Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

ODRL Operational Semantics to Rule Them All

Wout Slabbinck

wout.slabbinck@ugent.be

Ghent University

Wouter Termont

Ghent University

Joachim Van Herwegen

Ghent University

Beatriz Esteves

Ghent University

Research Article

Keywords:

Posted Date: November 12th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11010530/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

International Journal of Information Security manuscript No.
(will be inserted by the editor)

ODRL Operational Semantics to Rule Them All

Wout Slabbinck[a,1], **Wouter Termont**[b,1], **Joachim Van Herwegen**[c,1], **Beatriz Esteves**[d,1]

[1] IDLab, Department of Electronics and Information Systems, Ghent University – imec, Technologiepark-Zwijnaarde 126, 9052 Ghent, Belgium

Received: date / Accepted: date

Abstract Organisations increasingly connect AI agents to enterprise systems to enable cross-domain reasoning. However, granting agents access required for such tasks also exposes data holders to irreversible harm, as a misconfigured or poorly aligned agent may modify or delete information across multiple systems before (human) intervention becomes possible. Fine-grained guarantees over permissible data use are therefore essential. In this context, usage control policies provide fine-grained guarantees over data use and the Open Digital Rights Language (ODRL) has emerged as a standard language for expressing them. However, ODRL only standardises how policies are expressed. Without a common evaluation model, interoperability ends at policy exchange rather than policy enforcement. To enable interoperable enforcement of ODRL policies, we formalise the operational semantics required for policy evaluation and compliance checking. More specifically, we formalise the determination of whether an action is allowed and whether recorded executions conform to a policy. In this paper, we present *i)* formal definitions of the inputs required for policy evaluation and compliance, *ii)* an operational formal semantics for ODRL evaluation and compliance, including conflict resolution strategies, and *iii)* an initial alignment with interoperable vocabularies endorsed by the ODRL Formal Semantics Community Group. Together, the operational semantics and their initial alignment with interoperable representations provide a basis for consistent usage control decisions across ODRL implementations. This brings ODRL closer to enforceable guarantees that allow data holders to share data with agents while retaining control.

1 Introduction

Inter-organisational data sharing remains challenging despite the availability of mature data exchange technologies. A key challenge is not a lack of technical capability, but a lack of trust between data-sharing parties [1–3]. One contributing factor is the loss of control over data once it has been shared. The copyright infringement litigation against Microsoft and OpenAI concerning the use of books for AI training illustrates how enforcing data usage restrictions becomes difficult once data has been made available [4]. Consequently, increasing data sharing requires not only mechanisms that govern access to data, but also mechanisms that enforce how shared data may be used.

That is where legal and regulatory instruments can contribute. Data providers can specify usage constraints through licences, such as the Creative Commons Attribution (CC-BY)[1] and MIT licences[2]. Similarly, regulations such as the General Data Protection Regulation (GDPR) [5] and the Digital Millennium Copyright Act (DMCA) [6] govern how data may be used.

These legal texts are intended for human interpretation and enforcement. Semantic technologies such as the European Legislation Identifier (ELI)[3] and Akoma Ntoso [7] represent an important step by providing machine-readable representations of legal texts. However, machine readability alone is insufficient. While these technologies enable systems to

[a]ORCID: 0000-0002-3287-7312
e-mail: wout.slabbinck@ugent.be
[b]ORCID: 0000-0002-2968-1394
e-mail: wouter.termont@ugent.be
[c]Orcid: 0000-0002-8215-1827
e-mail: joachim.vanherwegen@ugent.be
[d]ORCID: 0000-0003-0259-7560
e-mail: beatriz.esteves@ugent.be

[1]Creative Commons Attribution (CC-BY) License: https://creativecommons.org/licenses/by/4.0/
[2]MIT License: https://opensource.org/license/mit
[3]European Legislation Identifier: https://op.europa.eu/en/web/eu-vocabularies/eli

identify and exchange legal provisions, they do not capture the executable semantics required to enforce them. Enforcement requires machine-actionable usage control policies that operationalise legal and contractual requirements.

To the best of our knowledge, the Open Digital Rights Language (ODRL) [8, 9] is the most mature standard for expressing interoperable usage control policies. As a W3C Recommendation, ODRL provides a domain-agnostic framework for expressing permissions, prohibitions, and obligations over the use of data. While ODRL standardises policy expression, it does not standardise policy interpretation and evaluation. Deterministic, interoperable policy evaluation and enforcement require that independent policy engines produce identical outcomes when evaluating the same policy under the same circumstances [10]. Achieving this requires both a shared interpretation of ODRL policies and a common representation of the contextual information used during policy evaluation [10, 11]. Furthermore, the lack of behavioural formal semantics in ODRL has long been recognised as a source of ambiguity and divergent interpretations [12–15]. Without a standardised interpretation and evaluation model, different policy engines may reach different conclusions when evaluating the same policy.

To enable interoperable ODRL evaluation, we make three contributions. In this paper, we make three contributions to enable interoperable ODRL evaluation. First, we provide formal definitions of the inputs required for policy evaluation and compliance, including evaluation requests and states of the world. Next, we present an operational formal semantics for ODRL evaluation and compliance based on five core functions: constraint satisfaction, rule evaluation, rule compliance, policy evaluation, and policy compliance. Finally, we demonstrate how these formal semantics can be applied through an initial alignment with interoperable vocabularies endorsed by the ODRL Formal Semantics Community Group[4].

The remainder of this paper is structured as follows. Section 2 reviews the foundations required for interoperable usage control enforcement through the lens of ODRL and motivates the need for interoperable evaluation semantics. Section 3 introduces the ODRL Information Model through a running example and illustrates its main modelling capabilities. Section 4 presents our main contribution: an operational formal semantics for ODRL evaluation with conflict resolution. Section 5 demonstrates how the proposed formalism can be applied in practice and connected to interoperable representations of its inputs and outputs. Section 6 discusses limitations of the approach, compares it with existing ODRL formal semantics. Finally, Section 7 concludes the paper and outlines directions for future work.

[4]ODRL Formal Semantics specification: https://w3c.github.io/odrl/formal-semantics/

2 Related Work

2.1 Usage Control Policy Languages

Traditional access control determines whether a subject is allowed to perform a particular action on a protected resource [16]. Usage control extends access control beyond the initial authorisation decision [17–19]. While access control determines whether access is granted, usage control governs the complete lifecycle of that usage. To this end, usage control models incorporate deontic concepts such as permissions, prohibitions, and obligations, enabling policies to specify which actions are allowed, forbidden, or required [17, 18]. Consequently, enforcement is not limited to the evaluation of an access request but also encompasses the monitoring of resource usage to ensure that policy conditions continue to hold and that obligations are fulfilled [19].

For example, an access control policy can define whether a researcher can download a dataset, whereas a usage control policy can further restrict the permitted purpose of that download and require its deletion after two weeks.

Data sharing increasingly occurs across federated and decentralised environments. For usage control to operate across multiple systems, the meaning of policies must be understood consistently such that data holders retain control over the full data lifecycle. Semantic technologies are particularly suitable for representing usage control policies because shared vocabularies and explicit representations of policy concepts and relationships enable interoperability and flexibility [20].

Within the Semantic Web, several policy languages and frameworks have been proposed. REI [21] is a Semantic Web policy language based on deontic concepts such as permissions, prohibitions, obligations, and dispensations. REI represents policies using RDF and was later extended with OWL to support Semantic Web reasoning [22]. Protune [23] extends trust negotiation with declarative policies, provisional actions, and monitoring mechanisms for distributed systems. The SPECIAL Policy Language (SPL) [24], developed within the SPECIAL project[5], focuses on the specification and compliance verification of personal-data processing policies. The Open Digital Rights Language (ODRL) [8] provides a standardised model for expressing permissions, prohibitions, and obligations over digital resources. Finally, the Generic Usage CONtrol (GUCON) model [25] proposes a graph-based representation for usage control policies and compliance checking, with a corresponding implementation based on RDF- and SPARQL-STAR [26].

Among these approaches, ODRL is particularly relevant because it is the only usage control language that progressed through the W3C standardisation process and became a W3C

[5]SPECIAL project: https://cordis.europa.eu/project/id/731601

Recommendation in 2018. ODRL can be extended through profiles to support domain-specific concepts, such as the Data Privacy Vocabulary (DPV) [27], for which mappings to ODRL have recently been proposed [28]. In addition, ODRL has been adopted by large-scale data-sharing initiatives, including the International Data Spaces Association (IDSA) [29], Gaia-X [30], and the European SIMPL programme [31]. For this reason, the remainder of this paper focuses on ODRL.

2.2 Open Digital Rights Language (ODRL)

2.2.1 The Path to the Open Digital Rights Language

At the end of the 1990s, connecting to the internet became more widespread thanks to among others the advances in broadband technologies and the introduction of the World Wide Web (WWW) [32]. Thus, it became easier to share and consume information digitally. This progress in technology paved the way for other forms of data sharing, more specifically the illegal sharing of copyrighted data, such as songs, pictures, and videos, which was accelerated through easy to use programs, such as Napster and Limewire. Government authorities were quick to react and legislate, leading to the enactment of regulations such as the Digital Millennium Copyright Act (DMCA) [6] in the United States, to extend the reach of copyright law to the digital world. To enable the enforcement of such Acts, the usage of technical standards emerged to represent digital copyright requirements, in an effort to have rules that can be interpreted by software [33, 34].

In 2001, the World Wide Web Consortium (W3C) [35], an organisation that develops standards and guidelines for the WWW, hosted a workshop on Digital Rights Management for the Web[6] [36]. One of the proposals emerging from that workshop was the Open Digital Rights Language (ODRL) by Renato Iannella, a set of specifications to express digital rights in an interoperable way with the goal to start a standardisation process. While the standardization process did not come to fruition, ODRL became a W3C Note[7] in 2002: ODRL 1.1[8], an XML-based information model and vocabulary for expressing permission rights to parties over assets.

In 2011, W3C initiated Community Groups[9] to broaden participation in web technology development beyond the formal Working Group process. Community Groups were open to non-members and provided a space to develop ideas and specifications, some of which could later progress towards standardisation. In this context, the people and organisations working on ODRL established the ODRL Community Group (CG)[10] to maintain existing ODRL specifications and develop future versions, resulting in the publication of ODRL 2.0 in 2012[11]. In this version, prohibitions were introduced to explicitly define that a party cannot perform a certain action. To group rules together, the term policy was added to the vocabulary. A final important change to improve interoperability is the switch from XML-based serialisation to represent policies using the Resource Description Framework (RDF)[37, 38]. RDF provides a graph-based data model in which resources and relationships are identified through globally unique IRIs[12]. This model is independent of its serialisation, allowing the same semantics to be expressed in multiple formats. For ODRL, this provides a shared underlying policy model, enabling policies to be exchanged and interpreted consistently across implementations.

In 2015, the CG finalized another iteration with a notable addition — ODRL profiles[13]. Profiles allow any adopter of ODRL to add domain specific elements, e.g., actions and specific constraint properties not specified by the model, to enhance extensibility of the core model.

Finally, in 2016, a charter was made to start a Working Group to standardise ODRL: the Permissions & Obligations Expression (POE) Working Group. After two years of hard work, this resulted in the W3C Recommendations as we know them today: the ODRL 2.2 Information Model [8] and the ODRL Vocabulary & Expression 2.2 [9].

2.2.2 ODRL Today: A Look at What's Standardized and What Still Isn't

As of today, the ODRL 2.2 W3C Recommendations contain an Information Model (IM) [8], that describes how to use ODRL to express usage control policies, and a Vocabulary [9], which defines all concepts and relations. An ODRL policy groups together a set of rules that describe the usage of an asset, such as digital content, data, software, services, or other resources. Each rule has a deontic type: a permission, a prohibition, or a duty[14]. The core concepts of

[6] Workshop on Digital Rights Management for the Web: https://www.w3.org/2000/12/drm-ws/Overview.html

[7] A W3C Note is a published document intended to record and disseminate work, but it is not part of the W3C standards track and is not endorsed by W3C as a Web standard, unlike a W3C Recommendation; see https://www.w3.org/standards/types/.

[8] ODRL 1.1: https://www.w3.org/TR/odrl/

[9] Introducing W3C Community Groups: https://www.wired.com/2011/08/w3cs-new-community-groups-give-everyone-a-voice-in-html5/

[10] ODRL Community Group: https://www.w3.org/community/odrl/

[11] ODRL 2.0 specification: https://www.w3.org/community/odrl/2-0/

[12] Internationalized Resource Identifiers (IRIs) are a standardised mechanism [39] that provides globally unique and unambiguous identifiers for resources, allowing the same concepts to be referenced consistently across systems.

[13] See the ODRL Profile Best Practices report for guidance on profile design and publication: https://www.w3.org/community/reports/odrl/CG-FINAL-profile-bp-20240808.html.

[14] A permission specifies what is allowed, a prohibition what is forbidden, and a duty what must be fulfilled.

a rule are actions, parties, and assets. Actions specify the operation, parties identify the actors involved, and assets denote the resources to which the rule applies. Together these concepts define *who* is allowed, forbidden, or must perform *which action* on *which asset*. Rules can be further refined through constraints and duties. Constraints restrict the circumstances under which a rule holds, for example by limiting the access in time, purpose, number of executions, etc. Duties specify actions that must be performed in conjunction with the rule, such as making a payment, or deleting data after use. The ODRL model is described in greater detail in Section 3.

Furthermore, a W3C Recommendation requires multiple implementations to demonstrate interoperability[15]. For ODRL 2.2, these implementations are best described as *ODRL Validators* which are defined as *"a system that checks the conformance of ODRL Policy expressions, including the cardinality of properties and if they are related to types of values as defined by the ODRL Information Model, and the Information Model's validation requirements."* [8]. Such implementations therefore verify whether a policy conforms to the ODRL Information Model and Vocabulary. The implementation report lists both the conformance requirements and the implementations used to satisfy them[16]. One such implementation is the open-source **odrlapi**[17].

The Working Group deliberately limited the scope of ODRL to policy representation and validation, leaving policy evaluation and enforcement outside the recommendation's scope [40]. Consequently, independent implementations may reach a different conclusion for the same ODRL policy evaluation. Such behavioural differences can lead usage control systems to enforce identical ODRL policies differently, motivating recent work on ODRL evaluation and formal semantics [10, 12–15]. The following sections discuss the areas that remain unstandardised in ODRL 2.2. Section 2.3 introduces the use of ODRL in usage control enforcement, Section 2.4 discusses ODRL compliance checking, and Section 2.5 reviews related work on ODRL formal semantics.

2.3 Enforcement of ODRL policies

Within the European Data Union Strategy [41], dataspaces have emerged as a key mechanism to facilitate data sharing while allowing data providers to retain control over how their data may be used. Consequently, several dataspace initiatives employ ODRL to express usage control policies. The International Data Spaces Association (IDSA) incorporates ODRL into its Reference Architecture Model[18] and defines an ODRL-based language for expressing usage control policies [29]. Similarly, Gaia-X employs ODRL as the policy language for contract and usage policies in data exchange scenarios [30].

ODRL-based enforcement is not limited to dataspaces. Slabbinck et al. [42] demonstrate how ODRL policies can be enforced over decentralised personal data stores. Another example is O-Prime, which employs ODRL policies for transparent monitoring, provenance tracking, and policy-violation detection in personal data management scenarios [43]. Furthermore, ODRL-based policy evaluation has been integrated into User-Managed Access (UMA) infrastructures to enable policy-driven authorisation decisions in OAuth-based ecosystems [44].

Although enforcement architectures may differ, they all rely on a policy evaluation component that determines whether a requested action complies with the applicable policies and contextual information. However, ODRL 2.2 leaves evaluation behaviour outside the scope of the recommendation. As a result, ODRL evaluators are not necessarily interchangeable, since different implementations may derive different decisions from the same policy. The next subsection examines this compliance-checking problem in greater detail.

2.4 ODRL Compliance Checking

Compliance checking verifies whether a state of the world conforms to specified norms [45]. Several ODRL evaluators have been proposed, each adopting a different interpretation of the standard. ODRL-PAP[19], developed within the DOME Marketplace project[20], relies on domain-specific contextual concepts. The Open Digital Rights Enforcement (ODRE) framework [46] supports policy evaluation by incorporating temporal information through implementation-specific Java code. The MOSAICrOWN policy engine[21] models purpose at the rule level, whereas ODRL defines purpose as a constraint operand. MYDATA Control Technologies [47] maps ODRL policies to a proprietary XML-based policy language, while the Prometheus-X ODRL Manager[22] operates on a fixed JSON structure rather than the RDF-based ODRL information model. Additional evaluators include the

[15] Section 6.3.2 of the W3C Process Document details the need for independent interoperable implementations: https://www.w3.org/policies/process/.

[16] Implementation Report of the Permissions & Obligations Expression Working Group (WG), the WG that standardised ODRL: https://w3c.github.io/poe/test/implementors

[17] Web interface: https://odrlapi.appspot.com/.

[18] IDSA RAM landing page: https://internationaldataspaces.org/offers/reference-architecture/

[19] ODRL-PAP: https://github.com/wistefan/odrl-pap

[20] DOME Marketplace project: https://dome-marketplace.eu/

[21] MOSAICrOWN policy engine: https://github.com/mosaicrown/policy-engine

[22] Prometheus-X ODRL manager: https://github.com/Prometheus-X-association/odrl-manager

Gaia-X Wizard Policy Decision Point (PDP)[23] and Polival[24], which evaluates ODRL policies through SPARQL query rewriting. The ODRL Evaluator by Slabbinck et al. [10] produces fine-grained evaluation reports that explain to what extent an access request is permitted or denied, rather than returning only a binary decision. Finally, Salas et al. [48] introduced the OVAL Evaluator, an implementation of the query-answering semantics proposed in their earlier work. The evaluators discussed above are not exhaustive. An up-to-date overview of policy editors, evaluators, validators, and related ODRL tooling is maintained through the ODRL Landscape[25].

Despite implementing the same standard, these evaluators are not directly interchangeable. Esteves et al. [11] identify a lack of common inputs and outputs across implementations. Kebede et al. [14] further show that ambiguities in the ODRL specification can lead to divergent interpretations of identical policies. Moreover, Cimmino et al. [12] and Termont et al. [49] observe that ODRL lacks a common behavioural semantics for policy evaluation. Consequently, ODRL evaluators frequently embed their own assumptions about context representation and runtime behaviour. This motivates the need for standardised formal semantics that precisely define the interpretation and execution of ODRL policies. The next subsection surveys existing formalisation efforts and their respective approaches to this challenge.

2.5 ODRL Formal Semantics

Formalisation efforts for ODRL began with ODRL 1.1. García et al. [50] sought to make ODRL semantics explicit by representing ODRL concepts using Semantic Web technologies and OWL. A similar objective was pursued by Kasten and Grimm [51]. In parallel, Pucella and Weissman [52] provided a logical foundation for ODRL. They defined a translation from ODRL policies to a fragment of many-sorted first-order logic with equality, allowing policy statements to be interpreted and analysed using formal logical semantics.

For ODRL 2.1, Steyskal and Polleres [53] identified ambiguities caused by dependencies between actions and introduced a formal semantics based on an abstract syntax for ODRL policies which enables rule-based reasoning over ODRL policies.

While the preceding works focused on earlier ODRL versions and influenced the development of the standard, the remainder of this section discusses formalisms for the W3C Recommendation itself. De Vos et al. [13] introduced the ODRL Regulatory Compliance Profile and demonstrated how ODRL policies can be translated into Answer Set Programs (ASP) to perform automated compliance checking. Fornara and Colombetti [54] formalised ODRL using state machines and production rules and identified the absence of deadlines as a limitation of ODRL obligations. Kieffer et al. [55] introduce Complex License Composition (CLiC), an approach for composing licenses with ODRL constraints and refinements. The approach provides a formalisation to compute the least restrictive license.

Bonatti et al. [56] proposed a declarative semantics for ODRL 2.2 covering both policy evaluation, where an access request is assessed against a policy, and compliance checking, where an execution history is analysed to determine whether policy requirements have been respected. Salas et al. [57] introduced a query-answering semantics for ODRL and distinguished between *ODRL Lite*, where the rules do not have duties, and full ODRL. Their work does not include some of the reasoning aspects of ODRL, such as the action hierarchy as they see this as a required pre materialisation step, and do not incorporate ODRL's conflict-resolution mechanism. In subsequent work, Salas et al. [48] demonstrated an implementation of these formal semantics for both access control and monitoring.

Papagiannakopoulou et al. [58] introduced a framework that leverages DPV and ODRL to semantically align policies from third parties. To address conflicts that may arise between aligned policies, the framework incorporates mechanisms for conflict identification and resolution. Mustafa et al. [59] ground ODRL in the legal ontology (UFO-L)[26]. With their formalism, they investigate the legal positions and authority structures that arise once an ODRL policy is in force, including who may declare violations and impose their consequences.

Existing formalisations address either policy reasoning [48, 56] or ontological grounding [59]. However, they do not define mappings between their formal objects and interoperable representations of evaluation requests, contextual information, and evaluation outcomes. Consequently, implementations may share the same formal semantics while relying on different input and output models, limiting interoperability across evaluators.

3 The ODRL Information Model

In this section, we will explain the ODRL 2.2 Information Model [8] through some examples about a movie enthusiast (Bob) and a fictional streaming company (Netplus TV).

Over the last couple of weeks, friends of Bob were constantly praising the particular movie Spiderpig that is exclusive to Netplus TV. Not being able to join those conversations, Bob finally decided to create an account and watch the

[23] Gaia-X Wizard Policy Decision Point (PDP): https://wizard.lab.gaia-x.eu/policyStepper

[24] Polival: https://codeberg.org/elbtech/Polival

[25] ODRL Landscape: https://w3c.github.io/odrl/landscape/

[26] UFO-L is the Unified Foundational Ontology for Legal Domains.

movie. To ensure not just everybody can watch any movie on its platform, Netplus TV controls usage through policies. In Listing 1 a simple ODRL policy is displayed that permits Bob to stream Spiderpig. An ODRL policy itself is a collection-like construct for grouping at least one rule. There are three types of rules in ODRL, following deontic concepts[27]: *i)* a *permission* which allows an action, *ii)* a *prohibition* that forbids an action, and *iii)* a *duty* entailing that an action must be done. Next to its type, a rule has several properties, of which the most notable are `action`, `target`, `assignee`, and `assigner`. The `action` property contains a value from a curated list in the standardised ODRL 2.2 Vocabulary [9]. The `target` property describes the resource, an asset, to which the action is bound. The optional `assignee` and `assigner` properties identify, respectively, the party to whom the rule applies and the party that issues the rule. If no `assignee` is specified, the rule applies to any party.

To ensure that a rule refers to the specific individual Bob rather than any arbitrary Bob, the `assignee` property uses IRIs. This applies to all rule properties, supporting precise identification and interoperability.

```
ex:odrlPolicy a odrl:Set;
  odrl:uid ex:odrlPolicy;
  odrl:permission ex:permissionBobSpiderpig.

ex:permissionBobSpiderpig a odrl:Permission;
  odrl:action odrl:stream;
  odrl:target ex:Spiderpig;
  odrl:assignee ex:Bob;
  odrl:assigner ex:Netplus TV.
```

Listing 1 ODRL policy with a permission rule that states: **permission** for **Bob** to **stream Spiderpig**.

Netplus TV is not a charity organisation and does not just allow users to watch movies for free. Instead, like many other streaming services, it offers a subscription model: customers are required to pay a certain amount of money and in return they are allowed to use the service (watch movies) for a specified amount of time. A policy as shown in Listing 1 does not cover such a business model, though ODRL is expressive enough. Rules can be made more specific through employing *constraints* and attaching *duties*. Moreover, *refinements* provide an additional mechanism for further narrowing actions, assets, and parties. The policy in Listing 2 demonstrates the business case clearly through a constraint to limit the duration and a duty to ensure that Bob pays first. Constraints in ODRL consist of three parts: a left operand, an operator and a right operand. The list of applicable left operands and operators are defined by ODRL. Next to ordered comparison operators (e.g., greater-than or less-than), ODRL also defines set-based comparison ones. The right operand is a value[28] to which the left operand is compared. To denote the subscription fee, a duty rule is added to the permission, implying that the duty must be fulfilled before the permission is granted[29,30]. Since the duty to `compensate` is still too vague as it does not describe the amount and currency owed, a `refinement` is added. As can be seen in the example, a refinement is nothing more than a constraint; not applied on a rule, but on a part of the rule (in this case the action).

```
ex:odrlPolicy a odrl:Set;
  odrl:uid ex:odrlPolicy;
  odrl:permission ex:permissionBobSpiderpig.

ex:permissionBobSpiderpig a odrl:Permission;
  odrl:action odrl:stream;
  odrl:target ex:Spiderpig;
  odrl:assignee ex:Bob;
  odrl:assigner ex:Netplus TV;
  odrl:constraint ex:temporalConstraint;
  odrl:duty ex:paymentObligation.

ex:temporalConstraint a odrl:Constraint;
  odrl:leftOperand odrl:dateTime;
  odrl:operator odrl:lteq;
  odrl:rightOperand "2026-05-31T23:59:59.999Z"^^xsd:dateTime.

ex:paymentObligation a odrl:Duty;
  odrl:action ex:refinedAction;
  odrl:assignee ex:Bob .

ex:refinedAction
  rdf:value odrl:compensate;
  odrl:refinement ex:refinement.

ex:refinement a odrl:Constraint;
  odrl:leftOperand odrl:payAmount;
  odrl:operator odrl:eq;
  odrl:rightOperand 10;
  odrl:unit <http://dbpedia.org/resource/Euro>.
```

Listing 2 ODRL policy with a permission rule which has a constraint and a duty: **permission** for **Bob** to **stream Spiderpig** until the end of the month with a pre-condition that he pays the subscription.

While Netplus TV is a company of respectable size, it does not have the right to distribute An even bigger issue is the fact that for some assets it lacks the rights to distribute them worldwide. In particular for Spiderpig, Netplus TV does not have the right to distribute the movie in Japan, which means that nobody is allowed to stream the movie in

[27] The deontic concepts come from *deontic logic*: a modal logic with operators for permission, obligation, and prohibition [60].

[28] In ODRL a right operand can also be a *rightOperandReference* with as value an URL. Unfortunately, how to dereference the actual value is not in the current standard [15, 61].

[29] Due to the absence of deadline properties for the Duty class in ODRL and in section 2.6.5 the Information Model states that all duties must be fulfilled, we infer that duties must be fulfilled before a permission can be granted.

[30] Duties can also be attached to prohibitions using the `remedy` property, to have a fallback if the action under the prohibition was executed, and to a duty itself using the `consequence` property, which defines an additional requirement to be fulfilled if the original duty is not fulfilled.

Japan. We can represent this as a prohibition rule in ODRL and add it to the policy in Listing 2, though this introduces a conflict: if Bob were to watch this movie in Japan, he would be both allowed and disallowed to do so; at the same time! To resolve conflicts of this matter, policies in ODRL include the `conflict` property for a resolution strategy. In our case, the movie can not be streamed at all costs in Japan and as such the prohibition rule prevails over the permission, necessitating the `prohibit` strategy as shown in Listing 3.

```
ex:odrlPolicy a odrl:Set;
  odrl:conflict odrl:prohibit ;
  odrl:uid ex:odrlPolicy;
  odrl:prohibition ex:prohibitionBobSpiderpig;
  odrl:permission ex:permissionBobSpiderpig.

ex:prohibitionBobSpiderpig a odrl:Prohibition;
  odrl:action odrl:stream;
  odrl:target ex:Spiderpig;
  odrl:assignee ex:Bob;
  odrl:assigner ex:Netplus TV;
  odrl:constraint ex:spatialConstraint.

ex:spatialConstraint a odrl:Constraint;
  odrl:leftOperand odrl:spatial;
  odrl:operator odrl:eq;
  odrl:rightOperand <https://w3id.org/dpv/loc#JP>.
```

Listing 3 ODRL policy containing a permission and a prohibition over the same action and asset with the `odrl:prohibit` conflict strategy: permission for **Bob** to **stream Spiderpig**, while simultaneously prohibiting the same action in Japan; the prohibition prevails when both rules apply.

Expressing policies at the level of individual assets and individual users quickly becomes tedious: writing a separate permission or prohibition for every movie, and for every customer, does not scale. ODRL's *collections* offer an elegant solution to this problem. Asset collections for example, as shown in Listing 4, allow policies to be defined once for whole groups of assets, avoiding duplication and making ODRL practical for large catalogues such as those of streaming services. The grouping mechanism is not limited to assets only; ODRL also defines a similar construct for parties, known as party collections.

```
ex:permissionBob a odrl:Permission;
  odrl:action odrl:stream;
  odrl:target ex:catalog;
  odrl:assignee ex:Bob;
  odrl:assigner ex:Netplus TV;
  odrl:constraint ex:temporalConstraint;
  odrl:duty ex:paymentObligation.

ex:catalog a odrl:AssetCollection.

ex:Spiderpig odrl:partOf ex:catalog.
```

Listing 4 ODRL permission with an asset collection.

In this brief introduction to the ODRL Information Model 2.2, we elaborated most concepts and how they intertwine. In the next section, we build on these concepts to provide *operational formal semantics*. This ensures evaluation and compliance can be guaranteed in each situation and additionally provide the sought-after *policy engine interoperability* which is required for decentralized ODRL enforcement.

However, some concepts are omitted from our formal semantics: **profiles**, **rightOperandReference**, **composite rules** and **compact policies**; nevertheless the formalism is constructed in such a way that they can be readily incorporated in future versions.

The first concept to leave out is profiles. The core ODRL model is domain-agnostic and as such covers a broad range of actions and constraints, though this is non exhaustive. To capture domain-specific elements, ODRL itself defines *profiles* as a property of a policy (through the `profile` property). As an example, there is the Data Privacy Vocabulary (DPV) for representing legal requirements related to data protection and AI regulation [62]. The vocabulary allows to describe the relevant legal terms of regulations, e.g., the GDPR [5], in a machine-readable manner. As it is impossible to represent interoperable regulatory compliant policies with core ODRL, the DPV-ODRL profile[31] was created to fill in that gap, creating an affordance to represent compliance requirements of relevant acts [28].

A subtler exclusion is the deliberate omission of the `rightOperandReference`. It enables dynamic constraint values, so limits (e.g., request quotas) can be updated without modifying the policy itself. This allows the resource host to autonomously adjust rate limits in real time based on its own conditions or load. However, the issue with the `rightOperandReference` property is the ambiguity in how to actually get that value itself after you dereference the URI [15, 61].

Finally, we decided to abstain from using composite rules and compact policies. The rules elaborated thus far were all atomic rules, where the cardinalities are very strict (i.e., always one target and maximum one asset, assignee, ...). *Composite rules* extend these cardinalities, as syntactic sugar, though relate to atomic rules with the following interpretation: a composite rule with multiple Assets, Parties, and/or Actions is equal to the set of distinct atomic rules formed by the Cartesian product of those properties. The second syntactic scheme sustaining compression is *compact policies*: here Assets, Parties, and/or Actions properties are placed on a policy level with the meaning that they apply to all rules contained within the policy. Composite rules and compact policies thus allow creating very expressive small policies, though they interpretation lies in normative, atomic policies. Furthermore, ODRL describes how to atomize policies and even recommends to expand policies before conformance calculation. As such, in our formal semantics for operational compliance, we only employ atomic policies.

[31] The mappings can be found at https://w3id.org/dpv/mappings/odrl. There is a guide that aids understanding of this mapping at https://w3id.org/dpv/mappings/odrl.

4 ODRL Formalism

This section presents an *operational formal semantics* for ODRL. By operational, we mean a formalisation that specifies how evaluation and compliance decisions are computed from policies, evaluation requests, and states of the world. The semantics addresses two problems. The first is **evaluation**, the access-control view of ODRL: given a policy, an evaluation request, and a state of the world, is the requested execution allowed? The second is **compliance**, the monitoring view of ODRL: given a policy and a state of the world containing recorded contextualised executions, does the observed behaviour conform to that policy?

We develop the semantics bottom-up. We first introduce basic concepts required to express evaluation and compliance, starting from *evaluation requests* and *rules*, which share a common structure that we call an *execution*. Building upon these definitions, we introduce the semantics of rule-level functions that determine when a rules' premises hold and how they contribute to evaluation and compliance decisions. We then turn to the same questions at the policy level, where a set of (potentially conflicting) rules is evaluated together and the resulting impact on evaluation and compliance is determined. Finally, we show how the resulting evaluation and compliance procedures realise the operational semantics introduced at the beginning of this section by providing formal answers to both access-control and monitoring questions.

4.1 The Basics: Execution, Rule and Evaluation Request

To make the distinction between evaluation requests and rules concrete, we begin with a border-control scenario that exhibits both their differences in meaning and similarities in structure. Consider a case in which a person from country Y wants to enter country X for tourism, and the currently known context includes their nationality, age, travel purpose, and visa status. Country X has its own entry rules, and these rules form the authoritative source. One rule may permit entry for tourists from country Y. Another rule may prohibit entry for persons from country Z. A third rule may require that a visa be obtained before entry is permitted. A more general duty may require every person to have a passport issued by their country of origin.

Before traveling to a country, people typically want to understand whether their trip is feasible. As such, they pose an evaluation request against the rules. For example, they may ask whether they are permitted or prohibited to enter country X, based on their context. They might wonder whether some duty must first be fulfilled, such as obtaining a visa or presenting a valid passport.

While the meaning differs for evaluation request and rule they do share a lot of similarities which we refer to as *execution* and is defined in Definition 1.

Definition 1 (Execution) The set of all executions is defined as

$$\mathrm{E} \coloneqq \Delta_{\mathrm{S}} \times \Delta_{\mathrm{A}} \times \Delta_{\mathrm{O}} \times \mathscr{P}(\mathrm{K}) \times \mathscr{P}(\Gamma_{\texttt{obl}}).$$

An execution ε is a tuple $\varepsilon = \langle \delta_{\mathrm{S}}, \delta_{\mathrm{A}}, \delta_{\mathrm{O}}, C, \mathscr{D} \rangle \in \mathrm{E}$, where $\delta_{\mathrm{S}} \in \Delta_{\mathrm{S}}$, $\delta_{\mathrm{A}} \in \Delta_{\mathrm{A}}$, $\delta_{\mathrm{O}} \in \Delta_{\mathrm{O}}$, $C \subseteq \mathrm{K}$, and $\mathscr{D} \subseteq \Gamma_{\texttt{obl}}$. An execution captures the information shared by queries and all rule types, thus absent of a deontic type.

We denote by Δ_{S}, Δ_{A}, and Δ_{O} the domains of parties (subjects), actions, and assets (objects), respectively. Their elements, $\delta_{\mathrm{S}} \in \Delta_{\mathrm{S}}$, $\delta_{\mathrm{A}} \in \Delta_{\mathrm{A}}$, and $\delta_{\mathrm{O}} \in \Delta_{\mathrm{O}}$, represent sets of parties, actions, and assets[32]. We write $\mathscr{P}(\mathrm{K})$ for the powerset of a set K, i.e., the set of all subsets of K. Constraints C further restrict the circumstances under which the execution is applicable and duties $\mathscr{D}$ associate a set of obligation rules as $\Gamma_{\texttt{obl}}$ represents the set of all duty rules. Both *constraints* and *rules* will be formalised and elaborated later in this section.

For convenience, we define the following accessor functions:

$$\mathsf{party}(\varepsilon) : \mathrm{E} \to \Delta_{\mathrm{S}} \coloneqq \langle \delta_{\mathrm{S}}, \delta_{\mathrm{A}}, \delta_{\mathrm{O}}, C, \mathscr{D} \rangle \mapsto \delta_{\mathrm{S}}$$

$$\mathsf{action}(\varepsilon) : \mathrm{E} \to \Delta_{\mathrm{A}} \coloneqq \langle \delta_{\mathrm{S}}, \delta_{\mathrm{A}}, \delta_{\mathrm{O}}, C, \mathscr{D} \rangle \mapsto \delta_{\mathrm{A}}$$

$$\mathsf{asset}(\varepsilon) : \mathrm{E} \to \Delta_{\mathrm{O}} \coloneqq \langle \delta_{\mathrm{S}}, \delta_{\mathrm{A}}, \delta_{\mathrm{O}}, C, \mathscr{D} \rangle \mapsto \delta_{\mathrm{O}}$$

$$\mathsf{constraints}(\varepsilon) : \mathrm{E} \to \mathscr{P}(\mathrm{K}) \coloneqq \langle \delta_{\mathrm{S}}, \delta_{\mathrm{A}}, \delta_{\mathrm{O}}, C, \mathscr{D} \rangle \mapsto C$$

$$\mathsf{duties}(\varepsilon) : \mathrm{E} \to \mathscr{P}(\Gamma_{\texttt{obl}}) \coloneqq \langle \delta_{\mathrm{S}}, \delta_{\mathrm{A}}, \delta_{\mathrm{O}}, C, \mathscr{D} \rangle \mapsto \mathscr{D}$$

An execution on itself does not provide much meaning. The construct does not allow to ask questions of what can/cannot/should be done. To introduce meaning, we have the introduce the concept *rule* with a deontic concept [60] in definition 2.

Definition 2 (Rule) Let $\Theta \coloneqq \{\texttt{perm}, \texttt{forb}, \texttt{obl}\}$. The set of all rules is defined as

$$\Gamma \coloneqq \Theta \times \mathrm{E}.$$

A rule γ is a tuple $\gamma = \langle \theta, \varepsilon \rangle \in \Gamma$, where $\theta \in \Theta$ and $\varepsilon \in \mathrm{E}$. A rule associates an execution ε with a deontic type θ, which determines whether the execution is interpreted as a permission (`perm`), prohibition (`forb`), or obligation (`obl`).

We define the following accessor functions for executions, which are used throughout the remainder of the paper:

$$\mathsf{type}(\gamma) : \Gamma \to \Theta \coloneqq \langle \theta, \varepsilon \rangle \mapsto \theta$$

$$\mathsf{execution}(\gamma) : \Gamma \to \mathrm{E} \coloneqq \langle \theta, \varepsilon \rangle \mapsto \varepsilon$$

[32] We use *subject*, *object*, and *action* as synonyms for ODRL *party*, *asset*, and *action*, respectively, following terminology commonly used in access-control models and policy languages [63, 64].

With the explanation of a rule, four out of five concepts from the execution are elaborated. Before introducing evaluation requests, we first define the untouched fifth concept of rules — *constraints* — in definition 3.

Definition 3 (Constraint) The set of all constraints is defined as

$$\mathrm{K} \coloneqq \mathrm{K_a} \cup \mathrm{K_l},$$

where $\mathrm{K_a} \cap \mathrm{K_l} = \varnothing$. A constraint $\kappa \in \mathrm{K}$ is either an atomic constraint $\kappa_\mathrm{a} \in \mathrm{K_a}$ or a logical constraint $\kappa_\mathrm{l} \in \mathrm{K_l}$.

Constraints are divided into atomic constraints and logical constraints. Atomic constraints (Definition 4) compare a value extracted from a subject against a reference value, whereas logical constraints (Definition 5) combine one or more constraints through logical operators. As the meaning of atomic and logical operators only becomes apparent when evaluating constraints, their semantics are introduced together with the satisfaction function (satisfy, see Function 5).

Definition 4 (Atomic Constraint) The set of all atomic constraints is defined as

$$\mathrm{K_a} \coloneqq \mathrm{I} \times \Lambda \times \Xi_\mathrm{a} \times \Upsilon.$$

An atomic constraint κ_a is a tuple $\kappa_\mathrm{a} = \langle \iota, \lambda, \xi_\mathrm{a}, \upsilon \rangle \in \mathrm{K_a}$, where $\iota \in \mathrm{I}$, $\lambda \in \Lambda$, $\xi_\mathrm{a} \in \Xi_\mathrm{a}$, and $\upsilon \in \Upsilon$. We denote I the set of all the constraint subjects, Λ the set of all selectors, Ξ_a the set of all atomic operators, and Υ the set of all values. An atomic constraint compares a value obtained by applying the selector λ to the constraint subject ι against the reference value υ using the atomic operator ξ_a.

The components of an atomic constraint can be obtained through the following accessor functions:

$$\mathsf{subj}(\kappa_\mathrm{a}) : \mathrm{K_a} \to \mathrm{I} \coloneqq \langle \iota, \lambda, \xi_\mathrm{a}, \upsilon \rangle \mapsto \iota$$

$$\mathsf{select}(\kappa_\mathrm{a}) : \mathrm{K_a} \to \Lambda \coloneqq \langle \iota, \lambda, \xi_\mathrm{a}, \upsilon \rangle \mapsto \lambda$$

$$\mathsf{operator_a}(\kappa_\mathrm{a}) : \mathrm{K_a} \to \Xi_\mathrm{a} \coloneqq \langle \iota, \lambda, \xi_\mathrm{a}, \upsilon \rangle \mapsto \xi_\mathrm{a}$$

$$\mathsf{value}(\kappa_\mathrm{a}) : \mathrm{K_a} \to \Upsilon \coloneqq \langle \iota, \lambda, \xi_\mathrm{a}, \upsilon \rangle \mapsto \upsilon$$

Definition 5 (Logical Constraint) The set of all logical constraints is defined as

$$\mathrm{K_l} \coloneqq \Xi_\mathrm{l} \times \mathscr{P}_{\geq 2}(\mathrm{K}),$$

where $\mathscr{P}_{\geq 2}(\mathrm{K}) = \{C \subseteq \mathrm{K} \mid |C| \geq 2\}$, i.e., the set of all subsets of constraints containing at least two constraints. A logical constraint κ_l is a tuple $\kappa_\mathrm{l} = \langle \xi_\mathrm{l}, K \rangle \in \mathrm{K_l}$, where $\xi_\mathrm{l} \in \Xi_\mathrm{l}$ and $C \in \mathscr{P}_{\geq 2}(\mathrm{K})$. A logical constraint combines two or more constraints using a logical operator.

To access the operator and operands of a logical constraint, we define the following accessor functions:

$$\mathsf{operator_l}(\kappa_\mathrm{l}) : \mathrm{K_l} \to \Xi_\mathrm{l} \coloneqq \langle \xi_\mathrm{l}, K \rangle \mapsto \xi_\mathrm{l}$$

$$\mathsf{operands}(\kappa_\mathrm{l}) : \mathrm{K_l} \to \mathscr{P}_{\geq 2}(\mathrm{K}) \coloneqq \langle \xi_\mathrm{l}, K \rangle \mapsto K$$

With the definition of atomic and logical constraints, all components of both executions and rules have now been formalised. We therefore return to the distinction between rules and evaluation requests introduced at the beginning of this section. Whereas rules provide meaning to an execution with a deontic concept, an evaluation request provides the contextualised execution that is to be evaluated against those rules. We formalise *evaluation requests* in Definition 6.

Definition 6 (Evaluation Request) The set of all evaluation requests is defined as

$$\mathrm{Q} \coloneqq \{\gamma \in \Gamma \mid \mathsf{constraints}(\mathsf{execution}(\gamma)) \subseteq \mathrm{K_a} \\ \wedge\, \mathsf{duties}(\mathsf{execution}(\gamma)) = \emptyset\}.$$

An evaluation request $\mathrm{q} \in \mathrm{Q}$ is a rule whose execution contains only atomic constraints and no duties.

The interpretation of an evaluation request depends on the associated deontic type. A permission request asks whether a particular execution is allowed under the applicable rules. A prohibition request asks whether a particular execution is forbidden under those rules. An obligation request asks whether a particular execution is required to occur. While the definition permits all three request types, the remainder of this paper focuses exclusively on permission requests.

Example 1 (Rule and Evaluation Request) As a concrete example, consider the policy shown in Listing 1, which contains a permission rule stating that Bob may stream *Spiderpig*. In our formalism, this rule can be represented as

$$\gamma = \langle \mathtt{perm}, \langle \{Bob\}, \{stream\}, \{Spiderpig\}, \varnothing, \varnothing \rangle \rangle .$$

Suppose Bob wishes to stream *Spiderpig*. This corresponds to the evaluation request

$$\mathrm{q} = \langle \mathtt{perm}, \langle \{Bob\}, \{stream\}, \{Spiderpig\}, \varnothing, \varnothing \rangle \rangle .$$

Intuitively, the evaluation request asks whether Bob is allowed to stream *Spiderpig*. Since the request matches the permission rule from Listing 1 and neither constraints nor duties need to be respectively satisfied or fulfilled, we would expect the request to be allowed.

In the following sections, we formalise precisely how to arrive at this conclusion.

4.2 Rule-level Evaluation and Compliance

Building on the concepts introduced in the previous subsection, we now formalise the evaluation of evaluation requests against rules. This subsection introduces four functions that constitute the rule evaluation process. The function satisfy determines whether the constraints of a rule are satisfied. We introduce the active function to determine the outcome of evaluating an evaluation request against an execution. The function exec incorporates the deontic type of the rule to determine whether the evaluation request is allowed or not. Finally, some rule evaluations depend on whether associated duties have been fulfilled. As duty fulfillment is determined through the compliance of the corresponding duty rules, we also define the function compliance, which determines whether a rule is compliant. Both exec and compliance additionally depend on a State of the World, introduced later in this section, which captures the contextual information and execution history relevant to evaluation and compliance.

4.2.1 Satisfaction of Constraint

The definition of atomic constraints $\mathrm{K_a}$ (Definition 4) differs from the ODRL representation of an atomic constraint as a triple consisting of a left operand, an operator, and a right operand. The reason is that ODRL only defines *what* must be compared, but not *how* the value is obtained in the evaluation. For example, in the ODRL constraint `purpose = research`, the operator and right operand are explicit, but the mechanism that resolves the left operand `purpose` to a concrete value remains unspecified.

We define atomic operators as

$$\Xi_\mathrm{a} \coloneqq \Upsilon \times \Upsilon \to \{\texttt{satisfied}, \texttt{not_satisfied}\},$$

and use values from the domain Υ as right operands. An atomic operator compares two values and determines whether the corresponding constraint is `satisfied`. The remaining challenge is the left operand. Rather than representing it directly, we model it through a constraint subject and a selector. The set of selectors is defined as

$$\Lambda \coloneqq \mathrm{I} \times \mathrm{S} \to \Upsilon.$$

Intuitively, a constraint subject identifies the entity of interest, while a selector specifies how values are obtained from that entity in a given state of the world S.

This still leaves one question: how are values obtained during evaluation? For that purpose we use *evaluation requests* $\mathrm{q} \in \mathrm{Q}$, which provide the context required to interpret constraints. If an evaluation request and a rule contain atomic constraints with the same subject and selector, they refer to the same underlying value. The evaluation request thus provides the values for a given subject-selector pair in the current context, while the rule determines the values that satisfy the corresponding constraint for that same subject-selector pair. Intuitively, a constraint is satisfied when every value consistent with the evaluation request is also permitted by the rule.

As a consequence, constraint satisfaction is defined relative to an evaluation request:

$$\mathsf{satisfy} : \mathrm{Q} \times \mathrm{K} \to \{\texttt{satisfied}, \texttt{not_satisfied}\}.$$

We first define a helper function that retrieves the atomic constraints in an evaluation request that refer to the same left operand as a given atomic constraint:

$$\mathsf{matchConstraint} : \mathrm{Q} \times \mathrm{K_a} \to \mathscr{P}(\mathrm{K_a}).$$

For an evaluation request q and an atomic constraint κ_a, this function is defined as follows:

Function 1 matchConstraint

$$\mathsf{matchConstraint}(\mathrm{q}, \kappa_\mathrm{a}) \coloneqq \{\, \kappa'_\mathrm{a} \in \mathsf{constraints}(\mathsf{execution}(\mathrm{q})) \mid \mathsf{subj}(\kappa'_\mathrm{a}) = \mathsf{subj}(\kappa_\mathrm{a}) \wedge \mathsf{select}(\kappa'_\mathrm{a}) = \mathsf{select}(\kappa_\mathrm{a}) \,\}.$$

Since evaluation requests contain only atomic constraints by Definition 6, the result of this function is a set of atomic constraints.

We also define the admissible values of an atomic constraint. These are the values that satisfy the constraint operator against its reference value:

Function 2 adm

$$\mathsf{adm}(\kappa_\mathrm{a}) : \mathrm{K_a} \to \mathscr{P}(\Upsilon) \coloneqq \langle \iota, \lambda, \xi_\mathrm{a}, \upsilon \rangle \mapsto \left\{ \upsilon' \in \Upsilon \mid \xi_\mathrm{a}(\upsilon', \upsilon) = \texttt{satisfied} \right\}.$$

Now we can define $\mathsf{satisfy}_\mathrm{a}$, the atomic satisfaction function. Let $M = \mathsf{matchConstraint}(\mathrm{q}, \kappa_\mathrm{a})$. The request satisfies the atomic constraint if at least one matching request constraint exists, the matching request constraints admit at least one common value, and every value admitted by the rule constraint. Formally,

Function 3 $\mathsf{satisfy}_\mathrm{a}$

$$\mathsf{satisfy}_\mathrm{a}(\mathrm{q}, \kappa_\mathrm{a}) \coloneqq \begin{cases} \texttt{not_satisfied} & \text{if } M = \varnothing, \\ \texttt{not_satisfied} & \text{if } \bigcap_{\kappa'_\mathrm{a} \in M} \mathsf{adm}(\kappa'_\mathrm{a}) = \varnothing, \\ \texttt{not_satisfied} & \text{if } \bigcap_{\kappa'_\mathrm{a} \in M} \mathsf{adm}(\kappa'_\mathrm{a}) \not\subseteq \mathsf{adm}(\kappa_\mathrm{a}), \\ \texttt{satisfied} & \text{otherwise.} \end{cases}$$

Logical constraints lift satisfaction from individual constraints to a compound constraint. Recall Definition 5 which states that a logical constraint contains an operator Ξ_l and a set of operands, which are in itself constraints K. Each operand is first evaluated using satisfy, after which the logical operator combines the obtained satisfaction results. For a logical constraint $\kappa_\mathrm{l} = \langle \xi_\mathrm{l}, K \rangle$ with $K = \{\kappa_1, \ldots, \kappa_n\}$, where $n \geq 2$, logical satisfaction is defined as follows.

Function 4 $\mathsf{satisfy}_\mathrm{l}$

$$\mathsf{satisfy}_\mathrm{l}(\mathrm{q}, \kappa_\mathrm{l}) \coloneqq \mathsf{operator}_\mathrm{l}(\kappa_\mathrm{l})\,(\mathsf{satisfy}(\mathrm{q}, \kappa_1), \ldots, \mathsf{satisfy}(\mathrm{q}, \kappa_n))\,.$$

ODRL defines four logical operators: `odrl:and`, `odrl:andSequence`, `odrl:or`, and `odrl:xone`. Bonatti et al. [56] observe in Remark 1 that, given the currently specified semantics of ODRL constraints, `odrl:andSequence` does not lead to a different evaluation result than `odrl:and`. Therefore, as in their formalisation, we do not consider `odrl:andSequence` further and restrict Ξ_l to `odrl:and`, `odrl:or`, and `odrl:xone`. Their operands are constraints whose satisfaction states belong to $\{\texttt{satisfied}, \texttt{not_satisfied}\}$. The operator `odrl:and` is satisfied when all operands are satisfied, `odrl:or` when at least one operand is satisfied, and `odrl:xone` when exactly one operand is satisfied.

The general satisfaction function is defined according to the type of constraint:

Function 5 satisfy

$$\mathsf{satisfy}(\mathrm{q}, \kappa) \coloneqq \begin{cases} \mathsf{satisfy}_\mathrm{a}(\mathrm{q}, \kappa) & \text{if } \kappa \in \mathrm{K}_\mathrm{a}, \\ \mathsf{satisfy}_\mathrm{l}(\mathrm{q}, \kappa) & \text{if } \kappa \in \mathrm{K}_\mathrm{l}. \end{cases}$$

Example 2 (Satisfaction of the temporal constraint) Consider the temporal constraint from Listing 2, which restricts the permission to stream Spiderpig until the end of May 2026. In the formalism, this constraint is represented as

$$\kappa_\mathrm{time} = \langle \iota_\mathrm{context}, \lambda_\mathrm{dateTime}, \leq, \texttt{2026-05-31T23:59:59.999Z} \rangle\,.$$

The subject ι_context identifies the contextual information source from which the selector $\lambda_\mathrm{dateTime}$ retrieves the datetime value during evaluation.

Recall the evaluation request q from Example 1:

$$\mathrm{q} = \langle \texttt{perm}, \langle \{Bob\}, \{stream\}, \{Spiderpig\}, \varnothing, \varnothing \rangle \rangle\,.$$

Evaluating $\mathsf{satisfy}(\mathrm{q}, \kappa_\mathrm{time})$ results in `not_satisfied`. The reason is that the evaluation request contains no atomic constraints, formally $\mathsf{matchConstraint}(\mathrm{q}, \kappa_\mathrm{time}) = \varnothing$ which violates the first condition of $\mathsf{satisfy}_\mathrm{a}$.

Next, consider a refined evaluation request (q_{2027}) containing the atomic constraint

$$\kappa_{2027} = \langle \iota_\mathrm{context}, \lambda_\mathrm{dateTime}, \geq, \texttt{2027-01-01T00:00:00Z} \rangle$$

and let

$$\mathrm{q}_{2027} = \langle \texttt{perm}, \langle \{Bob\}, \{stream\}, \{Spiderpig\}, \{\kappa_{2027}\}, \varnothing \rangle \rangle\,.$$

In this case, $\mathsf{matchConstraint}(\mathrm{q}_{2027}, \kappa_\mathrm{time})$ contains exactly one constraint because the subject and selector coincide. The first condition is therefore satisfied. Furthermore, the admissible values of the request constraint contain every timestamp on or after 1 January 2027 and are thus non-empty, satisfying the second condition.

However, $\mathsf{adm}(\kappa_{2027}) \not\subseteq \mathsf{adm}(\kappa_\mathrm{time})$ because no timestamp from 2027 satisfies the requirement of being on or before 31 May 2026. The third condition therefore fails and thus $\mathsf{satisfy}(\mathrm{q}_{2027}, \kappa_\mathrm{time})$ is also `not_satisfied`.

Finally, consider a refined evaluation request containing the atomic constraint

$$\kappa_{2026} = \langle \iota_\mathrm{context}, \lambda_\mathrm{dateTime}, =, \texttt{2026-05-15T12:00:00Z} \rangle\,,$$

and let

$$\mathrm{q}_{2026} = \langle \texttt{perm}, \langle \{Bob\}, \{stream\}, \{Spiderpig\}, \{\kappa_{2026}\}, \varnothing \rangle \rangle\,.$$

Again, $\mathsf{matchConstraint}(\mathrm{q}_{2026}, \kappa_\mathrm{time})$ contains exactly one constraint since the subject and selector coincide. Furthermore, $\mathsf{adm}(\kappa_{2026}) = \{\texttt{2026-05-15T12:00:00Z}\}$, satisfying the second condition. Moreover, $\mathsf{adm}(\kappa_{2026}) \subseteq \mathsf{adm}(\kappa_\mathrm{time})$ holds since 15 May 2026 is before the deadline of 31 May 2026. The third condition is therefore also satisfied, yielding $\mathsf{satisfy}(\mathrm{q}_{2026}, \kappa_\mathrm{time}) = \texttt{satisfied}$.

4.2.2 Activation of a Rule

We define a rule to be `active` for a given evaluation request when all of its *premises* are `satisfied`. In the previous section, we introduced the satisfy function to determine whether constraints K are `satisfied`. However, constraints are only one type of *premise*. In our formalism, the premises of a rule consist of its subject domain Δ_S, action domain Δ_A, object domain Δ_O, and constraint set $\mathscr{P}(\mathrm{K})$.

To determine whether a rule is `active`, we must therefore verify both that the subject, action, and object domains of the evaluation request match those of the rule, and that all associated constraints are `satisfied`. Duties are deliberately excluded from activation since their interpretation depends on the deontic type of the rule, whereas activation is defined independently of that type.

The activation function has the following signature

$$\mathsf{active} : \Gamma \times \mathrm{Q} \to \{\texttt{active}, \texttt{not_active}\}.$$

Let $\varepsilon_\gamma = \mathsf{execution}(\gamma)$, $\varepsilon_\mathrm{q} = \mathsf{execution}(\mathrm{q})$ and $\kappa_\mathsf{Violation}(\gamma, \mathrm{q}) = \exists \kappa \in \mathsf{constraints}(\varepsilon_\gamma) : \mathsf{satisfy}(\mathrm{q}, \kappa) = \texttt{not_satisfied}$. Activation is then defined as

Function 6 active

$$\mathsf{active}(\gamma, \mathrm{q}) \coloneqq \begin{cases} \texttt{not_active} & \text{if } \mathsf{party}(\varepsilon_\mathrm{q}) \not\sqsubseteq \mathsf{party}(\varepsilon_\gamma), \\ \texttt{not_active} & \text{if } \mathsf{action}(\varepsilon_\mathrm{q}) \not\sqsubseteq \mathsf{action}(\varepsilon_\gamma), \\ \texttt{not_active} & \text{if } \mathsf{asset}(\varepsilon_\mathrm{q}) \not\sqsubseteq \mathsf{asset}(\varepsilon_\gamma), \\ \texttt{not_active} & \text{if } \kappa_\mathsf{Violation}(\gamma, \mathrm{q}) \\ \texttt{active} & \text{otherwise.} \end{cases}$$

The relation $\sqsubseteq$ denotes the partial order on the corresponding domains. It is used because the elements of these domains represent sets of subjects, actions, or objects, while also allowing activation to account for specialisation relationships[33].

Example 3 (Activation of a rule) Recall the permission rule γ and evaluation request q from Example 1:

$$\gamma = \langle \texttt{perm}, \langle \{Bob\}, \{stream\}, \{Spiderpig\}, \varnothing, \varnothing \rangle \rangle,$$

$$\mathrm{q} = \langle \texttt{perm}, \langle \{Bob\}, \{stream\}, \{Spiderpig\}, \varnothing, \varnothing \rangle \rangle.$$

Since the subject, action, and asset domains of the evaluation request are identical to those of the rule, and no constraints are present, all premises of the rule hold. Therefore, $\mathsf{active}(\gamma, \mathrm{q}) = \texttt{active}$.

To illustrate the impact of constraints on activation, we extend the permission rule from Section 4.1 with the temporal constraint κ_{time} from Listing 2:

$$\gamma_{\text{time}} = \langle \texttt{perm}, \langle \{Bob\}, \{stream\}, \{Spiderpig\}, \{\kappa_{\text{time}}\}, \varnothing \rangle \rangle.$$

From Example 2, we know that $\mathsf{satisfy}(\mathrm{q}_{2027}, \kappa_{\text{time}}) = \texttt{not_satisfied}$. Although the subject, action, and asset domains still match, not all constraints are satisfied. Consequently, $\mathsf{active}(\gamma_{\text{time}}, \mathrm{q}_{2027}) = \texttt{not_active}$.

Activation identifies the rules whose premises hold for a given evaluation request. Whether such an active rule permits or forbids the requested execution is elaborated in the next section.

4.2.3 Execution of a Rule

The execution function (exec) builds upon the activation function and is defined for permission and prohibition rules only. It extends the activation function by taking the deontic type of the rule into account to determine whether the requested execution is `allowed` or not.

A complication arises for permission rules with associated duties. Activation alone is insufficient to conclude that the requested execution is `allowed`, because ODRL states that if a permission has duties, *"then all of the Duties MUST be agreed to be fulfilled"*[34]. This introduces the additional question of whether those duties have in fact been fulfilled. Following the operational interpretation proposed by Bonatti et al. [56], we require all associated duties to be fulfilled before the permission can be exercised[35]. Determining whether a duty has been fulfilled requires knowledge about past executions. To capture this historical information, we formalise in Definition 7 the state of the world $\mathrm{s} \in \mathrm{S}$, a concept briefly introduced in Section 4.2.1.

Definition 7 (State of the World) The set of all states of the world is denoted by S. A state of the world $\mathrm{s} \in \mathrm{S}$ represents knowledge about the world at a particular point in time. More specifically it contains a set of past contextualised executions

$$\Psi := \mathrm{E} \times \mathrm{S}.$$

Thus, a contextualised execution $\psi = \langle \varepsilon, \mathrm{s}' \rangle \in \Psi$ represents an execution ε that was performed under the contextual information captured by the state of the world s'.

We define the following accessor functions:

$$\mathsf{time} : \mathrm{S} \to \Delta_{\mathrm{dt}} \qquad \mathsf{performances} : \mathrm{S} \to \mathscr{P}(\Psi)$$

For a state of the world $\mathrm{s} \in \mathrm{S}$, $\mathsf{time}(\mathrm{s})$ returns the point in time represented by s, whereas $\mathsf{performances}(\mathrm{s})$ returns the set of contextualised executions known until that point in time. Consequently, if $\langle \varepsilon, \mathrm{s}' \rangle \in \mathsf{performances}(\mathrm{s})$, then $\mathsf{time}(\mathrm{s}') < \mathsf{time}(\mathrm{s})$.

The historical information captured by a state of the world allows us to determine whether duties have been fulfilled. To calculate fulfillment of duties from a state of the world, the execution function relies on the rule compliance function compliance (formalised in Section 4.2.4).

Having introduced the state of the world, we can now formalise rule execution. The execution function extends activation with the deontic interpretation of a rule and, for permission rules, additionally takes duty fulfillment into account through the compliance function.

Let $\mathscr{D}_\gamma := \mathsf{duties}(\mathsf{execution}(\gamma))$ for a rule $\gamma \in \Gamma$, $\Gamma_{\texttt{perm}} := \{\gamma_{\texttt{perm}} \in \Gamma \mid \mathsf{type}(\gamma_{\texttt{perm}}) = \texttt{perm}\}$, and $\Gamma_{\texttt{forb}} := \{\gamma \in \Gamma \mid \mathsf{type}(\gamma) = \texttt{forb}\}$. The execution function has the following signature:

$$\begin{aligned}\mathsf{exec} :& (\Gamma_{\texttt{perm}} \cup \Gamma_{\texttt{forb}}) \times \mathrm{Q} \times \mathrm{S} \\ & \to \{\texttt{allowed}, \texttt{not_applicable}, \texttt{not_allowed}\}.\end{aligned}$$

Its formal definition is given in Function 7.

Given a permission or prohibition rule $\gamma \in (\Gamma_{\texttt{perm}} \cup \Gamma_{\texttt{forb}})$, an evaluation request q, and a state of the world s, exec thus determines whether the requested execution is allowed or not. A permission rule evaluates to `allowed` when the rule is `active` and all associated duties are compliant in the current state of the world. Otherwise, the rule is `not_applicable`. In contrast, a prohibition rule evaluates to `not_allowed` when the rule is `active`, and to `not_applicable` otherwise. Since exec evaluates a single rule in isolation, it does not determine a final policy decision. A permission rule can therefore only contribute an `allowed` outcome, while a prohibition rule can only contribute a `not_allowed` outcome. The combination of multiple rule outcomes into a final decision is formalised in Section 4.3.

[33] This is particularly relevant for ODRL actions, which are organised hierarchically.

[34] Section 2.6.5 of the ODRL Information Model [8].

[35] We follow definition 6 of [56] as a reaction to their fourth remark: duties must be fulfilled before an action can be exercised.

Function 7 exec

$$\mathsf{exec}(\gamma, \mathrm{q}, \mathrm{s}) := \begin{cases} \texttt{allowed} & \text{if } \mathsf{type}(\gamma) = \texttt{perm} \wedge \mathsf{active}(\gamma, \mathrm{q}) = \texttt{active} \\ & \wedge \bigwedge_{\gamma_{\mathrm{obl}} \in \mathscr{D}_\gamma} \Big(\mathsf{compliance}(\gamma_{\mathrm{obl}}, \mathrm{s}) = \texttt{compliant} \Big) \\ \texttt{not_allowed} & \text{if } \mathsf{type}(\gamma) = \texttt{forb} \wedge \mathsf{active}(\gamma, \mathrm{q}) = \texttt{active} \\ \texttt{not_applicable} & \text{otherwise.} \end{cases}$$

Example 4 (Execution of a rule) Let $\mathrm{s}_\emptyset \in \mathrm{S}$ be an arbitrary state of the world such that $\mathsf{performances}(\mathrm{s}_\emptyset) = \varnothing$.

Recall the permission rule with temporal constraint introduced in Example 3:

$$\gamma_{\mathrm{time}} = \langle \texttt{perm}, \langle \{Bob\}, \{stream\}, \{Spiderpig\}, \{\kappa_{\mathrm{time}}\}, \varnothing \rangle \rangle .$$

Furthermore, let $\gamma^{\mathrm{forb}}_{\mathrm{time}} = \langle \texttt{forb}, \mathsf{execution}(\gamma_{\mathrm{time}}) \rangle$ be the prohibition rule obtained by assigning the same execution a prohibition type. From Example 3, we know that $\mathsf{active}(\gamma_{\mathrm{time}}, \mathrm{q}_{2026}) = \texttt{active}$.

Since γ_{time} is a permission rule and has no duties, $\mathsf{exec}(\gamma_{\mathrm{time}}, \mathrm{q}_{2026}, \mathrm{s}_\emptyset) = \texttt{allowed}$. Using the same evaluation request with the prohibition rule yields $\mathsf{exec}(\gamma^{\mathrm{forb}}_{\mathrm{time}}, \mathrm{q}_{2026}, \mathrm{s}_\emptyset) = \texttt{not_allowed}$.

Finally, recall from Example 2 that $\mathsf{active}(\gamma_{\mathrm{time}}, \mathrm{q}_{2027}) = \texttt{not_active}$. Consequently, $\mathsf{exec}(\gamma_{\mathrm{time}}, \mathrm{q}_{2027}, \mathrm{s}_\emptyset) = \texttt{not_applicable}$. The result is not $\texttt{not_allowed}$ because the rule is a permission and not active, which does not provide sufficient information to conclude whether the requested execution should be denied.

The previous example only considered rules without duties. When duties are present, execution depends on whether those duties have been fulfilled. This is captured by the compliance function introduced next.

4.2.4 Rule Compliance

The final rule level function is compliance. In contrast to exec, which determines whether an action may be performed, compliance evaluates whether past contextualised executions conform to a rule. Its outcome depends on the deontic type: permissions dictate what may occur, prohibitions dictate what must not occur, and duties dictate what must occur.

We therefore define compliance over rules and states of the world:

$$\mathsf{compliance} : \Gamma \times \mathrm{S} \to \{\texttt{compliant}, \texttt{not_applicable}, \texttt{not_compliant}\}.$$

Let

$$\mathsf{request}(\varepsilon) : \mathrm{E} \to \mathrm{Q} := \varepsilon \mapsto \langle \texttt{perm}, \varepsilon \rangle,$$

then compliance is formalised in Function 8.

The function $\mathsf{request}(\varepsilon)$ assumes that $\mathsf{constraints}(\varepsilon) \subseteq \mathrm{K_a}$. This assumption holds for all executions occurring in Ψ, since performances represent previous contextualised executions requests, which only contain atomic constraints by Definition 6.

For permission rules, compliance requires the existence of a past contextualised execution that is $\texttt{allowed}$ according to exec. In the absence of an execution, the permission is $\texttt{not_applicable}$, since the permission was neither exercised nor violated.

A prohibition is $\texttt{compliant}$ when no active execution violating the prohibition is present in the state of the world. If such an execution occurs, compliance depends on the associated duties: when no duties exist or at least one duty is itself $\texttt{not_compliant}$, the prohibition is $\texttt{not_compliant}$. Otherwise, when all associated duties are $\texttt{compliant}$, the prohibition is $\texttt{compliant}$, reflecting the ODRL definition of remedies[36].

Duties focus on required behaviour. A duty is $\texttt{compliant}$ when there exists an execution in the state of the world for which the duty is $\texttt{active}$. In the absence of such an execution, duties can still be $\texttt{compliant}$ if all its duties (consequences) are $\texttt{compliant}$ themselves. If any of its duties are $\texttt{not_compliant}$, the duty itself is likewise $\texttt{not_compliant}$.

Example 5 (Rule Compliance) Consider the obligation rule

$$\gamma_{\mathrm{pay}} = \langle \texttt{obl}, \langle \{Bob\}, \{\texttt{compensate}\}, \varnothing, \varnothing, \varnothing \rangle \rangle .$$

Let $\mathrm{s}_\emptyset \in \mathrm{S}$ be the state of the world from Example 4, for which $\mathsf{performances}(\mathrm{s}_\emptyset) = \varnothing$. Furthermore, let

$$\psi_{\mathrm{pay}} = \langle \mathsf{execution}(\gamma_{\mathrm{pay}}), \mathrm{s}' \rangle$$

be a contextualised execution, and let $\mathrm{s}_{\mathrm{pay}} \in \mathrm{S}$ be a state of the world such that $\mathsf{performances}(\mathrm{s}_{\mathrm{pay}}) = \{\psi_{\mathrm{pay}}\}$. Since a matching execution of the obligation is present in the execution history, $\mathsf{compliance}(\gamma_{\mathrm{pay}}, \mathrm{s}_{\mathrm{pay}}) = \texttt{compliant}$.

Next, we extend the permission rule from Example 4 with the duty γ_{pay}:

$$\gamma^{\mathrm{pay}}_{\mathrm{time}} = \big\langle \texttt{perm}, \big\langle \{Bob\}, \{stream\}, \{Spiderpig\}, \{\kappa_{\mathrm{time}}\}, \{\gamma_{\mathrm{pay}}\} \big\rangle \big\rangle .$$

[36] Section 2.6.7 of the ODRL Information Model [8]

Function 8 compliance

$$\mathsf{compliance}(\gamma, \mathrm{s}) := \begin{cases} \texttt{compliant} & \texttt{type}(\gamma) = \texttt{perm} \wedge \exists \langle \varepsilon, \mathrm{s}' \rangle \in \mathsf{performances}(\mathrm{s}) : \\ & \mathsf{exec}(\gamma, \mathsf{request}(\varepsilon), \mathrm{s}') = \texttt{allowed} \\ \texttt{not_compliant} & \texttt{type}(\gamma) = \texttt{forb} \wedge \exists \langle \varepsilon, \mathrm{s}' \rangle \in \mathsf{performances}(\mathrm{s}) : \\ & \mathsf{active}(\gamma, \mathsf{request}(\varepsilon)) = \texttt{active} \\ & \wedge \Big(\mathscr{D}_\gamma = \varnothing \vee \exists \gamma_{\texttt{obl}} \in \mathscr{D}_\gamma : \mathsf{compliance}(\gamma_{\texttt{obl}}, \mathrm{s}) = \texttt{not_compliant} \Big) \\ \texttt{compliant} & \texttt{type}(\gamma) = \texttt{forb} \wedge \neg \exists \langle \varepsilon, \mathrm{s}' \rangle \in \mathsf{performances}(\mathrm{s}) : \\ & \mathsf{active}(\gamma, \mathsf{request}(\varepsilon)) = \texttt{active} \\ \texttt{compliant} & \texttt{type}(\gamma) = \texttt{forb} \wedge \exists \langle \varepsilon, \mathrm{s}' \rangle \in \mathsf{performances}(\mathrm{s}) : \\ & \mathsf{active}(\gamma, \mathsf{request}(\varepsilon)) = \texttt{active} \wedge \mathscr{D}_\gamma \neq \varnothing \\ & \wedge \bigwedge_{\gamma_{\texttt{obl}} \in \mathscr{D}_\gamma} \Big(\mathsf{compliance}(\gamma_{\texttt{obl}}, \mathrm{s}) = \texttt{compliant} \Big) \\ \texttt{compliant} & \texttt{type}(\gamma) = \texttt{obl} \wedge \exists \langle \varepsilon, \mathrm{s}' \rangle \in \mathsf{performances}(\mathrm{s}) : \\ & \mathsf{active}(\gamma, \mathsf{request}(\varepsilon)) = \texttt{active} \\ \texttt{compliant} & \texttt{type}(\gamma) = \texttt{obl} \wedge \exists \langle \varepsilon, \mathrm{s}' \rangle \in \mathsf{performances}(\mathrm{s}) : \\ & \mathscr{D}_\gamma \neq \varnothing \wedge \bigwedge_{\gamma_{\texttt{obl}} \in \mathscr{D}_\gamma} \Big(\mathsf{compliance}(\gamma_{\texttt{obl}}, \mathrm{s}) = \texttt{compliant} \Big) \\ \texttt{not_compliant} & \texttt{type}(\gamma) = \texttt{obl} \wedge \neg \exists \langle \varepsilon, \mathrm{s}' \rangle \in \mathsf{performances}(\mathrm{s}) : \\ & \mathsf{active}(\gamma, \mathsf{request}(\varepsilon)) = \texttt{active} \\ & \wedge \Big(\mathscr{D}_\gamma = \varnothing \vee \exists \gamma_{\texttt{obl}} \in \mathscr{D}_\gamma : \mathsf{compliance}(\gamma_{\texttt{obl}}, \mathrm{s}) = \texttt{not_compliant} \Big) \\ \texttt{not_applicable} & \text{otherwise.} \end{cases}$$

Finally, let $\psi_{\text{time}} = \langle \mathsf{execution}(\mathrm{q}_{2026}), \mathrm{s}_\emptyset \rangle$ be a contextualised execution corresponding to q_{2026}, and let $\mathrm{s}_{\text{full}} \in \mathrm{S}$ be such that $\mathsf{performances}(\mathrm{s}_{\text{full}}) = \{\psi_{\text{pay}}, \psi_{\text{time}}\}$. Since the duty is compliant and the permission has been exercised under a satisfying request, $\mathsf{compliance}(\gamma^{\text{pay}}_{\text{time}}, \mathrm{s}_{\text{full}}) = \texttt{compliant}$.

4.3 Evaluation and Compliance of a Policy

Finally, having defined evaluation and compliance at the rule level, we turn to policies. ODRL policies group together multiple rules, which introduces additional complexity. Different rules may produce contradictory conclusions for the same evaluation request. For example, a permission rule may allow an execution while a prohibition rule forbids that same execution.

To address this additional complexity, we introduce policy-level evaluation and compliance functions. The function $\mathsf{evaluate}$ determines whether an evaluation request is allowed under a policy and a given state of the world, while $\mathsf{compliance}_p$ determines whether the executions recorded in a state of the world conform to that policy.

Both functions must account for the possibility that multiple rules simultaneously apply to the same evaluation request and may produce different outcomes. ODRL addresses such situations through a conflict property that specifies how conflicts between permissions and prohibitions are resolved. In Definition 8, we therefore model a policy as a set of rules together with a conflict property.

Definition 8 (Policy) Let $\mathrm{Z} := \{\texttt{perm}, \texttt{prohibit}, \texttt{invalid}\}$[37]. The set of all policies is defined as

$$\Pi := \mathrm{Z} \times \mathscr{P}(\Gamma).$$

A policy is a tuple $\pi = \langle \zeta, \Gamma' \rangle$, where $\zeta \in \mathrm{Z}$ and $\Gamma' \subseteq \Gamma$. A policy consists of set of rules Γ' with a conflict property

[37] Alternative conflict properties are proposed in Papagiannakopoulou et al. [58], though they require more features in ODRL Rules and/or further formalisation. `lex posterior` states that if there is a conflict between two rules, the most recent prevails. This requires timestamping rules. `lex specialis` indicates that for conflicts, the more constrained rule decision prevails, and `lex superior` requires a priority scheme within rules such that in conflict the decision of the highest priority rule is taken.

ζ. The conflict property dictates the strategy employed to resolve conflicts between permission and prohibitions within a policy.

For convenience, we define the following accessor functions:

$$\texttt{rules}(\pi) : \Pi \to \mathscr{P}(\Gamma) := \langle \zeta, \Gamma' \rangle \mapsto \Gamma'$$

$$\texttt{property}(\pi) : \Pi \to \mathrm{Z} := \langle \zeta, \Gamma' \rangle \mapsto \zeta$$

We adopt the semantics of the conflict properties as defined by ODRL[38]. `perm` gives precedence to permissions, `prohibit` gives precedence to prohibitions, and `invalid` treats such conflicts as void.

4.3.1 Evaluating a Policy

Now that a policy has been formalised in Definition 8, we move from individual rules to sets of rules. The function exec (see Function 7) evaluates an evaluation request against a single rule. A policy evaluation requires an additional step: combining these results. This is not trivial since different rules may produce conflicting outcomes for the same evaluation request. The policy-level evaluation function evaluate resolves these situations according to the policy's conflict property.

The signature of evaluate is the following:

$$\mathsf{evaluate} : \Pi \times \mathrm{Q} \times \mathrm{S} \to \{\texttt{allowed}, \texttt{void}, \texttt{not_allowed}\}$$

Let $P := \texttt{property}(\pi) = \texttt{perm}$, $F := \texttt{property}(\pi) = \texttt{prohibit}$, $I := \texttt{property}(\pi) = \texttt{invalid}$,

$A := \exists \gamma \in \texttt{rules}(\pi) : \mathsf{exec}(\gamma, \mathrm{q}, \mathrm{s}) = \texttt{allowed}$,

$D := \exists \gamma \in \texttt{rules}(\pi) : \mathsf{exec}(\gamma, \mathrm{q}, \mathrm{s}) = \texttt{not_allowed}$.

Then evaluate is defined as follows:

Function 9 evaluate

$$\mathsf{evaluate}(\pi, \mathrm{q}, \mathrm{s}) := \begin{cases} \texttt{allowed} & \text{if } P \wedge A \\ \texttt{allowed} & \text{if } (F \vee I) \wedge A \wedge \neg D \\ \texttt{not_allowed} & \text{if } F \wedge D \\ \texttt{not_allowed} & \text{if } (P \vee I) \wedge D \wedge \neg A \\ \texttt{void} & \text{otherwise.} \end{cases}$$

Given a policy π, an evaluation request q, and a state of the world s, evaluate determines whether the requested execution is allowed, void, or not allowed.

When the conflict property is `perm`, the existence of at least one rule that evaluates to `allowed` through exec is sufficient to obtain the result `allowed`, regardless of whether rules evaluating to `not_allowed` also exist. For the conflict properties `prohibit` and `invalid`, an allowed decision is obtained only when at least one rule evaluates to `allowed` and no rule evaluates to `not_allowed`.

In contrast, `not_allowed` is obtained when the conflict property is `prohibit` and at least one rule evaluates to `not_allowed` through exec, regardless of whether rules evaluating to `allowed` also exist. For `perm` and `invalid`, a not-allowed decision is obtained only when at least one rule evaluates to `not_allowed` and no rule evaluates to `allowed`.

All remaining situations evaluate to `void`. This happens when no rule evaluates to either `allowed` or `not_allowed` (no matter the conflict property), and when both outcomes occur under the conflict property `invalid`. The first case corresponds to the situation where no rule evaluates to either `allowed` or `not_allowed` through exec. Instead, every rule evaluates to `not_applicable`. Consequently, the policy does not provide sufficient information to determine whether the requested execution is allowed or not. Bonatti et al. [56] identify this situation as a "*gap resolution*" problem. The second case arises directly from the semantics of the conflict property `invalid`, which dictates that conflicting permission and prohibition outcomes invalidate the policy decision.

Example 6 (Evaluating a Policy) Consider the policy formalising Listing 3, which uses the conflict property `prohibit` to give precedence to prohibitions over permissions. We reuse the time-restricted permission rule γ_{time} from Example 3 in place of the permission from Listing 2, while retaining the prohibition modelling the restriction that Spiderpig may not be streamed in Japan.

The prohibition rule is modelled as

$$\gamma_{\text{JP}} = \langle \texttt{forb}, \langle \{Bob\}, \{stream\}, \{Spiderpig\}, \{\kappa_{\text{JP}}\}, \varnothing \rangle \rangle ,$$

where $\kappa_{\text{JP}} = \langle \iota_{\text{context}}, \lambda_{\text{spatial}}, =, \text{JP} \rangle$ represents the spatial constraint.

The resulting policy is formalised as

$$\pi_{\texttt{prohibit}} = \langle \texttt{prohibit}, \{\gamma_{\text{time}}, \gamma_{\text{JP}}\} \rangle .$$

To evaluate the policy, we extend the evaluation request q_{2026} from Example 2 with the information that the request originates from Japan:

$$\mathrm{q}^{\text{JP}}_{2026} = \langle \texttt{perm}, \langle \{Bob\}, \{stream\}, \{Spiderpig\}, \{\kappa_{2026}, \kappa_{\text{JP}}\}, \varnothing \rangle \rangle .$$

From Example 3, the temporal constraint is satisfied and therefore $\mathsf{exec}(\gamma_{\text{time}}, \mathrm{q}^{\text{JP}}_{2026}, \mathrm{s}_{\emptyset}) = \texttt{allowed}$. Likewise, the spatial constraint is satisfied because the evaluation request explicitly states that the execution takes place in Japan. Hence $\mathsf{exec}(\gamma_{\text{JP}}, \mathrm{q}^{\text{JP}}_{2026}, \mathrm{s}_{\emptyset}) = \texttt{not_allowed}$.

Consequently, for the policy evaluation function we obtain

[38] Section 2.10 of the ODRL Information Model [8] defines the semantics of the conflict property.

$$A = \texttt{true} \qquad \text{and} \qquad D = \texttt{true}.$$

Since the policy uses the conflict property `prohibit`, the branch $\texttt{property}(\pi) = \texttt{prohibit} \wedge D$ applies, yielding

$$\mathsf{evaluate}(\pi_{\texttt{prohibit}}, \mathrm{q}^{\mathrm{JP}}_{2026}, \mathrm{s}_{\emptyset}) = \texttt{not_allowed}.$$

Thus, although a permission applies, the prohibition prevails and the final decision is that Bob is not allowed to stream Spiderpig in Japan.

If instead the policy were defined with conflict property `invalid`, both A and D would still hold. Neither of the branches producing `allowed` or `not_allowed` would then apply. Therefore, the result would be

$$\mathsf{evaluate}(\langle \texttt{invalid}, \{\gamma_{\mathrm{time}}, \gamma_{\mathrm{JP}}\}\rangle, \mathrm{q}^{\mathrm{JP}}_{2026}, \mathrm{s}_{\emptyset}) = \texttt{void}.$$

Although `void` accurately represents an inconclusive policy decision, it is often not sufficient for access-control enforcement. In practice, an enforcement mechanism typically requires a definitive answer, namely either `allowed` or `not_allowed`. Since ODRL does not define how `void` outcomes should be handled, we introduce an additional mechanism in the following subsection.

4.3.2 Gap Resolution Strategies

The evaluation function defined in the previous subsection may return `void`. In order to obtain an evaluation function that always returns either `allowed` or `not_allowed`, we introduce a gap resolution strategy in Definition 9.

Definition 9 (Gap Resolution Strategy) Let

$$\Sigma \coloneqq \{\texttt{deny}, \texttt{allow}\}.$$

A gap resolution strategy $\sigma \in \Sigma$ determines how evaluation outcomes of `void` are resolved. The `deny` strategy follows the closed-world assumption: every evaluation outcome other than `allowed` is resolved to `not_allowed`. Conversely, the `allow` strategy resolves every evaluation outcome other than `not_allowed` to `allowed`.

We focus on the deny strategy in the remainder of this paper.

The resulting evaluation function has the following signature:

$$\mathsf{evaluate}_s : \Pi \times \mathrm{Q} \times \mathrm{S} \times \Sigma \to \{\texttt{allowed}, \texttt{not_allowed}\}.$$

Let P, F, I, A and D be defined as in Section 4.3.1. For the strategy `deny`, policy evaluation is defined as follows:

Function 10 $\mathsf{evaluate}_s$

$$\mathsf{evaluate}_s(\pi, \mathrm{q}, \mathrm{s}, \texttt{deny}) \coloneqq \begin{cases} \texttt{allowed} & \text{if } \big(P \wedge A\big) \vee \big((F \vee I) \wedge A \wedge \neg D\big) \\ \texttt{not_allowed} & \text{otherwise.} \end{cases}$$

With the inclusion of a gap resolution strategy, $\mathsf{evaluate}_s$ always returns either `allowed` or `not_allowed`.

The result `allowed` is obtained under the same conditions as in Section 4.3.1. In other words, the `deny` gap resolution strategy does not alter evaluation outcomes that already evaluate to `allowed`.

For the `deny` strategy, all remaining situations evaluate to `not_allowed`. This includes both cases that evaluate to `not_allowed` in Section 4.3.1 and cases that previously evaluated to `void`.

Example 7 (Evaluating a Policy with a Gap Resolution Strategy) Recall the policy with invalid conflict property from Example 6:

$$\pi_{\texttt{invalid}} = \langle \texttt{invalid}, \{\gamma_{\mathrm{time}}, \gamma_{\mathrm{JP}}\}\rangle.$$

In Example 6, we showed that $\mathsf{evaluate}(\pi_{\texttt{invalid}}, \mathrm{q}^{\mathrm{JP}}_{2026}, \mathrm{s}_{\emptyset}) = \texttt{void}$ because both A and D held. Applying the deny gap resolution strategy yields $\mathsf{evaluate}_s(\pi_{\texttt{invalid}}, \mathrm{q}^{\mathrm{JP}}_{2026}, \mathrm{s}_{\emptyset}, \texttt{deny}) = \texttt{not_allowed}$.

We now consider a different situation in which neither the permission nor the prohibition applies. Suppose Bob attempts to stream Spiderpig from Belgium in 2027. This is represented by the evaluation request

$$\mathrm{q}^{\mathrm{BE}}_{2027} = \langle \texttt{perm}, \langle \{\mathit{Bob}\}, \{\mathit{stream}\}, \{\mathit{Spiderpig}\}, \{\kappa_{2027}, \kappa_{\mathrm{BE}}\}, \varnothing\rangle\rangle,$$

where $\kappa_{\mathrm{BE}} = \langle \iota_{\mathrm{context}}, \lambda_{\mathrm{spatial}}, =, \mathrm{BE}\rangle$.

From Example 2, we know that $\mathsf{satisfy}(\mathrm{q}_{2027}, \kappa_{\mathrm{time}}) = \texttt{not_satisfied}$. Therefore $\mathsf{exec}(\gamma_{\mathrm{time}}, \mathrm{q}^{\mathrm{BE}}_{2027}, \mathrm{s}_{\emptyset}) = \texttt{not_applicable}$.

Likewise, the prohibition rule is not active because the evaluation request specifies Belgium rather than Japan. Hence $\mathsf{exec}(\gamma_{\mathrm{JP}}, \mathrm{q}^{\mathrm{BE}}_{2027}, \mathrm{s}_{\emptyset}) = \texttt{not_applicable}$. Consequently,

$$A = \texttt{false} \qquad \text{and} \qquad D = \texttt{false}.$$

The policy evaluation therefore yields

$$\mathsf{evaluate}(\pi_{\texttt{invalid}}, \mathrm{q}^{\mathrm{BE}}_{2027}, \mathrm{s}_{\emptyset}) = \texttt{void}.$$

Under the deny gap resolution strategy, the evaluation becomes

$$\mathsf{evaluate}_s(\pi_{\texttt{invalid}}, \mathrm{q}^{\mathrm{BE}}_{2027}, \mathrm{s}_{\emptyset}, \texttt{deny}) = \texttt{not_allowed}.$$

Hence, the evaluation functions defined thus far determine whether a particular execution is allowed under a policy. A complementary question is whether the executions that have actually occurred conform to a policy. To answer this monitoring question, we next formalise policy compliance.

4.3.3 Policy Compliance

The final function extends compliance from individual rules to policies. Whereas the rule-level compliance function (Function 8) determines whether a single rule is compliant, policy compliance considers the interaction between all rules contained in a policy.

The result depends on the selected gap resolution strategy, and the effect of the policy's conflict property is captured through the evaluation outcomes used in the definition of compliance. As in the rule-level setting, we reuse the name compliance for the policy-level function. Its signature is:

$$\mathsf{compliance} : \Pi \times \mathrm{S} \times \Sigma \to \{\texttt{compliant}, \texttt{not_compliant}\}.$$

As with the policy evaluation function, we only elaborate the deny gap-resolution strategy in the remainder of this paper.

Let $P_s := \mathsf{performances}(\mathrm{s})$,

$$A(\pi, \varepsilon, \mathrm{s}) := \mathsf{evaluate}_s(\pi, \mathsf{request}(\varepsilon), \mathrm{s}, \texttt{deny}) = \texttt{allowed},$$

$$O(\pi, \mathrm{s}) := \bigwedge_{\substack{\gamma \in \texttt{rules}(\pi) \\ \texttt{type}(\gamma) = \texttt{obl}}} (\mathsf{compliance}(\gamma, \mathrm{s}) = \texttt{compliant}).$$

$$D(\gamma, \varepsilon, \mathrm{s}) := \mathsf{exec}(\gamma, \mathsf{request}(\varepsilon), \mathrm{s}) = \texttt{not_allowed}$$

$$\begin{aligned} M(\gamma, \mathrm{s}) := {} & \mathsf{duties}(\mathsf{execution}(\gamma)) \neq \varnothing \\ & \wedge \forall \gamma_{\texttt{obl}} \in \mathsf{duties}(\mathsf{execution}(\gamma)) : \\ & \quad \mathsf{compliance}(\gamma_{\texttt{obl}}, \mathrm{s}) = \texttt{compliant} \end{aligned}$$

$$\begin{aligned} R(\pi, \varepsilon, \mathrm{s}) := {} & \Big(\exists \gamma \in \texttt{rules}(\pi) : D(\pi, \varepsilon, \mathrm{s})\Big) \\ & \wedge \Big(\forall \gamma \in \texttt{rules}(\pi) : D(\pi, \varepsilon, \mathrm{s}) \to M(\gamma, \mathrm{s})\Big) \end{aligned}$$

Then policy compliance is defined in Function 11 as

Function 11 $\mathsf{compliance}_p$

$$\mathsf{compliance}_p(\pi, \mathrm{s}, \texttt{deny}) := \begin{cases} \texttt{compliant} & \text{if } O(\pi, \mathrm{s}) \wedge \forall \langle \varepsilon, \mathrm{s}' \rangle \in P_s : \\ & \Big(A(\pi, \varepsilon, \mathrm{s}') \vee R(\pi, \varepsilon, \mathrm{s}')\Big) \\ \texttt{not_compliant} & \text{otherwise.} \end{cases}$$

The policy compliance function determines whether the execution history recorded in a state of the world conforms to a policy. In contrast to policy evaluation, which answers an access-control question for a single evaluation request, policy compliance adopts a monitoring perspective and considers all recorded executions. Intuitively, a policy is `compliant` if every recorded execution is accounted for by the policy and all policy-level obligations are fulfilled. Under the `deny` strategy, an execution is accounted for if it was allowed by the policy (A), or if it was not allowed that all remedies associated with the denying rules are compliant (R). In addition, all duty rules defined in the policy must themselves be compliant (O).

A policy is `not_compliant` whenever one of these conditions is violated. More specifically, this occurs when there exists a recorded execution for which the policy evaluates to `not_allowed` and the remedies associated with the denying rules are not compliant, or when at least one duty rule contained in the policy is itself `not_compliant`.

Example 8 (Policy Compliance) We extend the policy from Example 6, more precisely the prohibition γ_{JP} is replaced with $\gamma_{\mathrm{JP}}^{\mathrm{remedy}}$ which is γ_{JP} with the remedy duty γ_{pay}.

$$\pi_{\mathrm{remedy}} = \left\langle \texttt{prohibit}, \{\gamma_{\mathrm{time}}, \gamma_{\mathrm{JP}}^{\mathrm{remedy}}\} \right\rangle,$$

where

$$\gamma_{\mathrm{JP}}^{\mathrm{remedy}} = \left\langle \texttt{forb}, \left\langle \{Bob\}, \{stream\}, \{Spiderpig\}, \{\kappa_{\mathrm{JP}}\}, \{\gamma_{\mathrm{pay}}\} \right\rangle \right\rangle.$$

First, consider the state of the world $\mathrm{s}_1 \in \mathrm{S}$ containing only the contextualised execution corresponding to q_{2026} (i.e. $\mathsf{performances}(\mathrm{s}_1) = \{\langle \mathsf{execution}(\mathrm{q}_{2026}) \rangle, \mathrm{s}_\emptyset\}$). Since $A(\pi_{\mathrm{remedy}}, \mathsf{execution}(\mathrm{q}_{2026}), \mathrm{s}_\emptyset)$ holds, the recorded execution is allowed by the policy. Furthermore, the policy contains no standalone duties and therefore $O(\pi_{\mathrm{remedy}}, \mathrm{s}_1)$ holds trivially. Consequently, every recorded execution is accounted for by the policy and

$$\mathsf{compliance}_p(\pi_{\mathrm{remedy}}, \mathrm{s}_1, \texttt{deny}) = \texttt{compliant}.$$

Next, let $\mathrm{s}_2 \in \mathrm{S}$ be a state of the world such that

$$\mathsf{performances}(\mathrm{s}_2) = \mathsf{performances}(\mathrm{s}_1) \cup \left\{ \left\langle \mathsf{execution}(\mathrm{q}_{2026}^{\mathrm{JP}}), \mathrm{s}_\emptyset \right\rangle \right\}.$$

The execution corresponding to q_{2026} remains accounted for by the policy since $A(\pi_{\mathrm{remedy}}, \mathsf{execution}(\mathrm{q}_{2026}), \mathrm{s}_\emptyset)$ still holds. Furthermore, π_{remedy} still contains no standalone duty rules and therefore $O(\pi_{\mathrm{remedy}}, \mathrm{s}_2)$ holds trivially. However, the additional execution corresponds to $\mathrm{q}_{2026}^{\mathrm{JP}}$, which violates the prohibition rule. $R(\pi_{\mathrm{remedy}}, \mathsf{execution}(\mathrm{q}_{2026}^{\mathrm{JP}}), \mathrm{s}_\emptyset)$ does not hold since the associated remedy is not compliant. Consequently, not every recorded execution is accounted for by the policy and

$$\mathsf{compliance}_p(\pi_{\mathrm{remedy}}, \mathrm{s}_2, \texttt{deny}) = \texttt{not_compliant}.$$

Let $\mathrm{s}_3 \in \mathrm{S}$ be a state of the world such that

$$\mathsf{performances}(\mathrm{s}_3) = \mathsf{performances}(\mathrm{s}_2) \cup \{\psi_{\mathrm{pay}}\},$$

where ψ_{pay} is the past contextualised execution defined in Example 5. To allow the compensation action to be performed, we further extend π_{remedy} with a permission rule $\gamma_{\text{pay}}^{\text{perm}}$[39], yielding the policy

$$\pi_{\text{repair}} = \left\langle \texttt{prohibit}, \{\gamma_{\text{time}}, \gamma_{\text{JP}}^{\text{remedy}}, \gamma_{\text{pay}}^{\text{perm}}\} \right\rangle,$$

where $\gamma_{\text{pay}}^{\text{perm}} = \langle \texttt{perm}, \text{execution}(\gamma_{\text{pay}}) \rangle$.

The execution corresponding to q_{2026} remains accounted for by the policy since $A(\pi_{\text{repair}}, \text{execution}(q_{2026}), s_\emptyset)$ still holds. Furthermore, the prohibition violation corresponding to q_{2026}^{JP} is now remedied and therefore $R(\pi_{\text{repair}}, \text{execution}(q_{2026}^{\text{JP}}), s_\emptyset)$ holds. The compensation execution ψ_{pay} is accounted for by the added permission rule $\gamma_{\text{pay}}^{\text{perm}}$, and thus $A(\pi_{\text{repair}}, \text{execution}(\gamma_{\text{pay}}), s_\emptyset)$ holds as well. Since π_{repair} contains no standalone duty rules, $O(\pi_{\text{repair}}, s_3)$ holds trivially. Consequently, every recorded execution is accounted for by the policy and

$$\text{compliance}_p(\pi_{\text{repair}}, s_3, \texttt{deny}) = \texttt{compliant}.$$

Finally, consider a policy containing a standalone duty:

$$\pi_{\text{duty}} = \left\langle \texttt{prohibit}, \{\gamma_{\text{time}}, \gamma_{\text{pay}}, \gamma_{\text{pay}}^{\text{perm}}\} \right\rangle.$$

Recall that s_1 contains only the contextualised execution corresponding to q_{2026}. Since $A(\pi_{\text{duty}}, \text{execution}(q_{2026}), s_\emptyset)$ holds, the recorded execution is accounted for by the policy. However, the standalone duty γ_{pay} has not been fulfilled and therefore $O(\pi_{\text{duty}}, s_1)$ does not hold. Consequently,

$$\text{compliance}_p(\pi_{\text{duty}}, s_1, \texttt{deny}) = \texttt{not_compliant}.$$

Now consider the state of the world $s_4 \in S$ such that

$$\text{performances}(s_4) = \text{performances}(s_1) \cup \{\psi_{\text{pay}}\}.$$

The execution corresponding to q_{2026} remains accounted for by the policy since $A(\pi_{\text{duty}}, \text{execution}(q_{2026}), s_\emptyset)$ still holds. Furthermore, the execution contained in ψ_{pay} is accounted for by the permission rule $\gamma_{\text{pay}}^{\text{perm}}$ and therefore $A(\pi_{\text{duty}}, \text{execution}(\gamma_{\text{pay}}), s_\emptyset)$ holds as well. Finally, the duty has now been fulfilled and therefore $O(\pi_{\text{duty}}, s_4)$ holds. Consequently,

$$\text{compliance}_p(\pi_{\text{duty}}, s_4, \texttt{deny}) = \texttt{compliant}.$$

[39] We explicitly add a permission for the compensation action. ODRL does not state that duties or remedies imply permission to perform their associated actions. Under the default-deny gap resolution strategy, omitting this permission would cause the compensation execution itself to evaluate to `not_allowed`, preventing policy compliance.

5 Applying the ODRL Formalism

The previous section introduced an operational formal semantics for ODRL evaluation that enables interoperable behaviour across ODRL evaluators. To define this process formally, we defined abstract representations for policies, evaluation requests, states of the world, and evaluation outcomes, together with functions that determine how these elements contribute to evaluation and compliance decisions.

However, interoperable evaluation requires not only common operational semantics, but also interoperable representations of evaluation inputs and outputs. Recent work has introduced RDF models for evaluation requests and states of the world as inputs to ODRL evaluation [11], and Compliance Reports as a representation of evaluation outcomes [10]. The operational formal semantics presented in the previous section complements these representations by defining how evaluation and compliance decisions are computed from those inputs. The remainder of this section explains how the models and the formalism fit together and how they jointly support interoperable ODRL evaluation.

5.1 Evaluation Request Model

Consider the evaluation request q_{2026} from Example 2, which asks whether Bob is permitted to stream Spiderpig on 15 May 2026. Formally, this request is represented as

$$q_{2026} = \langle \texttt{perm}, \langle \{Bob\}, \{stream\}, \{Spiderpig\}, \{\kappa_{2026}\}, \varnothing \rangle \rangle.$$

Using the Evaluation Request model proposed by Esteves et al. [11], the same request is represented in RDF as shown in Listing 5. Listing 5 represents the subject, action, and object of q_{2026} through the properties `sotw:evaluatedParty`, `sotw:evaluatedAction`, and `sotw:evaluatedTarget`, respectively. The temporal constraint is represented as a request parameter whose value is the timestamp `2026-05-15T12:00:00Z`.

```
ex:request a sotw:EvaluationRequest ;
    sotw:evaluatedParty ex:Bob ;
    sotw:evaluatedAction odrl:stream ;
    sotw:evaluatedTarget ex:Spiderpig .
    sotw:requestParameter [
        a sotw:RequestParameter ;
        sotw:value "2026-05-15T12:00:00Z"^^xsd:dateTime ;
        sotw:describesFeature sotw:TemporalData ;
    ] .
```

Listing 5 Turtle (RDF) serialization of the evaluation request q_{2026} using the Evaluation Request vocabulary.

While the Evaluation Request model provides an interoperable representation of the contextual information required during evaluation, it does not capture the complete structure of an atomic constraint from Definition 4. Request parameters only represent values and their associated features.

Consequently, the model directly supports value bindings equivalent to `odrl:eq`, but it does not represent the full set of atomic operators. Furthermore, evaluators require a mapping between request features and the selectors used by policy constraints. In Listing 5, the feature `sotw:TemporalData` must therefore be associated with the selector that retrieves values for the `odrl:dateTime` left operand. Such mappings are currently evaluator-specific, as no standard correspondence between Evaluation Request features and ODRL constraint operands has yet been defined.

5.2 Compliance Report Model

To illustrate how evaluation outcomes are represented, we evaluate q_{2026} against the policy introduced in Example 8

$$\pi_{\text{duty}} = \langle \texttt{prohibit}, \{\gamma_{\text{time}}, \gamma_{\text{pay}}, \gamma_{\text{pay}}^{\text{perm}}\}\rangle.$$

This policy consists of a time-restricted permission rule γ_{time}, an obligation γ_{pay}, and a corresponding permission rule $\gamma_{\text{pay}}^{\text{perm}}$.

Evaluation outcomes can be represented using the Compliance Report model introduced by Slabbinck et al. [10]. The model captures evaluation outcomes at different levels of abstraction. At the highest level, a `report:PolicyReport`[40] represents the outcome of evaluating an entire policy. A policy report consists of one or more `report:RuleReports`, each documenting the evaluation outcome of a single rule. In turn, a rule report may contain one or more `report:PremiseReports` that explain why the rule obtained its evaluation outcome.

Evaluating q_{2026} against π_{duty} requires evaluating the permission rule γ_{time}. Since γ_{time} is a permission rule without duties, its execution outcome depends on whether the rule is active. Recall from Section 4.2.2 that activation requires $\mathsf{party}(\varepsilon_q) \sqsubseteq \mathsf{party}(\varepsilon_\gamma), \mathsf{action}(\varepsilon_q) \sqsubseteq \mathsf{action}(\varepsilon_\gamma), \mathsf{asset}(\varepsilon_q) \sqsubseteq \mathsf{asset}(\varepsilon_\gamma)$, together with satisfaction of all rule constraints. For γ_{time} and q_{2026}, these relations hold. Furthermore, from Example 2, we know that

$$\mathsf{satisfy}(q_{2026}, \kappa_{\text{time}}) = \texttt{satisfied}.$$

Hence,

$$\mathsf{active}(\gamma_{\text{time}}, q_{2026}) = \texttt{active}.$$

With the Compliance Report model, we can record each of those outcomes. Each premise report in this example is `report:Satisfied`, more specifically the `report:PartyRReport`, `report:ActionReport`, `report:TargetReport`, and `report:ConstraintReport`. The rule-level outcome is represented through a `report:PermissionReport` whose `report:activationState` is `report:Active`. Finally, this rule report is included in a `report:PolicyReport`, which represents the outcome of evaluating the policy as a whole. Then, based on the evaluation request and policy, the full Compliance Report can be seen in Listing 6[41].

```
ex:policyReport a report:PolicyReport ;
    dcterms:created "2026-05-15T12:00:00ZZ"^^xsd:dateTime ;
    report:ruleReport ex:permissionReport .

ex:permissionReport a report:PermissionReport ;
    report:rule ex:permissionBobSpiderpig ;
    report:ruleRequest ex:requestPermission ;
    report:activationState report:Active ;
    report:premiseReport ex:partyReport, ex:actionReport,
        ex:targetReport, ex:constraintReport .

ex:partyReport a report:PartyReport ;
    report:satisfactionState report:Satisfied .

ex:actionReport a report:ActionReport ;
    report:satisfactionState report:Satisfied .

ex:targetReport a report:TargetReport ;
    report:satisfactionState report:Satisfied .

ex:constraintReport a report:ConstraintReport ;
    report:constraint ex:temporalConstraint ;
    report:satisfactionState report:Satisfied .
```

Listing 6 Turtle (RDF) serialization of the Compliance Report Model calculated by evaluating q_{2026} with π_{duty} ($\mathsf{evaluate}(\pi_{\text{duty}}, q_{2026}, s_\emptyset)$).

While the Compliance Report model provides interoperable representations for premise- and rule-level outcomes, it can not represent the policy-level outcomes defined by evaluate and $\mathsf{compliance}_p$. In particular, the vocabulary does not provide terms for representing `allowed`, `not_allowed`, `void`, `compliant`, and `not_compliant`. Furthermore, the model does not provide a mechanism for representing the conflict strategy applied during policy evaluation.

5.3 State of the World Model

Compliance evaluation additionally requires a state of the world. Recall from Definition 7 that a state of the world s contains a set of contextualised executions $\mathsf{performances}(s)$.

Consider the state of the world shown in Listing 7. In the formalism, it corresponds to a state of the world $s_{\text{history}} \in S$ such that

$$\mathsf{performances}(s_{\text{history}}) = \{\langle \mathsf{execution}(q_{2026}), s_\emptyset\rangle, \langle \mathsf{execution}(\gamma_{\text{pay}}^{\text{perm}}), s_\emptyset\rangle\}.$$

These performances indicate that the streaming execution represented by q_{2026} and the compensation action required by γ_{pay} have both occurred. Using the State of the World

[40] The namespace prefix `report:` refers to https://w3id.org/force/compliance-report#.

[41] For brevity, Listing 6 only includes the report corresponding to γ_{time}. $\mathsf{evaluate}(\pi_{\text{duty}}, q_{2026}, s_\emptyset)$ is determined by this active permission rule. The reports corresponding to γ_{pay} and $\gamma_{\text{pay}}^{\text{perm}}$ are omitted since they are inactive and therefore do not contribute to the evaluation outcome.

model introduced by Esteves et al. [11], such contextualised executions are represented through Compliance Reports linked to a `sotw:SotW` resource[42].

The State of the World model therefore provides an interoperable representation of the abstract state s_{history}. Through the performances recorded in $\text{performances}(s_{\text{history}})$, it provides the information required as input to both the rule-level compliance function (Function 8) and the policy-level compliance function (Function 11).

```
ex:sotw a sotw:SotW ;
    sotw:context ex:policyReport .

ex:policyReport a report:PolicyReport ;
    dcterms:created "2026-05-15T12:00:00ZZ"^^xsd:dateTime ;
    report:ruleReport ex:permissionReport ;
# permission report is shown in Listing 6
    report:ruleReport ex:dutyReport .

ex:dutyReport a report:DutyReport ;
    report:rule ex:paymentObligation ;
    report:ruleRequest ex:requestPermission ;
    report:activationState report:Active ;
    report:premiseReport ex:partyReport, ex:actionReportPay .

    ex:partyReport a report:PartyReport ;
    report:satisfactionState report:Satisfied .

ex:actionReportPay a report:ActionReport ;
    report:satisfactionState report:Unsatisfied .
```

Listing 7 Turtle (RDF) serialization of the State of the World corresponding to $\text{performances}(s_{\text{history}}) = \{\langle \text{execution}(q_{2026}), s_{\emptyset}\rangle, \langle \text{execution}(\gamma_{\text{pay}}^{\text{perm}}), s_{\emptyset}\rangle\}$, indicating that Bob streamed Spiderpig and paid the required compensation.

6 Discussion

The work presented in this paper provides an operational formal semantics for an ODRL Evaluator and demonstrates how this semantics can be applied using interoperable representations through Sections 4 and 5, respectively.

Section 4 addresses the behaviour expected from an ODRL Evaluator as described in the ODRL Information Model. As discussed in Section 3, the formalism does not cover profiles, `rightOperandReference`, composite rules, and compact policies. We provide formal definitions for ODRL policies, rules, and constraints, where the latter are explicitly distinguished as atomic and logical constraints.

In addition to the concepts defined by ODRL, the formalism introduces an abstraction called an execution, which captures the structure shared by ODRL rule types independently of their deontic type. Building on this abstraction, we define the concept of an Evaluation Request, which represents the intent to perform an execution and serves as the input to evaluation. We further introduce the concept of a State of the World, which captures execution history through contextualised executions represented as performances.

On top of these definitions, we introduce operational semantics for constraint satisfaction as well as evaluation and compliance at both the rule and policy levels. Policy evaluation determines whether an evaluation request is allowed under a given policy, thereby providing a basis for access-control decisions. In contrast, policy compliance determines whether a recorded execution history conforms to a policy, thereby providing a basis for monitoring. Moreover, since policies may contain rules that yield conflicting outcomes, the formalism incorporates the conflict properties defined by ODRL. Furthermore, policy evaluation may result in situations where no applicable rule establishes whether a requested execution is allowed or not, a situation referred to as a gap [56]. To address such situations, we introduce gap-resolution strategies that determine how these otherwise inconclusive outcomes are resolved into a definitive evaluation decision.

In Section 5, we demonstrate how the concepts introduced by the formal semantics can be represented using existing RDF models. We show how Evaluation Requests and States of the World can be represented using their respective models [11], and how the Compliance Report Model [10] can be used to represent evaluation outcomes at both the premise and rule levels. In particular, the latter captures premise-level outcomes for subjects, actions, objects, and constraints, as well as rule-level activation outcomes. These models align with the ongoing standardisation efforts of the ODRL Formal Semantics Community Group. At the time of writing, the Evaluation Request, State of the World, and Evaluation Report vocabularies are endorsed by the Community Group, where the Evaluation Report vocabulary builds upon the Compliance Report Model.

The remainder of this section examines the limitations of both the operational formal semantics and its application, compares the proposed approach with related work, and discusses the current status of implementation efforts.

6.1 Limitations

A limitation of the proposed semantics originates from ODRL itself. As previously observed by Fornara et al. [54], ODRL does not provide an explicit notion of duty deadlines or expiry times. In line with Bonatti et al. [56], we interpret that duties attached to permissions are required to be fulfilled before an evaluation request is allowed to be executed. Likewise, our semantics treats a duty as compliant only if the corresponding execution has occurred. In the absence of explicit duty deadlines, the absence of such an execution is interpreted as non-fulfilment. Otherwise, under the open-world assumption, a duty could indefinitely be considered

[42] For brevity, Listing 7 omits the serialisations of the underlying rules and evaluation requests.

fulfillable in the future, preventing a definitive compliance decision.

Regarding our work, the formalism only elaborates evaluation requests whose deontic type is permission and therefore answer the question "*Am I allowed to perform this execution?*". Questions such as "*I am not allowed to perform this execution, right?*" and "*Do I have to perform this execution?*" correspond to prohibition- and obligation-based evaluation requests, respectively. While the underlying request abstraction supports such requests, their semantics were not developed further.

Evaluation requests are further restricted to a concrete execution described through a set of atomic constraints. As a result, the formalism addresses decision problems for specific requests rather than exploratory questions over sets of possible requests. Queries such as "*Can Alice use asset X for research or education?*" or "*For which purposes would access be allowed?*" are therefore not evaluated directly. Such questions can nevertheless be answered through multiple concrete evaluation requests, suggesting a possible direction for future optimisation and extension.

Moreover, policy evaluation is defined for a single policy only. Situations in which multiple policies simultaneously apply to the same execution are therefore outside the scope of the current formalism. Extending the approach to sets of policies would require additional policy-combination semantics to determine how evaluation outcomes from different policies should be combined.

The formalism introduces a general framework for gap resolution and defines both allow and deny strategies. However, only the deny strategy is developed further and incorporated into the subsequent evaluation and compliance semantics. Extending the formal treatment to alternative gap-resolution strategies remains future work.

Finally, Section 5 demonstrates the alignment between the formalism and existing RDF vocabularies. However, it does not establish a complete formal mapping between RDF serialisations and the mathematical objects of the operational semantics. Completing such a mapping requires further extensions to the expressivity of the underlying RDF vocabularies, including a formal correspondence between `sotw:RequestParameters` and atomic constraints, and support for representing policy-level evaluation and compliance outcomes.

6.2 Comparison to Other ODRL Formal Semantics

Bonatti et al. [56] similarly formalise ODRL evaluation in terms of intended and historical behaviour. Their formalism represents historical behaviour through traces of states and intended behaviour through action instances occurring in a state. More specifically, a state $S = \langle t^S, E^S, (\cdot)^S \rangle$ contains the actions (E^S a set of events) occurring at time t^S together with an interpretation of the left operands at that point in time, while traces capture the history of such states. In contrast, our formalism introduces the explicit notions of Evaluation Requests and States of the World. Conceptually, the action instances contained in a state correspond to the intended or observed executions considered by our formalism, whereas traces fulfil a role comparable to the execution history captured through performances in a State of the World. They incorporate ODRL conflict properties and adopt a deny-by-default interpretation when no rule yields a definitive outcome, an approach that also underlies the deny gap-resolution strategy developed in this paper. However, their notion of rule activation is based solely on constraint satisfaction, whereas our activation semantics additionally incorporates the action, target, and subject components of a rule. Finally, Bonatti et al. define access-control decisions through policy compliance over extended traces, thereby establishing a close relationship between access control and monitoring, whereas our formalism introduces dedicated semantics for policy evaluation and policy compliance.

Salas et al. [57] model ODRL evaluation in terms of events and sets of events. An event captures an action together with its associated parties, assets, and constraint values, and therefore plays a role comparable to the executions introduced in our formalism. A State of the World is represented as a set of such events, over which policy validity is defined. While the terminology differs, validity serves a role comparable to policy compliance in our formalism, as both determine whether recorded behaviour conforms to a policy. They define policy validity through violations (Definition 1), where a policy is violated whenever an event fails to match an explicit permission rule. As a result, events that are neither explicitly permitted nor prohibited are treated as violations rather than inconclusive outcomes. This effectively yields an implicit deny-by-default gap-resolution strategy. In later work, Salas et al. [48] define access-control requests by constructing a new event from the requested action and adding it to the existing State of the World. Access is granted when the resulting state remains valid with respect to the policy. In contrast, our formalism introduces Evaluation Requests as a first-class concept and defines policy evaluation directly over an Evaluation Request, a State of the World, and a policy. Another notable difference is that the Salas formalism does not incorporate reasoning over action hierarchies and instead treats such reasoning as a separate materialisation step. Furthermore, policy validity is defined directly through rule matching and violations, without incorporating ODRL conflict properties into the semantics.

We limit the comparison to Bonatti et al. [56] and Salas et al. [48, 57], as both formalise the ODRL 2.2 Recommendation. Earlier efforts, such as Steyskal and Polleres [53], predate the standardisation of ODRL 2.2 and were devel-

oped for earlier versions of the language, making a direct comparison less meaningful.

7 Conclusion & Future Work

In this paper, we presented an operational formal semantics that provides a basis for consistent ODRL evaluation and compliance decisions. The semantics addresses two questions left open by ODRL 2.2: whether a requested execution is allowed under a policy, and whether a recorded execution history complies with that policy. To provide the information required to answer these questions, we formalised Evaluation Requests and States of the World as inputs alongside ODRL policies. Building on these inputs, we first formalised constraint satisfaction and then defined evaluation and compliance at both the rule and policy levels. The formalism further incorporates ODRL's conflict-resolution properties and introduces gap-resolution strategies for cases in which policy evaluation produces no conclusive decision. Furthermore, we aligned the formalism with RDF vocabularies endorsed by the ODRL Formal Semantics Community Group, including those for representing Evaluation Requests, States of the World, and evaluation outcomes. Finally, we compared our formalism with existing formal semantics for ODRL, identifying differences in how these approaches define evaluation and compliance.

The research presented in this paper identifies several directions for future work. First, the Compliance Report Model could be extended to represent complete policy-level evaluation and compliance decisions, allowing the resulting reports to be used as audit trails. Once the input and output models cover all concepts defined by the formalism, a complete bidirectional mapping between the RDF representations and their formal counterparts could be established. Together, these extensions would provide the basis for implementing an interoperable ODRL Evaluator. The semantics could also be extended with support for ODRL profiles, additional conflict- and gap-resolution strategies, as well as policy-combination semantics for cases in which multiple policies apply to the same execution. Finally, the results of this work could be integrated into the ongoing work of the ODRL Formal Semantics Community Group, contributing to a stable specification that reflects community consensus and can serve as input when defining the scope and charter of a future ODRL Working Group.

Acknowledgements This research was funded by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10) and by the imec.icon project PACSOI (HBC.2023.0752), which was co-financed by imec and VLAIO and brings together the following partners: FAQIR Foundation, FAQIR Institute, MoveUP, Byteflies, AContrario, and Ghent University – IDLab.

References

1. M. Verstraete, S. Verbrugge, D. Colle, M. Fierens, in *Proceedings 62nd International FITCE Congress," Services and Infrastructures Toward Smart Society*. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4575790
2. I. Jussen, J. Schweihoff, F. Möller, in *2023 IEEE 25th Conference on Business Informatics (CBI)*. pp. 1–10. DOI 10.1109/CBI58679.2023.10187530
3. C. Alcaraz, I.H. Meskini, J. Lopez, **24**(1), 17. DOI 10.1007/s10207-024-00912-1. URL https://doi.org/10.1007/s10207-024-00912-1
4. D. Mangan. Microsoft, OpenAI sued for copyright infringement by nonfiction book authors in class action claim. URL https://www.cnbc.com/2024/01/05/microsoft-openai-sued-over-copyright-infringement-by-authors.html
5. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance) (2016)
6. N.A.a.R.A. Office of the Federal Register. Public Law 105 - 304 - Digital Millennium Copyright Act. URL https://www.govinfo.gov/app/details/PLAW-105publ304/summary
7. M. Palmirani, F. Vitali, in *Legislative XML for the Semantic Web: Principles, Models, Standards for Document Management*, ed. by G. Sartor, M. Palmirani, E. Francesconi, M.A. Biasiotti, Law, Governance and Technology Series (Springer Netherlands, Dordrecht, 2011), pp. 75–100. DOI 10.1007/978-94-007-1887-6_6
8. R. Ianella, S. Villata. ODRL Information Model 2.2. URL https://www.w3.org/TR/odrl-model/
9. R. Ianella, M. Steidl, S. Myles, V. Rodríguez-Doncel. ODRL Vocabulary & Expression 2.2. https://www.w3.org/TR/odrl-vocab/ (2018)
10. W. Slabbinck, J. Rojas Meléndez, B. Esteves, P. Colpaert, R. Verborgh, in *The Semantic Web*, ed. by E. Curry, M. Acosta, M. Poveda-Villalón, M. van Erp, A. Ojo, K. Hose, C. Shimizu, P. Lisena (Springer Nature Switzerland, Cham, 2025), pp. 192–209. DOI 10.1007/978-3-031-94578-6_11
11. B. Esteves, W. Slabbinck, Y. Sellami, A. Cimmino, V. Rodríguez-Doncel, R. Verborgh, in *16th Workshop on Ontology Design and Patterns, Co-Located with ISWC 2025* (2025)
12. A. Cimmino, N. Fornara, in *Joint Proceedings of the ESWC 2025 Workshops and Tutorials*, *CEUR Workshop Proceedings*, vol. 3977, ed. by M. Sabou, A. Harth,

P. Lisena, E. Curry, B. Zhang, R. Alharbi, Y. He, G. Rehm, S. Schimmler, S. Dietze, N. Manola, A. Cimmino, N. Fornara, V. Rodríguez-Doncel, J. Domingue, A. Rettinger, D. Trilling, M. Grobelnik, C. d'Amato, V. Fionda, I. Tiddi, G. Tolomei (CEUR, Portorož, Slovenia, 2025), *CEUR Workshop Proceedings*, vol. 3977

13. M. De Vos, S. Kirrane, J. Padget, K. Satoh, in *Rules and Reasoning: Third International Joint Conference, RuleML+ RR 2019, Bolzano, Italy, September 16–19, 2019, Proceedings 3* (Springer), pp. 36–51. URL https://research.wu.ac.at/ws/portalfiles/portal/21761457/RuleML_RR_2019_Final.pdf
14. M.G. Kebede, G. Sileno, T. Van Engers, in *AI Approaches to the Complexity of Legal Systems XI-XII*, ed. by V. Rodríguez-Doncel, M. Palmirani, M. Araszkiewicz, P. Casanovas, U. Pagallo, G. Sartor (Springer International Publishing), pp. 48–61. DOI 10.1007/978-3-030-89811-3_4
15. I. Akaichi, W. Slabbinck, J.A. Rojas, C.V. Gheluwe, G. Bozzi, P. Colpaert, R. Verborgh, S. Kirrane, in *Proceedings of the Second International Workshop on Semantics in Dataspaces (SDS 2024)*, *CEUR Workshop Proceedings*, vol. 3705, ed. by J. Theissen-Lipp, P. Colpaert, S.K. Sowe, E. Curry, S. Decker (CEUR, Hersonissos, Greece, 2024), *CEUR Workshop Proceedings*, vol. 3705. URL https://ceur-ws.org/Vol-3705/paper10. ISSN: 1613-0073
16. S. Kirrane, A. Mileo, S. Decker, **8**(2), 311. DOI 10.3233/SW-160236. URL https://doi.org/10.3233/SW-160236
17. J. Park, R. Sandhu, in *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies* (Association for Computing Machinery), SACMAT '02, pp. 57–64. DOI 10.1145/507711.507722. URL https://doi.org/10.1145/507711.507722
18. J. Park, R. Sandhu, **7**(1), 128. DOI 10.1145/984334.984339. URL https://doi.org/10.1145/984334.984339
19. A. Pretschner, M. Hilty, F. Schütz, C. Schaefer, T. Walter, **6**(4), 44. DOI 10.1109/MSP.2008.101. URL http://ieeexplore.ieee.org/document/4588229/
20. I. Akaichi, S. Kirrane. Usage Control Specification, Enforcement, and Robustness: A Survey. DOI 10.48550/arXiv.2203.04800. URL http://arxiv.org/abs/2203.04800
21. L. Kagal, T. Finin, A. Joshi, in *The Semantic Web - ISWC 2003*, vol. 2870, ed. by D. Fensel, K. Sycara, J. Mylopoulos (Springer Berlin Heidelberg), pp. 402–418. DOI 10.1007/978-3-540-39718-2_26. URL http://link.springer.com/10.1007/978-3-540-39718-2_26
22. L. Kagal, T. Finin, M. Paolucci, N. Srinivasan, K. Sycara, G. Denker, **19**(4), 50. URL https://ieeexplore.ieee.org/abstract/document/1333035/
23. P. Bonatti, D. Olmedilla, in *Sixth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'05)*. pp. 14–23. DOI 10.1109/POLICY.2005.13. URL https://ieeexplore.ieee.org/abstract/document/1454299
24. P.A. Bonatti, S. Kirrane, I.M. Petrova, L. Sauro, E. Schlehahn. The SPECIAL Usage Policy Language. URL https://ai.wu.ac.at/policies/policylanguage/
25. I. Akaichi, G. Flouris, I. Fundulaki, S. Kirrane, in *Rules and Reasoning*, vol. 14244, ed. by A. Fensel, A. Ozaki, D. Roman, A. Soylu (Springer Nature Switzerland, Cham, 2023), pp. 34–53. DOI 10.1007/978-3-031-45072-3_3
26. I. Akaichi, G. Flouris, I. Fundulaki, S. Kirrane, in *Proceedings of the ISWC 2024 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice*, *CEUR Workshop Proceedings*, vol. 3828, ed. by L. Etcheverry, V.L. Garcia, F. Osborne, R. Pernisch (CEUR), *CEUR Workshop Proceedings*, vol. 3828. URL https://ceur-ws.org/Vol-3828/#ISWC2024_paper_21
27. H.J. Pandit, B. Esteves, G.P. Krog, P. Ryan, D. Golpayegani, J. Flake. Data Privacy Vocabulary (DPV) – Version 2. DOI 10.48550/arXiv.2404.13426. URL http://arxiv.org/abs/2404.13426
28. B. Esteves, H.J. Pandit, in *Proceedings of the Fourth International Workshop on Semantics in Dataspaces (SDS 2026)* (EasyChair). URL https://beatrizesteves.org/assets/documents/2026/SDS/dpv-odrl.pdf
29. A. Eitel, C. Jung, R. Brandstädter, A. Hosseinzadeh, S. Bader, C. Kühnle, P. Birnstill, G. Brost, M. Gall, F. Bruckner, et al., Aufl. IDS Association, Berlin (2021)
30. B. Otto, M. Ten Hompel, S. Wrobel (eds.), *Designing Data Spaces: The Ecosystem Approach to Competitive Advantage* (Springer International Publishing). DOI 10.1007/978-3-030-93975-5. URL https://link.springer.com/10.1007/978-3-030-93975-5
31. Smart Open-Source Middleware (SIMPL). https://simpl-programme.ec.europa.eu/ (2026)
32. T.J. Berners-Lee. Information management: A proposal. URL https://cds.cern.ch/record/369245/files/dd-89-001
33. N. Rump, IEEE Signal Processing Magazine **21**(2), 63 (2004). DOI 10.1109/MSP.2004.1276114
34. D.L. Burk, J.E. Cohen, Harvard Journal of Law & Technology **15**(1) (2001)
35. World Wide Web Consortium (W3C). URL https://www.w3.org/
36. R. Ianella. ODRL: A Path Well Travelled (2018)

37. O. Lassila, R.R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. URL https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/
38. G. Schreiber, Y. Raimond. RDF 1.1 Primer. URL https://www.w3.org/TR/rdf11-primer/
39. M.J. Dürst, M. Suignard. Internationalized Resource Identifiers (IRIs). DOI 10.17487/RFC3987. URL https://datatracker.ietf.org/doc/rfc3987
40. V. Rodríguez-Doncel, N. Roman, in *Joint Proceedings of the ESWC 2025 Workshops and Tutorials*, *CEUR Workshop Proceedings*, vol. 3977, ed. by M. Sabou, A. Harth, P. Lisena, E. Curry, B. Zhang, R. Alharbi, Y. He, G. Rehm, S. Schimmler, S. Dietze, N. Manola, A. Cimmino, N. Fornara, V. Rodríguez-Doncel, J. Domingue, A. Rettinger, D. Trilling, M. Grobelnik, C. d'Amato, V. Fionda, I. Tiddi, G. Tolomei (CEUR, Portorož, Slovenia, 2025), *CEUR Workshop Proceedings*, vol. 3977
41. COMMUNICATION FROM THE COMMISSION TO THE EUROPEAN PARLIAMENT AND THE COUNCIL DATA UNION STRATEGY UNLOCKING DATA FOR AI. URL https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=COM:2025:835:FIN
42. W. Slabbinck, J.A. Rojas, B. Esteves, R. Verborgh, P. Colpaert, in *Proceedings of the Posters and Privacy Session of the Solid Symposium 2024*, *CEUR Workshop Proceedings*, vol. 3947, ed. by B. Esteves, J. Hofmann, S. Schmid (CEUR, Leuven, Belgium, 2024), *CEUR Workshop Proceedings*, vol. 3947, pp. 109–117. URL https://ceur-ws.org/Vol-3947/short15. ISSN: 1613-0073
43. A. Cahyono Adi, U. Isnaini, M. Hanif, K. Kurniawan, in *Proceedings of the Second Workshop on ODRL and Beyond: Practical Applications and Challenges for Policy-Base Access and Usage Control.*
44. W. Slabbinck, R. Dedecker, W. Termont, B. Esteves, P. Colpaert, R. Verborgh, (Leiden, Netherlands, 2025)
45. P. Hochstenbach, B. Esteves, R. Verborgh, in *Joint Proceedings of the ESWC 2025 Workshops and Tutorials*, *CEUR Workshop Proceedings*, vol. 3977, ed. by M. Sabou, A. Harth, P. Lisena, E. Curry, B. Zhang, R. Alharbi, Y. He, G. Rehm, S. Schimmler, S. Dietze, N. Manola, A. Cimmino, N. Fornara, V. Rodríguez-Doncel, J. Domingue, A. Rettinger, D. Trilling, M. Grobelnik, C. d'Amato, V. Fionda, I. Tiddi, G. Tolomei (CEUR, Portorož, Slovenia, 2025), *CEUR Workshop Proceedings*, vol. 3977
46. A. Cimmino, J. Cano-Benito, R. García-Castro, Comput. Secur. **150**(C) (2025). DOI 10.1016/j.cose.2024.104282. URL https://doi.org/10.1016/j.cose.2024.104282
47. A. Hosseinzadeh, A. Eitel, C. Jung, in *Proceedings of the 6th International Conference on Information Systems Security and Privacy* (SCITEPRESS - Science and Technology Publications), pp. 397–405. DOI 10.5220/0008936003970405. URL http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0008936003970405
48. J.O. Salas, P. Pareti, A. Aslam, C. Maidens, G. Konstantinidis, (2026). DOI 10.48550/arXiv.2607.15987. URL http://arxiv.org/abs/2607.15987
49. W. Termont, W. Slabbinck, J.V. Herwegen, P. Hochstenbach, B. Esteves, in *ODRL and beyond: practical applications and challenges for policy-based access and usage control. OPAL 2026 Co-located with the Extended Semantic Web Conference* (2026)
50. R. García, R. Gil, I. Gallego, J. Delgado,
51. A. Kasten, R. Grimm, pp. 77–91
52. R. Pucella, V. Weissman. A Formal Foundation for ODRL. DOI 10.48550/arXiv.cs/0601085. URL http://arxiv.org/abs/cs/0601085
53. S. Steyskal, A. Polleres, in *Rule Technologies: Foundations, Tools, and Applications*, ed. by N. Bassiliades, G. Gottlob, F. Sadri, A. Paschke, D. Roman (Springer International Publishing, Cham, 2015), pp. 360–375. DOI 10.1007/978-3-319-21542-6_23. URL https://ai.wu.ac.at/~polleres/publications/stey-poll-2015RuleML.pdf
54. N. Fornara, M. Colombetti, **32**(4), 319. URL https://content.iospress.com/articles/ai-communications/aic190617
55. M. Kieffer, P. Serrano-Alvarado, M. Bernelin, in *Joint Proceedings of the ESWC 2025 Workshops and Tutorials co-located with 22nd Extended Semantic Web Conference (ESWC 2025), Portorož, Slovenia, June 1-2, 2025* (2025)
56. P.A. Bonatti, N. Fornara, A. Harth, in *Joint Proceedings of the ESWC 2025 Workshops and Tutorials*, *CEUR Workshop Proceedings*, vol. 3977, ed. by M. Sabou, A. Harth, P. Lisena, E. Curry, B. Zhang, R. Alharbi, Y. He, G. Rehm, S. Schimmler, S. Dietze, N. Manola, A. Cimmino, N. Fornara, V. Rodríguez-Doncel, J. Domingue, A. Rettinger, D. Trilling, M. Grobelnik, C. d'Amato, V. Fionda, I. Tiddi, G. Tolomei (CEUR, Portorož, Slovenia, 2025), *CEUR Workshop Proceedings*, vol. 3977
57. J.O. Salas, P. Pareti, S. Yumuşak, S. Gheisari, L.D. Ibáñez, G. Konstantinidis. Evaluation and Comparison Semantics for ODRL (2025). DOI 10.48550/arXiv.2509.05139
58. E.I. Papagiannakopoulou, N.L. Dellas, G.V. Lioudakis, M.N. Koukovini, A. Mousas, in *Joint Proceedings of Posters, Demos, Workshops, and Tutorials of the 21st International Conference on Semantic Systems*, *CEUR*

Workshop Proceedings, vol. 4064, ed. by D. Chaves-Fraga, I. Heibi, D. Garijo, D. Collarana, A. Salatino, S. Vahdati (CEUR, Vienna, Austria, 2025), *CEUR Workshop Proceedings*, vol. 4064

59. D.M. Mustafa, C. Lange, G. Guizzardi, D. Collarana, C. Quix, S. Decker, (arXiv). DOI 10.48550/arXiv.2606.24344. URL http://arxiv.org/abs/2606.24344
60. G.H. Von Wright, **60**(237), 1. URL https://www.jstor.org/stable/2251395?casa_token=pk73w1roQb0AAAAA:w66imatjp2b0-fSI3Js5NyjhxQUVhjG70MOoXcYz2x8FIljcd-8R9dheNPfFRdbL0ELvr1jDMTp9ij_-BTYUYPnmcoDsBSouRaHYm8gOggWfc9Ue_JMK_g
61. W. Slabbinck, J. Rojas Meléndez, B. Esteves, R. Verborgh, P. Colpaert, in *Joint Proceedings of Posters, Demos, Workshops, and Tutorials of the 21st International Conference on Semantic Systems*, *CEUR Workshop Proceedings*, vol. 4064, ed. by D. Chaves-Fraga, I. Heibi, D. Garijo, D. Collarana, A. Salatino, S. Vahdati (CEUR, Vienna, Austria, 2025), *CEUR Workshop Proceedings*, vol. 4064
62. H. J. Pandit, B. Esteves, G. P. Krog, P. Ryan, D. Golpayegani, J. Flake, in *The Semantic Web – ISWC 2024*, vol. 15233, ed. by G. Demartini, K. Hose, M. Acosta, M. Palmonari, G. Cheng, H. Skaf-Molli, N. Ferranti, D. Hernández, A. Hogan (Springer Nature Switzerland, Cham, 2025), pp. 171–193. DOI 10.1007/978-3-031-77847-6_10
63. B.W. Lampson, **8**(1), 18. DOI 10.1145/775265.775268. URL https://dl.acm.org/doi/10.1145/775265.775268
64. eXtensible Access Control Markup Language (XACML) v3.0. URL https://www.oasis-open.org/standard/xacmlv3-0/