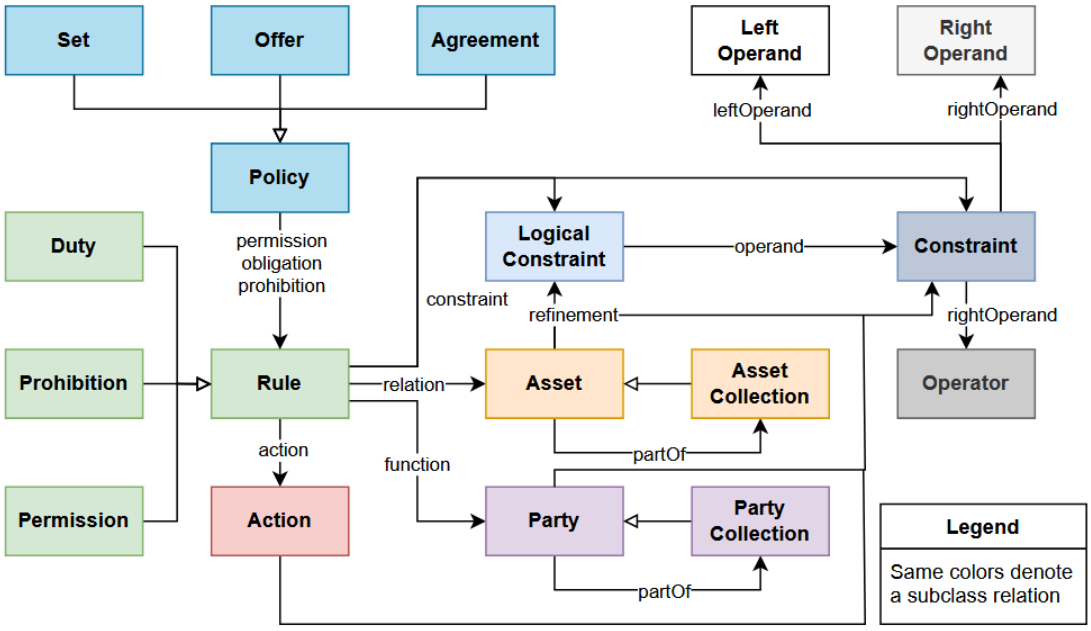


Figure 7: ODRL

3.3.2 Meta-Quality Constraints in ODRL

A meta-quality constraint is modelled as an ODRL **Duty** say *D*. *D* takes the form of an OWL class that satisfies the following axioms.

1. It is linked via the object property target to exactly one `Asset`, which is invariably the SLA that the meta-quality constraint constrains.

2. It is associated via the object property constraint with exactly one `Constraint` or `LogicalConstraint`, which articulates the requirements imposed by the meta-quality constraint.
3. It is linked via the object property function to exactly one `Party`. This property is specialized into either the assigner sub-property, denoting the entity issuing the meta-quality constraint, or the assignee sub-property, denoting the entity subject to the constraint, typically the workload consumer.
4. It is connected via the object property action to exactly one permit `Action`.

These axioms may be formally expressed as terminological (TBox) and assertional (ABox) axioms in the $\mathcal{SROIQ}$ Description Logic (DL) as follows:

$$D \sqsubseteq (= 1\ target.SLA) \sqcap \big(= 1\ constraint.(Constraint \sqcup LogicalConstraint)\big) \sqcap (= 1\ assigner.Party) \sqcap (= 1\ assignee.Party) \sqcap (= 1\ acion.\{permit\})$$

For any object property P and concept C, $(\leq 1P.C)$ represents the abstract class that comprises all those individuals that have at least one association through P with an instance of C; similarly, $(\geq 1P.C)$ represents the class that comprises all those individuals that have at most one association through P with an instance of C. The symbols $\sqcup$ and $\sqcap$ represent, respectively, class union and intersection. $(=1P.C)$ is an abbreviation for $(\leq 1P.C)$ $\sqcap$ $(\geq 1P.C)$. Notably an SLA may be linked to several meta-quality constraints. Intuitively, an SLA is 'permissible' if the logical condition of each of its linked meta-quality constraints is true. All meta-quality constraints linked to a particular SLA should be encapsulated within a single overarching Set policy corresponding to the SLA. Below we provide two example meta-quality constraints. The first binds an individual SLO. It states that the response time SLO should invariably impose a threshold below 10ms. Formally, in SROIQ:

$$D_1 \sqsubseteq constraint.(leftOperand.\{ResponseTime\} \sqcap operator.\{lt, leq\} \sqcap rightOperand \leq 10)$$

The second binds the entire SLA. It states that if any SLO is violated more than 4 times in an hour, workload consumption must be downgraded to a lower QoS level, or (if no such level exists) it must be terminated altogether. Formally, in SROIQ (see also Figure 7):

$$D_2 \sqsubseteq constraint.(evaluationPeriod = 1h \sqcap (violationThreshold \leq 4 \sqcup settlementCount \leq 4))$$

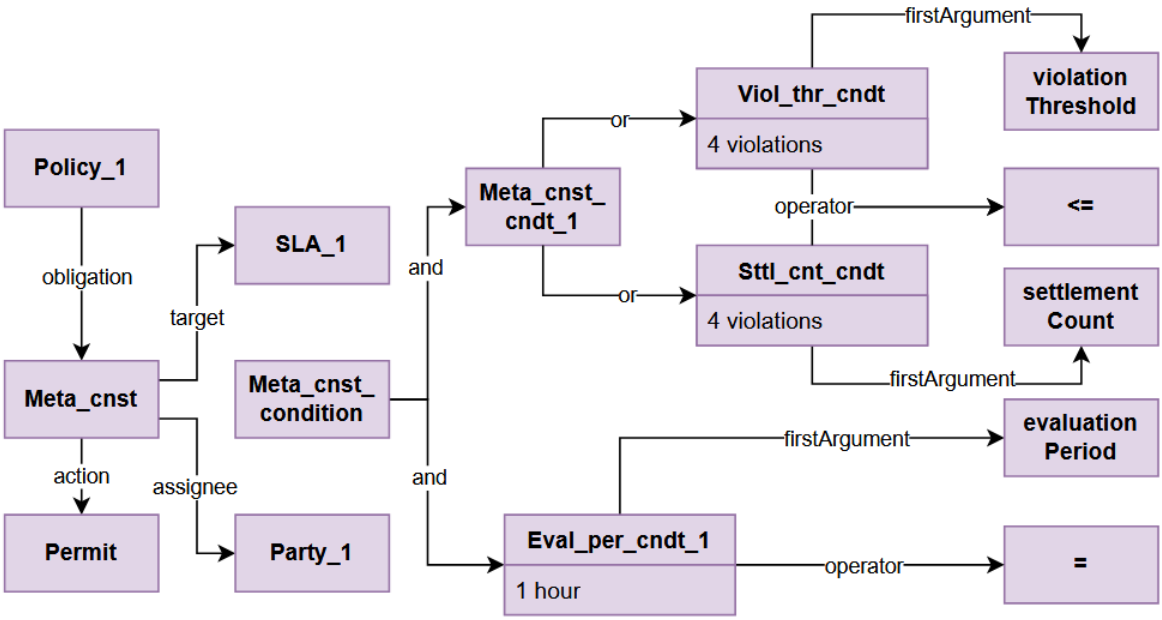


Figure 8: Meta-quality constraints in ODRL

4 CONCLUSIONS

This deliverable presented the matured NebulOuS Resource Discovery Mechanism and the semantic models that collectively enable automated, quality-assured deployment across the Cloud–Edge Continuum.

The Resource Discovery Mechanism has evolved into a robust, Kubernetes-native component capable of dynamically integrating heterogeneous edge resources, while ensuring security, configurability, and operational persistence.

The three semantic models—capturing resource capabilities, application requirements, and meta-quality constraints—provide the ontological foundation for NebulOuS's reasoning-driven QoS assurances.

Together, these advancements mark a significant step toward realising the NebulOuS vision of intelligent, adaptive, and trustworthy service brokerage in the Cloud–Edge Continuum.

5 REFERENCES

[1] Open Application Model (OAM). "An open-model for defining cloud native apps." *OAM Specification*, oam-dev/spec repository, GitHub, v0.3.0 (June 21 2021). Available at: https://github.com/oam-dev/spec

[2] Achilleos, A. P., Kritikos, K., Rossini, A., Kapitsaki, G., Domaschka, J., Orzechowski, M., ... Papadopoulos, G. A. 2019. *The cloud application modelling and execution language (CAMEL)*. *Journal of Cloud Computing: Advances, Systems and Applications* 8, 20 (Dec. 2019). DOI:https://doi.org/10.1186/s13677-019-0138-7

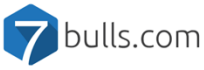
[3] W3C (2018). *ODRL Information Model 2.2*. W3C Recommendation, World Wide Web Consortium (W3C). Available at: https://www.w3.org/TR/odrl-model/

[4] KubeVela Authors. (2025). *KubeVela: The modern application delivery platform (v1.10.4)*. GitHub repository. https://github.com/oam-dev/kubevela

[5] Zhang, Q., Haller, A., & Wang, Q. (2019). *CoCoOn: Cloud Computing Ontology for IaaS Price and Performance Comparison.* CECS, The Australian National University. Available at: https://openresearch-repository.anu.edu.au/items/c690c5e3-9c1c-44d4-9aa2-97ccacc6bc68

[6] Kritikos, K., & Plexousakis, D. (2016). *Semantic SLAs for Services with Q-SLA*. In Proc. of the 9th International Conference on Cloud Computing and Services Science (CLOSER 2016). Available at: https://www.researchgate.net/publication/309196705_Semantic_SLAs_for_Services_with_Q-SLA

CONSORTIUM



NebulOuS

A META OPERATING SYSTEM FOR BROKERING HYPER-DISTRIBUTED APPLICATIONS ON CLOUD COMPUTING CONTINUUMS