Automated Validation of ODRL Policies for Usage Control and Data Spaces

Elena Molino-Peña[1], Wout Slabbinck[2], José María García[1], Antonio Ruiz-Cortés[1] and Beatriz Esteves[2]

[1]Smart Computer Systems Research and Engineering Laboratory (SCORE), Universidad de Sevilla, Seville, Spain
[2]IDLab, Department Electronics and Information Systems, Ghent University – imec, Ghent, Belgium

Abstract

As the adoption of trustworthy policy mechanisms grows across domains such as legal compliance, digital rights management and data governance, ensuring that exchanged policies conform to standard specifications becomes a prerequisite for reliable interoperability. One such standard is the Open Digital Rights Language (ODRL), a W3C Recommendation for expressing policies governing the use of digital assets. ODRL has been widely adopted as a policy language for usage control enforcement, including in the context of European Data Spaces. However, ensuring the validity of ODRL policies remains an issue, as RDF validity does not imply that a policy is ODRL conformant, and ODRL conformance does not imply that a policy is semantically consistent. To address this need, we present an open-source ODRL validator that, in three stages, normalizes a policy into an atomic form, validates its conformance to the ODRL model, and detects deontic inconsistencies between its rules. The validator is available as a web demonstrator and as npm packages that can be integrated into other applications, making ODRL validation accessible to both developers and end users.

Keywords

ODRL, Policy Validation, Deontic Conflicts, SHACL, N3 rules

1. Introduction

Digital resources have become the foundation of today's data-driven economy, powering applications from artificial intelligence and digital public services to scientific research and industrial innovation. As organizations increasingly exchange data across institutional and national boundaries, governing not only who can access resources but how they may be used after access has been granted has become a critical challenge. Usage control lets data providers specify permissions, prohibitions, and obligations that accompany said data, supporting legal compliance, intellectual property protection, and responsible data reuse [1]. This need is particularly evident in the European Union's vision for common data spaces, which aim to facilitate the secure sharing of personal and non-personal data across sectors. To enable participants to share data with confidence, usage control has emerged as one of the fundamental building blocks [2], requiring policies to be expressed in a standardized, interoperable, and machine-processable manner so that all parties can interpret and enforce them consistently.

The Open Digital Rights Language (ODRL) has become one of the main standards for expressing such policies, offering a common structure and vocabulary to describe permissions, prohibitions and duties over digital assets [3, 4]. ODRL has been adopted in data spaces [5] and usage control frameworks, such as UMA [6], and is increasingly supported by tools, including ODRL evaluators that enforce policies [7, 8], automated approaches based on Large Language Models (LLMs) to generate ODRL policies [9, 10], policy composition [11], and solutions to check compliance with regulations [12]. All these tools depend on the policies being valid. For instance, a validator is needed to catch the errors, hallucinations, or inconsistencies that LLM-based generators may introduce.

3rd NeXt-generation Data Governance workshop (NXDG 2026), co-located with the 22nd SEMANTiCS conference, 15 September 2026, Ghent, Belgium

mmolino@us.es (E. Molino-Peña); wout.slabbinck@ugent.be (W. Slabbinck); josemgarcia@us.es (J. M. García); aruiz@us.es (A. Ruiz-Cortés); beatriz.esteves@ugent.be (B. Esteves)
0000-0001-7024-5300 (E. Molino-Peña); 0000-0002-3287-7312 (W. Slabbinck); 0000-0002-0303-2740 (J. M. García); 0000-0001-9827-1834 (A. Ruiz-Cortés); 0000-0003-0259-7560 (B. Esteves)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-13

CEUR-WS.org/Vol-4327/paper6.pdf

To the best of our knowledge, no tool exists to check whether an ODRL policy is valid, reporting structural and cardinality errors, flagging misuse of the ODRL vocabulary or detecting inconsistencies. The validation requirements are dispersed throughout the specification, expressed mainly in natural language, so checking a policy manually requires expert knowledge and is error-prone. In this paper, we present an open-source ODRL validator to automatically validate ODRL policies at the syntactic and semantic level. The main contributions of this work are the following: *(i)* a catalogue of ODRL validity requirements, encoded as SHACL shapes and N3 rules, covering conformance to the ODRL model specification and the detection of deontic inconsistencies between rules; *(ii)* a normalization process that rewrites any policy in an equivalent atomic form; *(iii)* an ODRL Validator available as a web interface, where a policy can be checked, returning a validation report explaining any violations, warnings, and conflicts; and *(iv)* two npm packages that encapsulate the normalizer and the validator, such that it can be integrated directly into other software projects.

The remainder of the article is organized as follows. Section 2 presents the ODRL model and existing ODRL validators and evaluators. Section 3 introduces the syntactic and semantic requirements for a valid ODRL policy. Section 4 describes the system implemented to automatically validate ODRL policies, and finally Section 5 concludes the article.

2. Background and Related Work

2.1. The Open Digital Rights Language (ODRL)

ODRL is a W3C Recommendation for expressing policies over the use of digital content and services, providing an interoperable modeling language and a common vocabulary specified in the ODRL 2.2 Information Model [3] and Vocabulary and Expression [4].

Figure 1 illustrates the core model classes, the relationships between them and their cardinalities. An ODRL policy, of type set, offer or agreement, is a non-empty composition of one or more rules, which can be permissions, prohibitions, or duties. A Permission allows an action to be exercised on an asset. It can be aggregated with one or more duties acting as preconditions, such as prepayment or prior notice. A Prohibition forbids an action from being exercised on an asset. It can be associated with one or more duties defining corrective actions as remedies to compensate for non-compliance. A Duty expresses an obligation to perform an action. It can be associated with other duties that act as consequences when a prior obligation is not fulfilled. Thus, each rule links three core elements: an action, the asset it applies to, and the parties involved. Rules, assets, actions and parties can be further refined through constraints, which either impose restrictions or narrow the semantics of a property. Typical constraints express temporal, spatial or financial conditions that must be satisfied for the rule to take effect.

The model imposes strict cardinalities on rules and policy types (e.g., a permission in an agreement must have an associated asset, assigner, assignee and action), which can lead to considerable repetition. In order to reduce it, the ODRL model proposes two mechanisms for defining more concise policies[1]: rule composition and compact policies, which respectively extend rule-level cardinalities and elevate properties to the policy level. Both can be combined without altering the meaning of the policy.

ODRL also supports extensibility through its profile mechanism, enabling specific domains to incorporate additional concepts that go beyond the baseline, which offers a large set of predefined terms. An ODRL profile can define not only new vocabulary terms but also new types of policies, rules and new objects. As a result, there are many different ways of defining a policy using ODRL, and this wide range of possibilities makes it hard to determine manually whether a given policy is valid.

2.2. ODRL in Practice

Although policy evaluation was originally outside the scope of ODRL [13], several initiatives have adopted ODRL policies for usage control enforcement, including data space architectures [14, 5] and the User-Managed Access (UMA) implementation of Slabbinck et al. [6]. Since enforcement decisions

[1]ODRL IM section 2.7 about Policy Rule Composition: https://www.w3.org/TR/odrl-model/#composition

depend on well-formed policies, we organise the related work on ODRL tooling, guided by the ODRL Landscape[2], into two categories: **ODRL Validators** and **ODRL Evaluators**.

ODRL Validators is a term defined in ODRL that states *"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."* [3]. In essence, that means these systems verify that ODRL policies conform to the model by checking their structure, cardinalities, and datatypes. ODRL validation is used both in dedicated tooling and in recent research on ODRL policy generation. One of the official ODRL implementations[3], odrlapi[4], performs ODRL validation using SHACL shapes [15], which is the standard for doing syntactical RDF validation. To demonstrate its validation capabilities, the project provides 83 examples[5] of ODRL policies, 55 valid and 28 invalid. However, this validator does not address inconsistencies between rules, cardinality constraints at the rule level, or type checking. Mustafa et al. [10] and Zhong et al. [9] also applied SHACL to validate LLM-generated ODRL policies, though limited to syntactic checks[6]. While these approaches rely on SHACL for ODRL validation, the ODRL Information Model additionally recommends several normalization steps[7] prior to conformance checking in Section 2.7. The reviewed papers do not discuss these steps, and odrlapi does not document them.

ODRL Evaluators are defined by the ODRL Information Model as *"a system that determines whether the Rules of an ODRL Policy expression have meet their intended action performance"* [3], that is, systems that decide whether a given action should be granted or whether a past decision complied with the policy. Existing evaluators, such as the ODRL Evaluator by Slabbinck et al. [8], based on the ODRL Formal Semantics, and the ODRE framework by Cimmino et al. [7], assume policies that are already well-formed, which is precisely what our validator provides.

3. Validation Requirements

An ODRL validator checks the conformance of an ODRL policy against the validation requirements of the model [3], introduced in Section 2. However, a policy that conforms to the model may still contain inconsistencies between its rules and constraints, meaning that they contradict each other. For example, a policy may permit and prohibit the same action for the same assignee on the same asset, or two constraints within the same rule may share the same left operand and the equality operator but point to different right operands, which cannot be satisfied in both cases. The model provides the `odrl:conflict` property to declare a resolution strategy (`perm`, `prohibit` or `invalid`) that resolves such conflicts at evaluation time. A validator, in contrast, should detect and report such inconsistencies before evaluation, independently of the resolution strategy, so that they can be inspected. In this section, we describe how we collected the corresponding validation requirements and why a policy must first be normalized to an atomic structure.

To establish the set of requirements for the validator, we relied on the ODRL 2.2 Information Model [3] and the validation wiki of the W3C POE Working Group[8], which specifies a list of normalization and validation operations. In the model, the requirements are defined in natural language throughout the document. Although it includes a model diagram, the cardinalities are not specified in it. We captured them in the UML class diagram shown in Figure 1. The list of operations in the wiki does not fully cover the model, since it does not capture all cardinalities, such as requiring that every rule has an associated action, nor aspects like upper bounds, class typing, the use of vocabulary instances or structures that require refinement, among others. We used both sources and extended the catalogue of requirements.

[2]ODRL Landscape: https://github.com/w3c/odrl/tree/master/landscape
[3]Implementation Report of the Permissions & Obligations Expression Working Group (WG), the WG that standardized ODRL: https://w3c.github.io/poe/test/implementors
[4]Web interface of odrlapi: https://odrlapi.appspot.com/
[5]ODRL examples used in odrlapi: https://github.com/oeg-upm/licensius/tree/master/odrlapi/src/main/webapp/samples
[6]LLM4ODRL github repository: https://github.com/Daham-Mustaf/LLM4ODRL
[7]The POE WG lists these steps in its validation wiki: https://www.w3.org/2016/poe/wiki/Validation
[8]Validation wiki of the W3C POE Working Group: https://www.w3.org/2016/poe/wiki/Validation

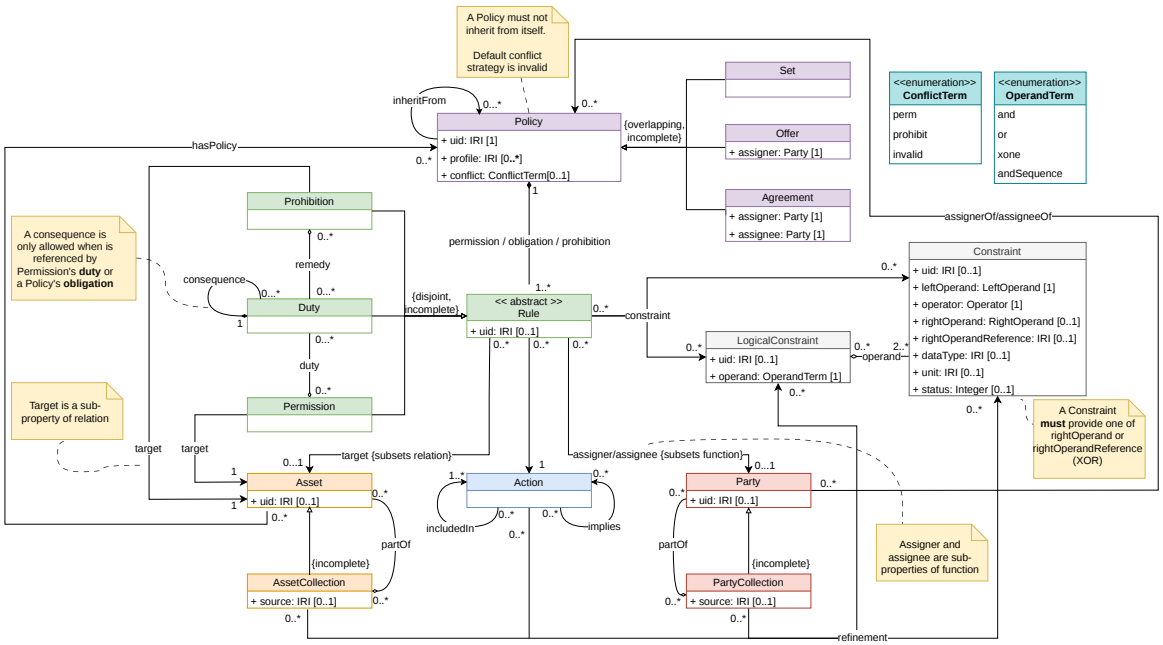


Figure 1: ODRL 2.2 information model including hierarchies and cardinality constraints.

Since ODRL policies can be expressed in different ways, as explained in Section 2.1, consistent validation requires that cardinalities be checked on atomic policy. Therefore, the normalization process is necessary before checking the structure, cardinalities, value types, and inconsistencies between rules.

3.1. Normalization Operations

The normalization process performs a sequence of transformations to rewrite any policy into an irreducible atomic form. Of the normalization operations defined by the W3C POE Working Group, we cover the three most frequent and relevant for structural validation: external references, compact policies, and rule composition. These transformations are applied in this order since each one builds on the previous. The remaining normalization operations, such as type inference and inheritance resolution, are less frequent and are left for future work. **N1 – internalization of external references,** converts inverse properties into direct properties within the rules: `assigneeOf` becomes `assignee`, `assignerOf` becomes `assigner`, and `hasPolicy` becomes `target` in the corresponding rule. **N4 – interiorization of policy-level properties,** moves properties declared at the policy level, such as assigner or action, into each individual rule. Finally, **N5 – expansion of compound rules,** expands rules that reference multiple assets, actions or parties into atomic rules, each containing a single element.

(a) Compact policy and composite rule

```
<#policy1> a odrl:Policy ;
   odrl:target <#resourceX> ;
   odrl:permission <#rule1> .

<#rule1> a odrl:Permission ;
   odrl:assignee <#alice> ;
   odrl:action odrl:read, odrl:modify .
```

(b) Normalized policy

```
<#policy1> a odrl:Policy ;
    odrl:permission <#rule1>, <#rule2> .

<#rule1> a odrl:Permission ;
    odrl:assignee <#alice> ;
    odrl:action odrl:read ;
    odrl:target <#resourceX> .

<#rule2> a odrl:Permission ;
    odrl:assignee <#alice> ;
    odrl:action odrl:modify ;
    odrl:target <#resourceX> .
```

Figure 2: Normalization of a compact policy and composite rule into its atomic form.

Figure 2 shows an example of the normalization process for a policy allowing Alice to read and modify `resourceX`. On the left, `resourceX` is expressed at the policy level and the two actions are combined in a single compound rule. On the right, the policy is expanded into two atomic rules. A detailed description of the normalization process, including examples, is available in our documentation[9].

3.2. Model Conformance Requirements

The ODRL validation process verifies that an atomic policy is well-formed according to the ODRL model. For each element, it must satisfy the properties and cardinalities. The requirements are organized by the classes of the model. At **policy** level, a policy must be identified by an IRI and contain at least one rule. Properties such as `profile`, `conflict` or `inheritFrom` are checked for cardinality and value type, and the rules of an Offer or an Agreement must declare the required parties. Each **rule** must have one action and at most one target, assignee and assigner, each conforming to its class. Permissions and prohibitions require a target, and any associated duties must conform to the Duty requirements. A **constraint** must have one left operand, one operator, and either a right operand or a right operand reference. Logical constraints must contain a logical operator. Finally, **party** and **asset** collections may declare at most one source and their refinements must be valid constraints.

```
<#policy1> a odrl:Policy ;
   odrl:permission <#rule1> .

<#rule1> a odrl:Permission ;
   odrl:assignee <#alice> ;
   odrl:target <#resourceX> .
```

Figure 3: Permission without an action, violating the requirement R11 that every rule must have one action.

Figure 3 shows a policy that defines a permission without an action. Since an action is a mandatory property for every rule (requirement R11), this policy is to be considered invalid. The complete catalogue of 52 model conformance requirements, based on the validation operations of the W3C POE wiki and extending them with specification restrictions, is available in our documentation[10].

3.3. Inconsistency Detection Requirements

The inconsistency detection process checks whether the policy contains contradictions among its rules or constraints. For instance, Figure 4 shows a policy that both permits and prohibits Alice modifying the same asset — these two rules produce a deontic inconsistency.

```
<#policy1> a odrl:Policy ;
   odrl:permission <#rule1> ;
   odrl:prohibition <#rule2> .

<#rule1> a odrl:Permission ;
   odrl:assignee <#alice> ;
   odrl:action odrl:modify ;
   odrl:target <#resourceX> .

<#rule2> a odrl:Prohibition ;
   odrl:assignee <#alice> ;
   odrl:action odrl:modify ;
   odrl:target <#resourceX> .
```

Figure 4: Permission and prohibition over the same assignee, action and target.

[9]ODRL Validator normalization documentation: https://github.com/odrl-force/odrl-validator/blob/main/docs/Atomization.md
[10]ODRL Validator requirements: https://github.com/odrl-force/odrl-validator/blob/main/docs/Requirements.md

We detect three types of inconsistency. An **obligation–prohibition** conflict arises when an obligation and a prohibition refer to the same assignee, action and asset, since an action cannot be both obligatory and forbidden. A **permission–prohibition** conflict is contradictory in the same way, as permission and prohibition are the negation of each other. We report it as a potential conflict rather than an error, since a policy may intentionally combine both and rely on the `odrl:conflict` strategy to resolve them. Finally, an **unsatisfiable constraint** occurs within a single rule when two constraints share the same left operand and equality operator but require different right operands, so the rule never applies.

4. Implementation

The requirements described in Section 3 have been implemented as an ODRL validator system that runs in the browser[11]. Users can paste an ODRL policy and obtain the validation report. Since it runs on the client side, policies are never sent to a server or stored. In addition, the validator[12] and the normalization[13] components are also distributed as open-source npm packages[14], so they can be integrated into other applications. Finally, we provide a video walk-through of the ODRL Validator in the browser to elaborate how it works and how it can be used[15].

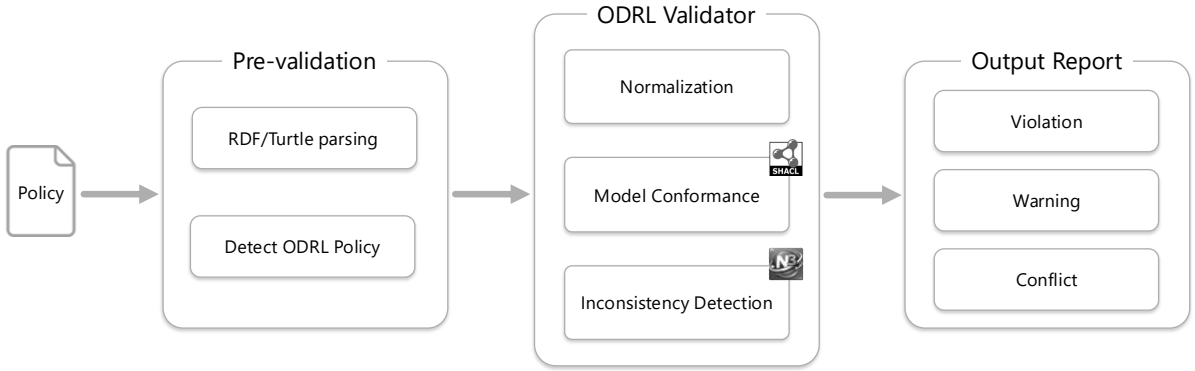


Figure 5: ODRL validator pipeline.

Figure 5 illustrates the stages the system performs when validating a policy, which consists of a pre-validation and a validation phase. Pre-validation checks that the input is valid RDF and contains ODRL policies. If either check fails, the process stops and the error is reported. The validation phase consists of three steps. First, the ODRL policy is normalized to its atomic form. Second, the atomic policy is checked against the SHACL shapes that encode the model conformance requirements, which detect structural and cardinality problems, missing type information, and incorrect use of the ODRL vocabulary. Finally, if the policy conforms to the model, N3 inconsistency detection rules are applied, identifying deontic conflicts between rules and contradictory constraints. The results of these steps are combined into a single validation report, divided into three categories based on their severity level (violation, warning, or conflict), and include an explanatory message of the detected issues.

Table 1 groups the validation results into the three categories of severity. When the policy does not conform to the ODRL Information Model, the results are categorized as violations. When the policy can still be interpreted but it relies on constructs that are incomplete or omitted and must be inferred, such as terms used outside of the ODRL vocabulary, they are classified as warnings. Finally, when the policy contains rules or other constructs that contradict one another, they are classified as conflicts.

[11]ODRL Validator demonstrator: https://w3id.org/force/validator/demo
[12]ODRL Validator npm package: https://www.npmjs.com/package/odrl-validator
[13]ODRL Atomizer npm package:https://w3id.org/force/atomizer/npm
[14]Source code available at https://w3id.org/force/validator
[15]ODRL Validator demonstration: https://zenodo.org/records/21259443

Table 1
Validation categories with their mechanism and severity level.

Validation Category	Checks	Mechanism	Severity
Structure and cardinality	Required properties and cardinalities	SHACL	Violation
Type and vocabulary	Value types and ODRL vocabulary	SHACL	Warning
Consistency between rules	Contradictions in rules and constraints	N3 rules	Conflict

The system is implemented using TypeScript, including the pre-validation phase and the normalization step of the validation stage. For model conformance, SHACL [15] was chosen because it is a W3C standard for validating RDF graphs. With SHACL, we can define mandatory properties, strict cardinalities, structural requirements and specific instances. Moreover, its severity mechanism allows each shape to be configured to report the issues as violations or warnings, and each failed check is followed by a machine-readable message. The shapes are divided into modules, available in the GitHub repository[16]. However, detecting inconsistencies between rules requires reasoning over the semantics of the policy rather than validating its structure. For that reason, we use Notation3 (N3) [16], an RDF-based language that provides constructs for expressing rules and performing reasoning over knowledge graphs. The consistency requirements are encoded as N3 rules[17] and executed with the eyeling reasoner [17], which infers the conflicts present in the policy.

5. Conclusions and Future Work

In this paper we presented an open-source validator for ODRL policies that checks their conformance to the ODRL 2.2 model and reports inconsistencies between their rules, applying a normalization step so that policies expressed in different ways can be validated uniformly. The validator is available as an online platform and as npm packages that can be integrated into other systems.

As future work, we plan to make the validation more complete by supporting ODRL profiles, inferring additional restrictions from the vocabulary such as incompatibilities between operands and operators, and broadening the inconsistency checks. Then, to assess the validator, we intend to build a test suite of ODRL policies with positive and negative examples, and to compare our results against *odrlapi*, reporting coverage and execution times. A further direction is to move from detecting inconsistencies to resolving them according to a chosen strategy. Finally, we aim to integrate the validator as a preliminary step in ODRL pipelines, so that policies are validated before they are passed to evaluators.

Acknowledgments

This work was supported by the R&D projects PID2021-126227NB-C21 and PID2021-126227NB-C22 funded by MICIU/AEI/10.13039/501100011033/ERDF/EU; and PID2024-155693NB-C44, PDC2025-166101-C21 and PDC2025-166101-C22 funded by MICIU/AEI/10.13039/501100011033/FEDER, EU; and DGP_PIDI_2024_00918 funded by Consejería de Universidad, Investigación e Innovación, Junta de Andalucía. We would like to thank Jos De Roo for his support regarding eyeling and Patrick Hochstenbach and Ruben Dedecker for their valuable feedback on the Web demonstrator.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT to improve grammar and spelling. The authors reviewed and edited the text and are fully responsible for its final content.

[16]SHACL shape modules: https://github.com/odrl-force/odrl-validator/tree/main/src/shapes/modules
[17]Notation3 rules: https://github.com/odrl-force/odrl-validator/tree/main/src/rules

References

[1] E. Farrell, M. Minghini, A. Kotsev, J. Garrido, B. Tapsall, M. Micheli, M. Posada, European Data Spaces - Scientific Insights into Data Sharing and Utilisation at Scale, Scientific analysis or review, 2023.

[2] Data Spaces Support Centre, Summary of the DSSC Blueprint Version 3.0, 2026. URL: https://assets.dssc.eu/rsc/papers/Blueprint%20V3%20Summary.pdf.

[3] R. Iannella, S. Villata, ODRL Information Model 2.2, W3C Recommendation, W3C, 2018. URL: https://www.w3.org/TR/odrl-model/, [Accessed 08-06-2026].

[4] R. Iannella, M. Steidl, S. Myles, V. Rodríguez-Doncel, ODRL Vocabulary Expression 2.2, W3C Recommendation, W3C, 2018. URL: https://www.w3.org/TR/odrl-vocab/, [Accessed 08-06-2026].

[5] V. Siska, V. Karagiannis, M. Drobics, Building a Dataspace: Technical Overview, 2023. URL: https://www.gaia-x.at/wp-content/uploads/2023/04/WhitepaperGaiaX.pdf.

[6] W. Slabbinck, R. Dedecker, W. Termont, B. Esteves, From Access Control to Usage Control with User-Managed Access, in: 3rd Privacy & Personal Data Management Session @ Solid Symposium, 2025.

[7] A. Cimmino, J. Cano-Benito, R. García-Castro, Open Digital Rights Enforcement framework (ODRE): From descriptive to enforceable policies, Comput. Secur. 150 (2025).

[8] W. Slabbinck, J. Rojas Meléndez, B. Esteves, P. Colpaert, R. Verborgh, Interoperable Interpretation and Evaluation of ODRL Policies, in: The Semantic Web (ESWC), 2025, pp. 192–209.

[9] W. Zhong, K. Huang, X. Du, AgentODRL: A Large Language Model-based Multi-agent System for ODRL Generation 40 (2026) 19477–19485.

[10] D. M. Mustafa, A. Nadgeri, D. Collarana, B. T. Arnold, C. Quix, C. Lange, S. Decker, From Instructions to ODRL Usage Policies: An Ontology Guided Approach, in: Proceedings of the LLM+KG Workshop at VLDB, 2024.

[11] M. Kieffer, P. Serrano-Alvarado, M. Bernelin, Composing complex licenses to facilitate contextual resources reuse, 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.

[12] E. Molino-Peña, J. M. Cruz-Lorite, J. M. García, A. Ruiz-Cortés, TOSL: An Ontology to Detect Abusive Services, 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.

[13] V. Rodríguez-Doncel, N. Roman, Towards Conformance in ODRL 3.0, 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.

[14] H. Drees, D. O. Kubitza, J. Lipp, S. Pretzsch, C. S. Langdon, Mobility data space–first implementation and business opportunities, in: ITS World Congress, 2021.

[15] H. Knublauch, D. Kontokostas, Shapes constraint language (SHACL), W3C Recommendation 11 (2017) 1. URL: https://www.w3.org/TR/shacl/.

[16] Van Woensel, W. and Arndt, D. and Champin, P.-A. and Tomaszuk, D. and Kellogg, G., Notation3 Language, Technical Report, W3C, 2023. URL: https://w3c.github.io/N3/spec/, [Accessed 13-03-2026].

[17] J. De Roo, eyeling, 2026. doi:10.5281/zenodo.19686791.