Towards Formal Semantics for ODRL Policies*

Simon Steyskal[1,2] and Axel Polleres[1]

[1] Vienna University of Economics and Business, Austria
[firstname.lastname]@wu.ac.at
[2] Siemens AG, Vienna, Austria
[firstname.lastname]@siemens.com

Abstract. Most policy-based access control frameworks explicitly model whether execution of certain actions (read, write, etc.) on certain assets should be permitted or denied and usually assume that such actions are disjoint from each other, i.e. there does not exist any explicit or implicit dependency between actions of the domain. This in turn means, that conflicts among rules or policies can only occur if those contradictory rules or policies constrain the same action. In the present paper - motivated by the example of ODRL 2.1 as policy expression language - we follow a different approach and shed light on possible dependencies among actions of access control policies. We propose an interpretation of the formal semantics of general ODRL policy expressions and motivate rule-based reasoning over such policy expressions taking both explicit and implicit dependencies among actions into account. Our main contributions are (i) an exploration of different kinds of ambiguities that might emerge based on explicit or implicit dependencies among actions, and (ii) a formal interpretation of the semantics of general ODRL policies based on a defined abstract syntax for ODRL which shall eventually enable to perform rule-based reasoning over a set of such policies.

1 Introduction

ODRL (Open Digital Rights Language) [7] is a comprehensive policy expression language that aims to develop and promote an open international specification for interchangeable policy expressions. As shown in [1, 12], ODRL has proven to be suitable to express fine-grained access restrictions, access policies, as well as licensing information for Linked Data. It was recently published as version 2.1 and allows to not only model permission or prohibitions of actions over assets, but also to define (optional) obligations for permission rules which need to be fulfilled in order for associated permissions to become active.[3] By using obligations, data owners would be able to define preconditions for using their

* Simon Steyskal has been partially funded by the Vienna Science and Technology Fund (WWTF) through project ICT12-015 and by the Austrian Research Promotion Agency (FFG) grant 845638 (SHAPE).

[3] We note that the specification so far does not define obligations in the form of contractual debts referring to the future upon using the permission, which may be a potential extension.

data, e.g. paying a certain amount of money, which might in turn serve as an incentive to publish their data in the first place as well as duties to be fulfilled when re-sharing the data. Obviously, if there is no possibility to protect or regain some of the expenses made during creating and curating a dataset, data owners might not see any benefit from publishing it.

In order to be able to use ODRL in an automated environment where requests against a set of control policies can be automatically processed and inconsistencies/conflicts among policies automatically detected, a formal specification of the semantics of policies expressed in ODRL is necessary. Unfortunately, there does not exist such an official formal specification, which is primarily caused by the fact that ODRL claims to follow an open design approach which shall allow applications using ODRL to each impose their own concrete interpretation of its semantics [8]. This, however, leads to difficulties when trying to process and consume ODRL policies automatically (i.e. perform reasoning over them), especially because natural language definitions usually leave a margin for interpretation.

Another issue we want to address within the present paper came up during our work on defining the formal semantics of ODRL policies. Most policy-based access control frameworks (e.g. PROTUNE [2]) consider conflicts among policies to only occur between ones that constrain the same action(s) contradictorily (e.g. by prohibiting and permitting a specific action at the same time), but do not take potential dependencies among different actions into account when checking for conflicts. Such dependencies can occur in different manifestations (cf. Section 3) and should be taken into account appropriately when processing requests.

In the present paper we aim to close those gaps of (i) a missing formal specification of ODRL and (ii) resolving ambiguities when handling explicit or implicit dependencies among actions. In particular, our contributions can be summarized as follows:

1. Definition of an abstract syntax for expressing ODRL policies.
2. Formalization of a possible interpretation of ODRL policy semantics.
3. Discussion of a solution proposal for handling implicit dependencies between ODRL actions.

The remainder of this paper is structured as follows: Section 2 provides a brief introduction into ODRL and defines an abstract syntax for expressing ODRL policies. Section 3 discusses the relationship between explicit and implicit dependencies among ODRL actions, and their impact on processing potential query requests, while Section 4 introduces a possible formal interpretation of ODRL policy semantics and Section 5 discusses proposed extended semantics of ODRL conflict resolution strategies. Finally, we discuss related work in Section 6 before we conclude our paper in Section 7.

2 Abstract Syntax of ODRL

The Open Digital Rights Language (ODRL) was invented to provide an open standard for defining policy expressions for digital content and media. The

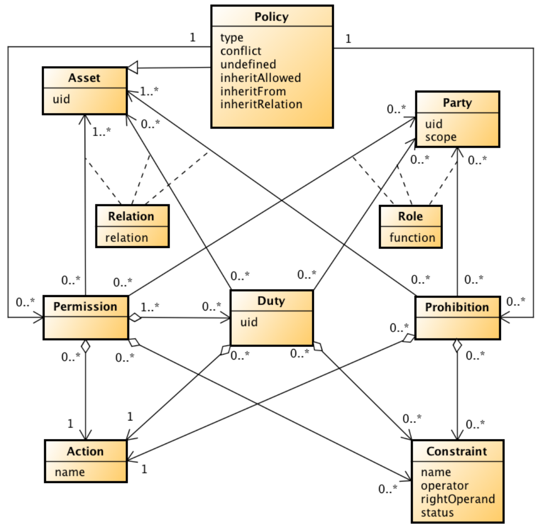

Fig. 1. ODRL Core Model Version 2.1 taken from http://www.w3.org/community/odrl/model/2.1/

ODRL Core Model (cf. Figure 1) contains all major components of an ODRL policy expression.

To the best of our knowledge, there exists no officially agreed on abstract syntax of ODRL that covers all main concepts of the ODRL core model. In the following, we will introduce such an abstract syntax of ODRL that covers its main concepts and continue with utilizing this concise representation to propose a potential interpretation of the formal semantics of ODRL.

Table 1 represents the abstract syntax of ODRL, which was inspired by an approach to formalize XACML used in [9] and can be read as follows:

- text in **bold** represents non-terminal symbols
- text in `typewriter` represents terminal symbols
- text in *italic* represents functions and identifiers
- A^* indicates zero or more occurrences of symbol A
- A^+ indicates one or more occurrences of symbol A
- $A?$ indicates zero or one occurrence of symbol A

A *Policy* contains at least one *PermissionRule* or *ProhibitionRule* and has an associated ODRL *ConflictResolutionStrategy* which is either *permit overrides* (`perm`), *prohibition overrides* (`prohibit`), or *no conflicts allowed* (`invalid`). A *Policy* is applicable, if at least one of the *Rules* it contains matches with the request.

A *ProhibitionRule* defines the prohibition of performing an *Action* on an asset by a particular party which are both declared in the *RuleMatch* component of the *ProhibitionRule*. When its *RuleMatch* and *Action* components match a particular request, the applicability of the *ProhibitionRule* can be further constrained by a

	ODRL Policy Components	
Policy	$\boldsymbol{\mathcal{P}}$	$::= \mathcal{P}_{id} = [\langle(\mathcal{PRR}_{id}\vert\mathcal{PER}_{id})^{+}\rangle,\boldsymbol{\mathcal{ALG}}]$
ProhibitionRule	$\boldsymbol{\mathcal{PRR}}$	$::= \mathcal{PRR}_{id} = [\boldsymbol{\mathcal{RM}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{CONS}}]$
PermissionRule	$\boldsymbol{\mathcal{PER}}$	$::= \mathcal{PER}_{id} = [\boldsymbol{\mathcal{RM}}, \boldsymbol{\mathcal{A}}, \langle\mathcal{DUR}^{*}_{id}\rangle, \boldsymbol{\mathcal{CONS}}]$
DutyRule	$\boldsymbol{\mathcal{DUR}}$	$::= \mathcal{DUR}_{id} = [\boldsymbol{\mathcal{RM}}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{CONS}}]$
ConstraintSet	$\boldsymbol{\mathcal{CONS}}$	$::= \mathcal{CONS}_{id} = \langle\boldsymbol{\mathcal{CON}}^{*}_{id}\rangle$
Constraint	$\boldsymbol{\mathcal{CON}}$	$::= \mathcal{CON}_{id} = f^{bool}(status(a), operator(o), bound(a))$
RuleMatch	$\boldsymbol{\mathcal{RM}}$	$::= \mathcal{RM}_{id} = \langle\boldsymbol{\mathcal{M}}^{+}\rangle$
Match	$\boldsymbol{\mathcal{M}}$	$::= \mathcal{M}_{id} = \phi(a)$
Action	$\boldsymbol{\mathcal{A}}$	$::= \mathcal{A}_{id} = action(a)$
	$\phi(a)$	$::= party(a) \mid asset(a)$
	a	$::= value$
	o	$::=$ `eq` | `neq` | `lt` | `lteq` | `gt` | `gteq`
ConflictRes.Strat.	$\boldsymbol{\mathcal{ALG}}$	$::=$ `perm` | `prohibit` | `invalid`
	Query & Proof	
QueryRequest	$\boldsymbol{\mathcal{Q}}$	$::= \mathcal{Q}_{id} = \langle party(a)?, action(a), asset(a)\rangle$
DutyTarget	$\boldsymbol{\mathcal{DT}}$	$::= \mathcal{DT}_{id} = \langle party(a)?, action(a), asset(a)?\rangle$
DutyProof	$\boldsymbol{\mathcal{DPF}}$	$::= \mathcal{DPF}_{id} = [\boldsymbol{\mathcal{DT}},\mathcal{CON}_{id},status(a)]$
Proof	$\boldsymbol{\mathcal{PF}}$	$::= \mathcal{PF}_{id} = [\mathcal{CON}_{id},status(a)]$
ProofSet	$\boldsymbol{\mathcal{PFS}}$	$::= \langle(\mathcal{DPF}_{id}\vert\mathcal{PF}_{id})^{*}\rangle$

Table 1. Abstract Syntax of ODRL

set of *Constraints*. *Constraints* are represented as boolean formulas that compare a *status* according to an *operator*[4] with a respective *bound*. The *status* of a particular *Constraint* is provided by a respective *Proof* or *DutyProof* that serve as input for the *Constraint*.

PermissionRules are similarly defined as *ProhibitionRules*, but instead of prohibiting the execution of an *Action* they permit it. Furthermore, a sequence of *DutyRules* can be associated with *PermissionRules*. All associated *DutyRules* must be fulfilled in order for the respective *PermissionRule* to become valid.

A *QueryRequest* contains a particular access request that consists of an action and the respective asset it should be performed on, as well as optional information about the party which shall be performing the action.

3 Explicit and Implicit Dependencies among Actions in ODRL

Policy-based access control frameworks allow to explicitly model whether the execution of certain actions on certain assets should be permitted or prohibited and usually consider those actions to be disjoint from each other, i.e. there does not exist any explicit or implicit dependency between actions of the domain.

[4] Note, that we do not take set operators into account, but see them as a potential extension for further work

Which in turn means, that conflicts among rules or policies can only occur if those contradictory rules or policies constrain the same action. However, in some situations there might indeed be interferences between different actions which have to be taken into account. Therefore, we have identified two different types of dependencies among actions of ODRL policies, namely: (i) *implicit dependencies*, and (ii) *explicit dependencies*.

In the following, we will discuss those dependencies in more detail.

3.1 Implicit Dependencies among ODRL Actions

The first dependency we discuss, defines a part-of relationship between actions which is related to Aggregations in UML [3].

Definition 1. *Let A_1 and A_2 be two arbitrary ODRL actions, then A_1 requires the permission of A_2 for its execution,* requires(A_1,A_2)*, if the execution of A_1 involves the execution of A_2.*

That means, if the execution of an action A_1 implies, that an action A_2 must be executable (i.e. execution of A_2 is not denied), then requires(A_1,A_2) holds. To illustrate this relationship, consider the definition of odrl:share given in Figure 2, where its natural language semantics definition is taken from the official ODRL 2.0 specification [6].

odrl:share: *The act of the non-commercial* ***reproduction*** *and* ***distribution*** *of the asset to third-parties.*

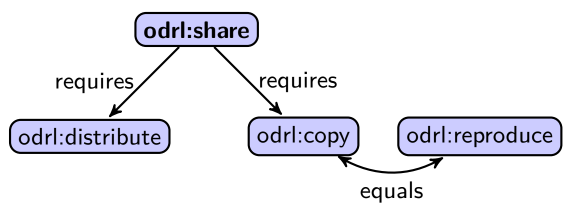

Fig. 2. Implicit dependencies of **odrl:share** (ODRL 2.0).

According to its semantics, odrl:share defines the non-commercial reproduction and distribution of an asset to third-parties. Which obviously would lead to a conflict when considering a policy as defined in Listing 1 which generally permits to share dataset :dataset1 but at the same time denies Assignee :alice to distribute it. A naive evaluation approach would allow :alice to share :dataset1 because there does not exist any rule that prohibits her from performing odrl:share on :dataset1. But since odrl:share defines the non-commercial reproduction (odrl:reproduce) and distribution (odrl:distribute) of an asset, it requires their execution permission to become valid itself, i.e. requires(odrl:share,odrl:reproduce) and requires(odrl:share,odrl:distribute) hold.

```
@prefix odrl: <http://w3.org/ns/odrl/2/> .
@prefix : <http://www.example.com/> .

:sharePolicy a odrl:Set ;
```

```
odrl:permission [
        a odrl:Permission ;
        odrl:action odrl:share ;
        odrl:target :dataset1 ] ;
odrl:prohibition [
        a odrl:Prohibition ;
        odrl:assignee :alice ;
        odrl:action odrl:distribute ;
        odrl:target :dataset1 ] .
```

Listing 1. Prohibition of action **odrl:distribute** causes a conflict with permission of **odrl:share**.

Furthermore, some actions are defined to be equal according to the ODRL 2.0 specification [6] which means that they can be used interchangeably[5].

Definition 2. *Let A_1 and A_2 be two arbitrary ODRL actions, then A_1 is equal to A_2,* equals(A_1,A_2)*, if A_1 and A_2 represent the same functionality according to the official ODRL specification.*

For the example of odrl:share given in Figure 2, this means that odrl:share depends not only on the explicitly mentioned action odrl:reproduce but also on its equivalent action odrl:copy, i.e. equals(odrl:reproduce,odrl:copy) and requires(odrl:share,odrl:copy) hold both.

3.2 Explicit Dependencies among ODRL Actions

In contrast to the aforementioned implicit part-of dependencies among actions in ODRL which are based on their natural language description, there also exist explicit relationships which are indicated by a subsumption hierarchy in the ODRL specification.

Definition 3. *Let A_1 and A_2 be two arbitrary ODRL actions, then* broader(A_1,A_2) *holds, if A_1 represents a broader term for A_2,*

In contrast to the previous defined part-of dependency, this explicit dependency imposes different semantics for the evaluation of ODRL policy expressions. Whenever broader(A_1,A_2) holds and both A_1 and A_2 have different access rights (i.e. permission or prohibition), then either A_1 or A_2 has to adapt its rights, according to the respective conflict resolution strategy in place.

Consider the excerpt of the subsumption hierarchy between actions illustrated in Figure 3. Based on the chosen conflict resolution strategy, if e.g. action odrl:use is prohibited then there cannot exist any other action that represents a narrower term of odrl:use and is permitted (cf. Section 5 for a more detailed discussion).

[5] Note that one of each pair of equivalent terms was defined as deprecated in ODRL 2.1

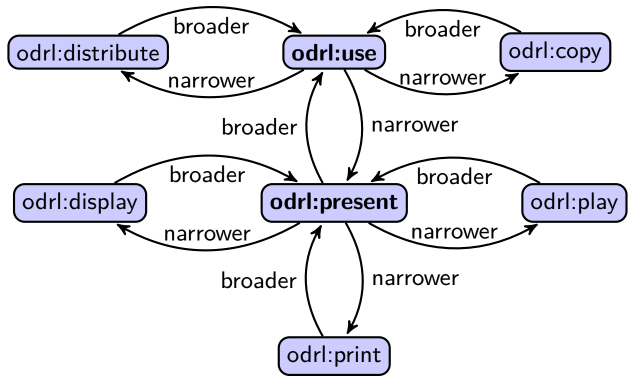

Fig. 3. Excerpt of explicit subsumption hierarchy between actions.

4 Basic Semantics of ODRL Policies

The following section proposes a possible interpretation of the formal semantics of ODRL which differs from earlier approaches defined in [5,10]. Starting from a potential request that was issued against a system, we first evaluate which rules are triggered by the request, and then check whether those rules hold according to potential duties or constraints they might have attached[6]. Eventually, all policies that contain rules which have matched are evaluated by following one of the three proposed ODRL conflict resolution strategies.

Match and RuleMatch. Let $\mathcal{MRM}$ be either a *Match* or a *RuleMatch* component and let $\boldsymbol{\mathcal{QDT}}$ either be a set of all possible *QueryRequests* or *DutyTargets*. A match semantic function is a mapping $[[\mathcal{MRM}]] : \boldsymbol{\mathcal{QDT}} \rightarrow \{\mathsf{m}, \mathsf{nm}\}$, where m and nm denote match and no match respectively.

A certain *Match* component $\mathcal{M}$ (i.e. the attribute value it represents) matches, whenever it is part of a particular *Query* or *DutyTarget*.

$$[[\mathcal{M}]](\mathcal{QDT}) = \begin{cases} \mathsf{m} & \text{if } \mathcal{M} \in \mathcal{QDT} \\ \mathsf{nm} & \text{if } \mathcal{M} \notin \mathcal{QDT} \end{cases} \tag{1}$$

A *RuleMatch* component $\mathcal{RM}$ (i.e. a set of *Match* components defined as $\langle \mathcal{M}_1, \ldots, \mathcal{M}_n \rangle$) only matches, if all of its *Match* components are evaluated to m.

$$[[\mathcal{RM}]](\mathcal{QDT}) = \begin{cases} \mathsf{m} & \text{if } \forall i : [[\mathcal{M}_i]](\mathcal{QDT}) = \mathsf{m} \\ \mathsf{nm} & \text{if } \exists i : [[\mathcal{M}_i]](\mathcal{QDT}) = \mathsf{nm} \end{cases} \tag{2}$$

[6] For now, we assume to have evidence of the fulfillment or violation of constraints/obligations available denoted as *proofs*. Future work will tackle the issue of actually generating or providing those evidences.

Action. Let $\mathcal{A}$ be an *Action* component and let $\boldsymbol{\mathcal{QDT}}$ either be a set of all possible *QueryRequests* or *DutyTargets*. An action semantic function is a mapping $[[\mathcal{A}]] : \boldsymbol{\mathcal{QDT}} \rightarrow \{\mathsf{m}, \mathsf{broadm}, \mathsf{narm}, \mathsf{reqm}, \mathsf{partm}, \mathsf{nm}\}$, where m denotes match, broadm match of broader action, narm match of narrower action, reqm match of requiring action, partm match of required action, and nm denotes no match.

A certain *Action* component (i.e. the action it represents) matches, whenever it is part of a particular *QueryRequest* or *DutyTarget* or if an equivalent action is part of a particular *QueryRequest* or *DutyTarget*. Otherwise, it evaluates to broadm if it is related to a broader action that is part of the *QueryRequest* or *DutyTarget*, or to narm if it is related to a narrower action that is part of the *QueryRequest* or *DutyTarget*, or to partm if it is related to an action that is part of the *QueryRequest* or *DutyTarget* and this action requires the *Action* component for its execution, or to reqm if it requires another action for its execution and this required action is part of the *QueryRequest* or *DutyTarget*, or to nm otherwise.

$$[[\mathcal{A}]](\mathcal{QDT}) = \begin{cases} \mathsf{m} & \text{if } \mathcal{A} \in \mathcal{QDT} \text{ or} \\ & \exists i : \mathsf{equals}(\mathcal{A}, \mathcal{A}_i) \wedge \mathcal{A}_i \in \mathcal{QDT} \\ \mathsf{narm} & \text{if } \exists i : \mathsf{broader}(\mathcal{A}_i, \mathcal{A}) \wedge \mathcal{A}_i \in \mathcal{QDT} \\ \mathsf{broadm} & \text{if } \exists i : \mathsf{broader}(\mathcal{A}, \mathcal{A}_i) \wedge \mathcal{A}_i \in \mathcal{QDT} \\ \mathsf{partm} & \text{if } \exists i : \mathsf{requires}(\mathcal{A}_i, \mathcal{A}) \wedge \mathcal{A}_i \in \mathcal{QDT} \\ \mathsf{reqm} & \text{if } \exists i : \mathsf{requires}(\mathcal{A}, \mathcal{A}_i) \wedge \mathcal{A}_i \in \mathcal{QDT} \\ \mathsf{nm} & \text{otherwise} \end{cases} \tag{3}$$

Constraint and ConstraintSet. Let $\mathcal{CON}$ be a *Constraint* component, $\mathcal{CONS} = \langle \mathcal{CON}_1, \ldots, \mathcal{CON}_n \rangle$ a *ConstraintSet* component, and let $\boldsymbol{\mathcal{PFS}} = \langle \mathcal{DPF}_1, \ldots, \mathcal{DPF}_m, \mathcal{PF}_1, \ldots, \mathcal{PF}_n \rangle$ represent all possible *ProofSets*. A constraint semantic function is a mapping $[[\mathcal{CON}]] : \boldsymbol{\mathcal{PFS}} \rightarrow \{\mathsf{t}, \mathsf{f}\}$, where t and f indicate whether the boolean formula represented by $\mathcal{CON}$ holds, given a *ProofSet* $\mathcal{PFS}$ as input.

This boolean formula is evaluated, if the provided *ProofSet* $\mathcal{PFS}$ contains a *Proof* $\mathcal{PF}$ that is associated with the respective *Constraint* of the formula. If no associated *Proof* exists, it is evaluated to f.

$$[[\mathcal{CON}]](\mathcal{PFS}) = \begin{cases} f^{bool}(\mathcal{PF}_i, operator(o), bound(a)) & \text{if } \exists i : \mathcal{PF}_i \ \wedge \ i = id \\ \mathsf{f} & \text{otherwise} \end{cases} \tag{4}$$

A *ConstraintSet* component only evaluates to t, if all of its *Constraint* components are evaluated to t or the *ConstraintSet* is empty, i.e. there do not exist any associated *Constraints* at all.

$$[[\mathcal{CONS}]](\mathcal{PFS}) = \begin{cases} \mathsf{t} & \text{if } \forall i : [[\mathcal{CON}_i]](\mathcal{PFS}) = \mathsf{t} \text{ or} \\ & \quad \mathcal{CONS} = \emptyset \\ \mathsf{f} & \text{if } \exists i : [[\mathcal{CON}_i]](\mathcal{PFS}) = \mathsf{f} \end{cases} \tag{5}$$

DutyRule. Let $\mathcal{DUR} = [\mathcal{RM}, \mathcal{CONS}]$ be a *DutyRule* component and let $\boldsymbol{\mathcal{PFS}} = \langle \mathcal{DPF}_1, \ldots, \mathcal{DPF}_m, \mathcal{PF}_1, \ldots, \mathcal{PF}_n \rangle$ represent all possible *ProofSets*. A duty rule semantic function is a mapping $[[\mathcal{DUR}]] : \boldsymbol{\mathcal{PFS}} \rightarrow \{\mathsf{t}, \mathsf{f}\}$, where t represents the fulfillment of $\mathcal{DUR}$, and f the opposite.

$\mathcal{DUR}$ evaluates to t, if there exists at least one *DutyProof* $\mathcal{DPF}$ in the provided *ProofSet* $\mathcal{PFS}$ whose *DutyTarget* $\mathcal{DT} \in \mathcal{DPF}$ matches with the *RuleMatch* component of $\mathcal{DUR}$, and its *ConstraintSet* returns true. It evaluates to f in any other case.

$$[[\mathcal{DUR}]](\mathcal{PFS}) = \begin{cases} \mathsf{t} & \text{if } \exists i : \mathcal{DPF}_i \in \mathcal{PFS} \wedge [[\mathcal{RM}]](\mathcal{DT}) = \mathsf{m} \wedge \\ & [[\mathcal{A}]](\mathcal{DT}) = \mathsf{m} \wedge [[\mathcal{CONS}]](\mathcal{PFS}) = \mathsf{t} \\ \mathsf{f} & \text{otherwise} \end{cases} \tag{6}$$

PermissionRule. Let $\mathcal{PER}$ be a *PermissionRule* component of the form $\mathcal{PER} = [\mathcal{RM}, \mathcal{A}, \boldsymbol{\mathcal{DUR}}, \mathcal{CONS}]$ where $\boldsymbol{\mathcal{DUR}} = \langle \mathcal{DUR}_1, \ldots, \mathcal{DUR}_n \rangle$, let $\boldsymbol{\mathcal{Q}}$ be a set of all possible *QueryRequests*, and let $\boldsymbol{\mathcal{PFS}}$ denote all possible *ProofSets*. A permission rule semantic function is a mapping $[[\mathcal{PER}]] : \boldsymbol{\mathcal{Q}}, \boldsymbol{\mathcal{PFS}} \rightarrow \{\mathsf{permission}, \mathsf{cper}, \mathsf{cpro}, \mathsf{na}, \mathsf{nm}\}$, where given $\boldsymbol{\mathcal{PFS}}$ as input, $\mathsf{permission}$ represents permission of $\boldsymbol{\mathcal{Q}}$, cper denotes conditional permission of $\boldsymbol{\mathcal{Q}}$, cpro indicates conditional prohibition of $\boldsymbol{\mathcal{Q}}$, and na, nap represent that $\mathcal{PER}$ is not active or not applicable respectively.

$\mathcal{PER}$ evaluates to $\mathsf{permission}$, if its *RuleMatch* component matches with provided *QueryRequest* $\mathcal{Q}$, its *ConstraintSet* component returns true, and if it has no associated duties. It evaluates to cpro if its *RuleMatch* component matches with $\mathcal{Q}$, its *ConstraintSet* component returns true, but it has at least one associated *DutyRule* component that evaluates to false given a specific *ProofSet* $\mathcal{PFS}$ as input. It evaluates to cper if its *RuleMatch* component matches with $\mathcal{Q}$, its *ConstraintSet* component returns true, and all associated *DutyRule* components evaluate to true given $\mathcal{PFS}$ as input. Finally, a *PermissionRule* component evaluates to na if its *RuleMatch* component matches with $\mathcal{Q}$ but its *ConstraintSet* component returns false, and it evaluates to nap if its *RuleMatch* component does not match with $\mathcal{Q}$.

$$[[\mathcal{PER}]](\mathcal{Q}, \mathcal{PFS}) = \begin{cases} \mathsf{permission} & \text{if } [[\mathcal{RM}]](\mathcal{Q}) = \mathsf{m},\ [[\mathcal{A}]](\mathcal{Q}) \neq \mathsf{nm}, \\ & [[\mathcal{CONS}]](\mathcal{PFS}) = \mathsf{t} \text{ and } \boldsymbol{\mathcal{DUR}} = \emptyset \\ \mathsf{cpro} & \text{if } [[\mathcal{RM}]](\mathcal{Q}) = \mathsf{m},\ [[\mathcal{A}]](\mathcal{Q}) \neq \mathsf{nm}, \\ & [[\mathcal{CONS}]](\mathcal{PFS}) = \mathsf{t} \text{ and } \exists i : [[\mathcal{DUR}_i]](\mathcal{PFS}) = \mathsf{f} \\ \mathsf{cper} & \text{if } [[\mathcal{RM}]](\mathcal{Q}) = \mathsf{m},\ [[\mathcal{A}]](\mathcal{Q}) \neq \mathsf{nm}, \\ & [[\mathcal{CONS}]](\mathcal{PFS}) = \mathsf{t} \text{ and } \forall i : [[\mathcal{DUR}_i]](\mathcal{PFS}) = \mathsf{t} \\ \mathsf{na} & \text{if } [[\mathcal{RM}]](\mathcal{Q}) = \mathsf{m},\ [[\mathcal{A}]](\mathcal{Q}) \neq \mathsf{nm} \text{ and} \\ & [[\mathcal{CONS}]](\mathcal{PFS}) = \mathsf{f} \\ \mathsf{nap} & \text{otherwise} \end{cases} \tag{7}$$

ProhibitionRule. Let $\mathcal{PRR}$ be a *ProhibitionRule* component of the form $\mathcal{PRR}$ $= [\mathcal{RM}, \mathcal{A}, \mathcal{CONS}]$, let $\boldsymbol{\mathcal{Q}}$ be a set of all possible *QueryRequests*, and let $\boldsymbol{\mathcal{PFS}}$ denote all possible *ProofSets*. A prohibition rule semantic function is a mapping $[[\mathcal{PRR}]] : \boldsymbol{\mathcal{Q}}, \boldsymbol{\mathcal{PFS}} \rightarrow \{\textsf{prohibition}, \textsf{na}, \textsf{nm}\}$, where given $\boldsymbol{\mathcal{PFS}}$ as input, prohibition represents the prohibition of $\boldsymbol{\mathcal{Q}}$, na denotes that $\mathcal{PRR}$ is not active, and nap states that $\mathcal{PRR}$ is not applicable.

$\mathcal{PRR}$ evaluates to prohibition, if its *RuleMatch* component matches with the *QueryRequest*$\mathcal{Q}$ and its *ConstraintSet* component returns true given a specific *ProofSet* $\mathcal{PFS}$ as input. It evaluates to na if its *RuleMatch* component matches with $\mathcal{Q}$ but its *ConstraintSet* component returns false, and it evaluates to nap if its *RuleMatch* component does not match with $\mathcal{Q}$ (i.e. the rule is not applicable).

$$[[\mathcal{PRR}]](\mathcal{Q}, \mathcal{PFS}) = \begin{cases} \textsf{prohibition} & \text{if } [[\mathcal{RM}]](\mathcal{Q}) = \textsf{m},\ [[\mathcal{A}]](\mathcal{Q}) \neq \textsf{nm} \text{ and} \\ & [[\mathcal{CONS}]](\mathcal{PFS}) = \textsf{t} \\ \textsf{na} & \text{if } [[\mathcal{RM}]](\mathcal{Q}) = \textsf{m},\ [[\mathcal{A}]](\mathcal{Q}) \neq \textsf{nm} \text{ and} \\ & [[\mathcal{CONS}]](\mathcal{PFS}) = \textsf{f} \\ \textsf{nap} & \text{otherwise} \end{cases} \tag{8}$$

Policy. Let $\mathcal{P}$ be a *Policy* component of the form $\mathcal{P} = [\boldsymbol{\mathcal{R}}, \boldsymbol{\mathcal{ALG}}]$, where $\boldsymbol{\mathcal{R}} = \langle \mathcal{R}_1, \ldots, \mathcal{R}_n \rangle$ is the set of all *Rules* of $\mathcal{P}$ with $\mathcal{R}_i, \mathcal{R}_j \in \boldsymbol{\mathcal{R}}$ representing either a *ProhibitionRule* or a *PermissionRule*, and $\boldsymbol{\mathcal{ALG}}$ is denoting the conflict resolution strategy of the *Policy*. Further, let $\boldsymbol{\mathcal{Q}}$ be a set of all possible *QueryRequests*, and let $\boldsymbol{\mathcal{PFS}}$ denote all possible *ProofSets*. A policy semantic function is a mapping $[[\mathcal{P}]] : \boldsymbol{\mathcal{Q}}, \boldsymbol{\mathcal{PFS}} \rightarrow \{\textsf{permission}, \textsf{prohibition}, \textsf{cpro}, \textsf{na}, \textsf{nm}\}$, where given $\boldsymbol{\mathcal{PFS}}$ as input, permission represents permission of $\boldsymbol{\mathcal{Q}}$, prohibition represents prohibition of $\boldsymbol{\mathcal{Q}}$, cpro indicates conditional prohibition of $\boldsymbol{\mathcal{Q}}$, and na, nap represent that $\mathcal{P}$ is not active or not applicable respectively.

A *Policy* $\mathcal{P}$ is not active, if all $\mathcal{R}$ in $\mathcal{P}$ are evaluated to na. $\mathcal{P}$ is not applicable (nap), if all $\mathcal{R}$ in $\mathcal{P}$ are evaluated to nap. If there is at least one $\mathcal{R}$ in $\mathcal{P}$ which is neither evaluated to na nor nap, $\mathcal{P}$ is evaluated to the result returned by the respective conflict resolution strategy $\boldsymbol{\mathcal{ALG}}$ that takes $\mathcal{I} = [\boldsymbol{\mathcal{R}}, \mathcal{Q}, \mathcal{PFS}]$ as input.

$$[[\mathcal{P}]](\mathcal{Q}, \mathcal{PFS}) = \begin{cases} \textsf{na} & \text{if } \forall i : [[\mathcal{R}_i]](\mathcal{Q}, \mathcal{PFS}) = \textsf{na} \\ \textsf{na} & \text{if } \exists i : \neg([[\mathcal{R}_i]](\mathcal{Q}, \mathcal{PFS}) = {\scriptstyle (\textsf{permission}|\textsf{prohibition})}) \\ & \wedge\ \exists j : [[\mathcal{R}_j]](\mathcal{Q}, \mathcal{PFS}) = \textsf{na} \\ \textsf{nap} & \text{if } [[\mathcal{RM}]](\mathcal{Q}) = \textsf{nm} \text{ and } [[\mathcal{A}]](\mathcal{Q}) = \textsf{nm} \\ \otimes_{\boldsymbol{\mathcal{ALG}}(\mathcal{I})} & \text{otherwise} \end{cases} \tag{9}$$

5 Proposed Semantics of ODRL Conflict Resolution Strategies

Sometimes, it may be the case that an unambiguous answer to a certain query request cannot be computed. Which is usually the case, if two or more mutually exclusive rules are triggered and thus produce multiple (possibly mutually exclusive) answers. Such a potential conflict is illustrated in Listing 2 where execution of action odrl:use on asset :dataset1 is both permitted and prohibited at the same time.

```
@prefix odrl: <http://w3.org/ns/odrl/2/> .
@prefix : <http://www.example.com/> .

:policy1 a odrl:Set ;
        odrl:permission [
                a odrl:Permission ;
                odrl:action odrl:use ;
                odrl:target :dataset1 ] ;
        odrl:prohibition [
                a odrl:Prohibition ;
                odrl:action odrl:use ;
                odrl:target :dataset1 ] .
```

Listing 2. Two conflicting rules of a policy.

To deal with this issue, the official ODRL specification defines an optional attribute for policies called conflict, that represents the conflict resolution strategy a policy must adhere to. There are three different conflict resolution strategies defined, namely:

perm: Permissions always take precedence over prohibitions.
prohibit: Prohibitions always take precedence over permissions.
invalid: Any conflicts cause invalidity of the policy.

In case attribute conflict is omitted, the default conflict resolution strategy is set to invalid.

Apart from their rather concise natural language description listed above, there does not exist any detailed definition of the semantics of ODRL conflict resolution strategies. Although, they all might seem quite straightforward to realize, there are some specific scenarios where a more elaborate semantics definition is necessary. For example, consider the policy illustrated in Listing 3, where actions odrl:use and odrl:delete are prohibited and action odrl:give is permitted to be performed on :dataset1.

```
@prefix odrl: <http://w3.org/ns/odrl/2/> .
@prefix : <http://www.example.com/> .

:policy2 a odrl:Set ;
        odrl:prohibition [
                a odrl:Prohibition ;
```

```
        odrl:action odrl:use ;
        odrl:target :dataset1 ] ;
odrl:permission [
        a odrl:Permission ;
        odrl:action odrl:give ;
        odrl:target :dataset1 ] .
odrl:prohibition [
        a odrl:Prohibition ;
        odrl:action odrl:delete ;
        odrl:target :dataset1 ] .
```

Listing 3. Two conflicting rules of a policy.

In the following, we will propose and explain suitable semantics for each ODRL conflict resolution strategy.

Note, that we (i) value evaluation results obtained by duties, i.e. cper or cpro higher than any conflict resolution strategy, and (ii) do not treat *Rules* assigned to a specific party different from those having no associated party. Furthermore, we abbreviate *QueryRequests* with $\mathcal{Q}$, *Rules* with $\mathcal{R}$, and *Actions* with $\mathcal{A}$.

5.1 Permission Overrides (perm)

First conflict resolution strategy values permissions more than prohibitions thus, whenever there are two *Rules* in conflict with each other, the one granting permission to execute an action a on a particular asset cannot be overwritten. Nevertheless, there are some exceptions:

1. If there exists a rule which constrains an action that is either (i) equal to the one contained in the query request, (ii) a broader term for the action contained in the query request, or (iii) an action which is required to be executable in order to perform the one contained in the query request, and this rule evaluates to cpro, return cpro.
2. If 1. does not hold and there exists a rule which constrains an action that is either (i) equal to the one contained in the query request, (ii) a broader term for the action contained in the query request, or (iii) an action which requires the one contained in the query request to be executable, and this rule evaluates to cper or permission, return permission.
3. If all rules contain the same or equal actions to the ones queried and all rules evaluate to the same result r, then return r.
4. Otherwise, return na.

$$\bigotimes_{perm}(\mathcal{I}) = \begin{cases} \mathsf{cpro} & \text{if } \exists i : [[A_i]](\mathcal{Q}) = \mathsf{(m|broadm|partm)} \wedge [[R_i]](\mathcal{Q}, \mathcal{PFS}) = \mathsf{cpro} \\ \mathsf{permission} & \text{if } \exists i : [[A_i]](\mathcal{Q}) = \mathsf{(m|broadm|reqm)} \wedge [[R_i]](\mathcal{Q}, \mathcal{PFS}) = \mathsf{(permission|cper)} \\ & \text{and } \neg\exists j : [[A_j]](\mathcal{Q}) = \mathsf{(m|broadm|partm)} \wedge [[R_j]](\mathcal{Q}, \mathcal{PFS}) = \mathsf{cpro} \\ r & \text{if } \forall i : [[A_i]](\mathcal{Q}) = \mathsf{m} \wedge [[R_i]](\mathcal{Q}, \mathcal{PFS}) = r \\ \mathsf{na} & \text{otherwise} \end{cases} \quad (10)$$

5.2 Prohibition Overrides (prohibit)

Second conflict resolution strategy values prohibitions more than permissions thus, whenever there are two *Rules* in conflict with each other the one prohibiting the execution of an action a on a particular asset cannot be overwritten. Again, there are some exceptions:

1. If there exists a rule which constrains an action that is either (i) equal to the one contained in the query request, (ii) a broader term for the action contained in the query request, or (iii) an action which is required to be executable in order to perform the one contained in the query request, and this rule evaluates to cpro, return cpro.
2. If 1. does not hold and there exists a rule which constrains an action that is either (i) equal to the one contained in the query request, (ii) a broader term for the action contained in the query request, or (iii) an action which requires the one contained in the query request to be executable, and this rule evaluates to cper, return permission.
3. If 1. and 2. does not hold and there exists a rule which constrains an action that is either (i) equal to the one contained in the query request, (ii) a broader term for the action contained in the query request, or (iii) an action which is required to be executable in order to perform the one contained in the query request, and this rule evaluates to prohibition, return prohibition.
4. If all rules contain the same or equal actions to the ones queried and all rules evaluate to the same result r, then return r.
5. Otherwise, return na.

$$\bigotimes_{prohibit}(\mathcal{I}) = \begin{cases} \textsf{cpro} & \text{if } \exists i : [[A_i]](\mathcal{Q}) = \textsf{(m|broadm|partm)} \wedge [[R_i]](\mathcal{Q},\mathcal{PFS}) = \textsf{cpro} \\ \textsf{permission} & \text{if } \exists i : [[A_i]](\mathcal{Q}) = \textsf{(m|broadm|reqm)} \wedge [[R_i]](\mathcal{Q},\mathcal{PFS}) = \textsf{cper} \\ & \text{and } \neg\exists j : [[A_j]](\mathcal{Q}) = \textsf{(m|broadm|partm)} \wedge [[R_j]](\mathcal{Q},\mathcal{PFS}) = \textsf{cpro} \\ \textsf{prohibition} & \text{if } \exists i : [[A_i]](\mathcal{Q}) = \textsf{(m|broadm|partm)} \wedge [[R_i]](\mathcal{Q},\mathcal{PFS}) = \textsf{prohibition} \\ & \text{and } \neg\exists j : [[A_j]](\mathcal{Q}) = \textsf{(m|broadm|reqm)} \wedge [[R_j]](\mathcal{Q},\mathcal{PFS}) = \textsf{cper} \\ & \text{and } \neg\exists k : [[A_k]](\mathcal{Q}) = \textsf{(m|broadm|partm)} \wedge [[R_k]](\mathcal{Q},\mathcal{PFS}) = \textsf{cpro} \\ r & \text{if } \forall i : [[A_i]](\mathcal{Q}) = \textsf{m} \wedge [[R_i]](\mathcal{Q},\mathcal{PFS}) = r \\ \textsf{na} & \text{otherwise} \end{cases} \tag{11}$$

5.3 No Conflicts Allowed (invalid)

Third conflict resolution strategy does not allow any conflicting *Rules*, therefore whenever there are two *Rules* returning inconsistent answers, no results can be provided.

1. All rules must evaluate to the same result. If two rules evaluate to different results, those results must be one of cper or permission.

2. Otherwise, return an error.

$$\bigotimes_{invalid}(\mathcal{I}) = \begin{cases} r_i & \forall i \forall j : ([[R_i]](\mathcal{Q}, \mathcal{PFS}) = r_i \wedge [[R_j]](\mathcal{Q}, \mathcal{PFS}) = r_j) \rightarrow (r_i = r_j \vee \\ & r_i \neq r_j \rightarrow (r_i = \text{(cper|permission)} \wedge r_j = \text{(cper|permission)})) \\ \text{error} & \text{otherwise} \end{cases} \quad (12)$$

6 Related Work

Over the last couple of years, very little research has been conducted into the formal semantics for ODRL. While in [10] the authors propose formal semantics to a fragment of ODRL based on First-Order Logic and limit themselves to a very small subset of supported actions, the authors of [5] use finite-automata like structures to model permissions and their respective actions they permit. In contrast to both of those approaches, we defined an abstract syntax for all basic concepts of ODRL and formalized their semantics together with the semantics of conflict resolution strategies accordingly. Other approaches try to capture the semantics of ODRL in terms of ontologies [4, 8] which is very similar to the semantics definition of our approach but differs in terms of treatment of implicit dependencies between actions as well as the proposed abstract syntax. Complementary our work, there has been work to formalize licence compatibility [11], which though was not embedded in the framework of ORDL, but might be an interesting direction to look into for formally grounding our semantics likewise into Deontic logic.

7 Conclusion

In the present paper, we defined an abstract syntax for expressing ODRL policies which served as a foundation for formalizing a possible interpretation of basic ODRL policy semantics. We furthermore discussed the impact of explicit and implicit dependencies among ODRL actions on the evaluation of policy expressions. While the former is explicitly defined in the ODRL specification and modeled as subsumption hierarchy between actions, the latter can only be implicitly derived from the natural language semantics definition of actions and expressed as part-of relationship among actions. Which we both took into account when formalizing ODRL's semantics.

First point to be addressed is to introduce the concept of *PolicySets* as container for policies which allows to combine the evaluation results of policies independently of their respective chosen conflict resolution strategy. Second, we want to formalize and extend the mapping between ODRL policies and logic programs, which enables basic, rule-based reasoning tasks and was omitted in the present paper because of page restrictions. Finally, we will address the elaborate provision of proofs for constraints and duties which are currently assumed to be provided by the requester itself. Especially addressing the latter point, offers interesting new research directions and allows for possible collaborations with other research fields like Business Process Management, where correct completion of a business process that was automatically generated based on a constraint or duty serves as a proof of their fulfillment.

References

1. Elena Cabrio, Alessio Palmero Aprosio, and Serena Villata. These are your rights - A natural language processing approach to automated RDF licenses generation. In *The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings*, pages 255–269, 2014.
2. Juri Luca De Coi, Daniel Olmedilla, Piero A Bonatti, and Luigi Sauro. Protune: A framework for semantic web policies. In *International Semantic Web Conference (Posters & Demos)*, volume 401, page 128, 2008.
3. Martin Fowler and Kendall Scott. *UML distilled - a brief guide to the Standard Object Modeling Language (2. ed.)*. Addison-Wesley-Longman, 2000.
4. Roberto García, Rosa Gil, Isabel Gallego, and Jaime Delgado. Formalising ODRL semantics using web ontologies. In *Proc. 2nd Intl. ODRL Workshop*, pages 1–10, 2005.
5. Markus Holzer, Stefan Katzenbeisser, and Christian Schallhart. Towards formal semantics for ODRL. In *Proceedings of the First International Workshop on the Open Digital Rights Language (ODRL), Vienna, Austria, April 22-23, 2004*, pages 137–148, 2004.
6. Renato Iannella and Susanne Guth. Odrl version 2.0 common vocabulary. W3C ODRL Community Group, 2012. https://www.w3.org/community/odrl/two/vocab/.
7. Renato Iannella, Susanne Guth, Daniel Pähler, and Andreas Kasten. Odrl: Open digital rights language 2.1. W3C ODRL Community Group, 2012. http://www.w3.org/community/odrl/.
8. Andreas Kasten and Rüdiger Grimm. Making the Semantics of ODRL and URM Explicit Using Web Ontologies. *Virtual Goods*, pages 77–91, 2010.
9. Carroline Dewi Puspa Kencana Ramli, Hanne Riis Nielson, and Flemming Nielson. XACML 3.0 in Answer Set Programming. In Elvira Albert, editor, *Logic-Based Program Synthesis and Transformation*, volume 7844 of *Lecture Notes in Computer Science*, pages 89–105. Springer Berlin Heidelberg, 2013.
10. Riccardo Pucella and Vicky Weissman. A Formal Foundation for ODRL. *CoRR*, abs/cs/0601085, 2006.
11. Antonino Rotolo, Serena Villata, and Fabien Gandon. A deontic logic semantics for licenses composition in the web of data. In *Int'l Conf. on Artificial Intelligence and Law ICAIL*, pages 111–120, 2013.
12. Simon Steyskal and Axel Polleres. Defining expressive access policies for linked data using the ODRL ontology 2.0. In *Proceedings of the 10th International Conference on Semantic Systems, SEMANTICS 2014, Leipzig, Germany, September 4-5, 2014*, pages 20–23, 2014.