3. ODRL Information Model

The basic context of an ODRL Policy is that only an explicitly permitted use may be executed. Any use not explicitly permitted is prohibited by default. An ODRL Policy only permits the action explicitly specified in a Permission and all other actions are implicitly prohibited. An action defined in a Prohibition *SHOULD* only refine (or directly relate to) the semantics of an action defined in one of the Permissions in the ODRL Policy.

For example, an ODRL Policy that has the action "present" Permission and may also have the action "print" Prohibition (as these actions are related hierarchically in the ODRL Vocabulary [*vocab-odrl*]).

Note that ODRL Profiles can be developed and used to refine the basic context of an ODRL Policy. Hence, the application of an ODRL Profile must be understood by the consuming community and systems.

The figure below shows the ODRL Information Model. The Policy is the central entity that holds an ODRL policy together.

Fig. 1 ODRL Information Model

As the Information Model diagram shows the key Permission, Prohibition and Duty entities are subtypes of the abstract Rule class. These Rules have the same relationships to the other key entities (Action, Constraint, Asset, and Party). The core difference is in their semantics:

- Permission says that the Party *MAY* perform an Action,
- Duty says that the Party *MUST* perform an Action, and
- Prohibition says that the Party *MUST NOT* perform an Action.

The *Rule* class also makes it possible to easily extend the Information Model in Profiles by adding policy expressions (as subclasses of *Rule*) that are not possible by default.

The cardinalities shown in the ODRL Information Model allow for the greatest flexibility in expressing associations between the key entities. However, Policy types and ODRL Profiles may express narrower and/or specific cardinalities on these entities.

A Permission *MAY* allow a particular Action to be executed on a related Asset, e.g., "play the audio file abc.mp3". A Constraint such as "at most 10 times" might be added to specify the Permission more precisely. The Party that grants this Permission is linked to it with the Role Assigner, the Party that is granted the Permission is linked to it with the Role Assignee, e.g., "assigner VirtualMusicShop grants the Permission to Assignee Alice". Additionally, a Permission *MAY* be linked to Duty entities.

Similar to Permissions, a Duty states that a certain Action *MUST* be executed by the Party with the Role Assignee for the Permission to be valid, e.g. "Alice must pay 5 Euros in order to get the Permission to play abc.mp3".

The Prohibition entity is used in the same way as Permission, with the difference that it *MUST NOT* refer to Duties and that the Action *MUST NOT* be exercised, e.g. "Alice is forbidden to use abc.mp3 commercially".

The following sections describes each entity of the Information Model in greater detail.

3.1 Policy

The Policy entity contains the following attributes:

- `uid`: the unique identification of the Policy entity (*REQUIRED*)
- `type`: indicates the semantics of the Policy entity (*REQUIRED*). These are further described in the ODRL vocabulary [*vocab-odrl*] and ODRL Profiles.
- `conflict`: indicates how to handle Policy conflicts (*OPTIONAL*)
- `undefined`: indicates how to handle undefined Actions (*OPTIONAL*)
- `inheritAllowed`: indicates if the Policy entity can be inherited (*OPTIONAL*)
- `profile`: the identifier of the ODRL Profile that this Policy conforms to (*OPTIONAL*)

The `uid` attribute *MUST* be a unique identifier.

The range of values for the Policy entity's `type` attribute will be described in the ODRL vocabulary [*vocab-odrl*] or in ODRL Profiles. This value *MAY* also impose further constraints on the Information Model. It is important that the `type` attribute be clearly understood in policy expressions as the semantics *MAY* impose restrictions on the expression language constructs such as cardinalities between entities. For example, the ODRL Agreement Policy Type stipluates that it must contain two Parties (an Assigner and Assignee).

Example Use Case: A Policy of type Set states that the Asset http//example.com/asset:9898 is the target of the Permission read and the Prohibition reproduce. No Parties or other elements are involved.

Example 1

```
{
"@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Set",
    "uid": "http://example.com/policy:1010",
    "permission": [{
        "target": "http://example.com/asset:9898",
        "action": "odrl:read"
    }],
    "prohibition": [{
        "target": "http://example.com/asset:9898",
        "action": "odrl:reproduce"
    }]
}
```

Example Use Case: A Policy of type Set states that the Asset http//example.com/asset:9898 is the target of the Permission reproduce, the Duty attribute to http://example.com/owner:9898, and the Prohibition translate. Two Parties are involved, namely the Assigner of the Permissions and the Party to be attributed.

Example 2

```
{
"@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Set",
    "uid": "http://example.com/policy:1010",
    "permission": [{
        "target": "http://example.com/asset:9898",
        "action": "odrl:reproduce",
        "assigner": "http://example.com/assigner:88",
        "duty": [{
                "action": "odrl:attribute",
                "attributedParty": "http://example.com/owner:9898"
        }]
    }],
    "prohibition": [{
        "target": "http://example.com/asset:9898",
        "action": "odrl:translate"
    }]
}
```

3.1.1 Policy Structures

The ODRL Information Model supports flexibility in the information structures to declare ODRL expressions. A Policy *MAY* contain multiple Rules, and each Rule *MAY* contain multiple Assets, Parties, Actions, Constraints, and Duties. A Policy *MAY* also contain Assets, Parties, and Actions at the Policy level, and these apply to all of the enclosing Rules in the Policy.

At the core *atomic* level, an ODRL Rule (Permission and/or Prohibition) would typically contain one Asset, one or more Parties, one Action, and potentially one Constraint and/or Duty, as shown in the example below:

Example 3

```
{
    "@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Agreement",
    "uid": "http://example.com/policy:8888",
    "permission": [{
        "target": "http://example.com/music/1999.mp3",
        "assigner": "http://example.com/org/sony-music",
        "assignee": "http://example.com/people/billie",
        "action": "odrl:play",
        "constraint": "...",
        "duty": "..."
        }]
    }]
}
```

As multiple Assets, Parties, and Actions are support for each Rule, then the following (snippet) example with two Assets:

Example 4

```
"permission": [{
    "target": "http://example.com/music/1999.mp3",
```

```
    "target": "http://example.com/music/PurpleRain.mp3",
    "assigner": "http://example.com/org/sony-music",
    "assignee": "http://example.com/people/billie",
    "action": "odrl:play",
    "constraint": "...",
    "duty": "..."
    }]
```

Would then be mapped to two *atomic* Rules:

Example 5

```
"permission": [{
    "target": "http://example.com/music/1999.mp3",
    "assigner": "http://example.com/org/sony-music",
    "assignee": "http://example.com/people/billie",
    "action": "odrl:play",
    "constraint": "...",
    "duty": "..."
    }]
"permission": [{
    "target": "http://example.com/music/PurpleRain.mp3",
    "assigner": "http://example.com/org/sony-music",
    "assignee": "http://example.com/people/billie",
    "action": "odrl:play",
    "constraint": "...",
    "duty": "..."
    }]
```

The processing model for Permission/Prohibition Rules with multiple Assets, Parties, and Actions to generate *atomic* Rules includes:

1. If there are multiple Assets (with the same Relation), then create new Rules (one for each Asset) and include one Asset relation, and all the other (non-Asset) entities.
2. If there are multiple Parties (with the same Role), then create new Rules (one for each Party) and include one Party Role, and all the other (non-Party Role) entities.
3. If there are multiple Actions, then create new Rules (one for each Action) and include one Action, and all the other (non-Action) entities.

An ODRL Policy *MAY* also declare multiple Assets, Parties, and Actions at the Policy level. This indicates that these are all common to all the enclosing Rules, as shown in the below example:

Example 6

```
{
    "@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Agreement",
    "uid": "http://example.com/policy:8888",
    "target": "http://example.com/music/1999.mp3",
    "assigner": "http://example.com/org/sony-music",
    "action": "odrl:play",
    "permission": [{
        "assignee": "http://example.com/people/billie"
        },
        {
        "assignee": "http://example.com/people/murphy"
```

```
        }]
    }]
}
```

To fully expand the Rules, the Policy-level Assets, Parties, and Actions *MUST* be added to all the Rules in the Policy. As shown below, the policy-level Target, Assigner, and Action are added to the two permission Rules:

Example 7

```
{
    "@context": {
        "odrl": "http://www.w3.org/ns/odrl/2/"
        }
    "type": "odrl:Agreement",
    "uid": "http://example.com/policy:8888",
    "permission": [{
        "target": "http://example.com/music/1999.mp3",
        "assigner": "http://example.com/org/sony-music",
        "action": "odrl:play",
        "assignee": "http://example.com/people/billie"
        },
        {
        "assignee": "http://example.com/people/murphy"
        "target": "http://example.com/music/1999.mp3",
        "assigner": "http://example.com/org/sony-music",
        "action": "odrl:play",
        }]
    }]
}
```

The processing model for Policies with multiple Assets, Parties, and Actions includes:

1. Replicate all Policy-level Assets in the enclosing Permission/Prohibition Rules.
2. Replicate all Policy-level Parties in the enclosing Permission/Prohibition Rules.
3. Replicate all Policy-level Actions in the enclosing Permission/Prohibition Rules.
4. Follow the processing model (defined above) to create *atomic* Rules.

3.1.2 Policy Provenance

Provenance attributes *MAY* be added to the Policy entity to support additional authenticity, integrity, and interpretation. Typically, these will be from external community vocabularies.

The following Dublin Core Metadata Terms [*DCTERMS*] *SHOULD* be used:

- dc:creator - the individual, agent, or organisation that authored the Policy.
- dc:issued - the date (and time) the Policy was first issued.
- dc:modified - the date (and time) the Policy was updated.
- dc:coverage - the jurisdiction under which the Policy is relevant.
- dc:replaces - the identifier (uid) of a Policy that this Policy supersedes.
- dc:isReplacedBy - the identifier (uid) of a Policy that supersedes this Policy.

The processing model for Policies with the above provenance properties include:

1. If a Policy contains the **dc:isReplacedBy** property, then the identified Policy *MUST* be retrieved and processed.

Example Use Case: The below Policy contains provenace properties that indicate who created it, when it was issued, which jurisdiction is relevant, and an older version of the Policy it replaces.

Example 8

```
{
    "@context": [{ "odrl": "http://www.w3.org/ns/odrl/2/",
                   "dc": "http://purl.org/dc/terms/"}],
    "type": "odrl:Agreement",
    "uid": "http://example.com/policy:8888",
    "dc:creator": "billie@example.com",
    "dc:issued": "2017-01-01:12:00",
    "dc:jurisiction: "Queensland, Australia",
    "dc:replaces: "http://example.com/policy:8887",
    "permission": [{ ... }]
}
```

3.1.3 Policy Conflict

The `conflict` attribute is used to establish strategies to resolve conflicts that arise from the merging of Policies or conflicts between Permissions and Prohibitions in the same Policy. Conflicts may arise when merging Policies about the same Asset and the resultant Actions are inconsistent. For example, one Policy stated that Assignee Fred had been granted an "exclusive" distribution Permission, and the other Policy said the same for Assignee Mary. In the same Policy, conflicts can also arise when the same Action is used in a Permission and a Prohibition.

The `conflict` attribute *MUST* take one of the following values:

- `perm`: the `Permission`s *MUST* override the Prohibitions
- `prohibit`: the `Prohibition`s *MUST* override the Permissions
- `invalid`: the entire Policy *MUST* be invalid if any conflict is detected

If the `conflict` attribute is not explicitly set, its default value of `invalid` will be used.

The processing model for Policies conflict strategies includes:

1. If a Policy has the `conflict` attribute of `perm` and there are detected conflicts between a Permission and a Prohibition, then the Permission *MUST* overide the Prohibition.
2. If a Policy has the `conflict` attribute of `prohibit` and there are detected conflicts between a Permission and a Prohibition, then the Prohibition *MUST* overide the Permission.
3. If a Policy has the `conflict` attribute of `invalid` and there are detected conflicts between a Permission and a Prohibition, then the entire Policy *MUST* be processed as invalid .

4. If a Policy has multiple `conflict` attribute values (for example, after a Policy merge) AND there are detected conflicts between a Permission and a Prohibition, then the entire Policy *MUST* be processed as invalid .

Example Use Case: Two Policies are associated to the same target Asset http://example.com/asset:1212. The first Policy http://example.com/policy:0001 allows to use the Asset. The second Policy http://example.com/policy:0002 allows for the display of the Asset, but it prohibits print. Both policies explicitly state how to deal with conflicts through the `conflict` attribute being set to `perm`. Hence the Permissions will always override any Prohibitions. In this use case, since the print Action is a subset of the use Action, there could be a conflict. However, the `perm` conflict strategy means that the use Permission will override the print Prohibition.

Example 9

```
{
"@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Set",
    "uid": "http://example.com/policy:0001",
    "conflict": "perm",
    "permission": [{
        "target": "http://example.com/asset:1212",
        "action": "odrl:use",
        "assigner": "http://example.com/owner:181"
    }]
}
```

Example 10

```
{
"@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Set",
    "uid": "http://example.com/policy:0002",
    "conflict": "perm",
    "permission": [{
        "target": "http://example.com/asset:1212",
        "action": "odrl:display",
        "assigner": "http://example.com/owner:182"
    }]
    "prohibition": [{
        "target": "http://example.com/asset:1212",
        "action": "odrl:print"
    }]
}
```

In the above use case, if the second Policy had the conflict value of `prohibit`, then the outcome would be a direct contradiction, and the result will be an invalid Policy.

3.1.4 Undefined Actions

The `undefined` attribute is used to indicate how to process unsupported Actions. That is, if an ODRL expression contains an Action that is not from a known (or supported) ODRL vocabulary, how should the Action be treated in the context of the whole ODRL Policy?

The `undefined` attribute *MUST* take one of the following values:

- `support`: the undefined Action is to be **supported** as part of the Policy – and the Policy remains valid
- `ignore`: the undefined Action is to be **ignored** and is not part of the Policy – and the Policy remains valid
- `invalid`: the undefined Action is unknown – and the entire Policy is **invalid**

In all cases, systems that process ODRL expressions *SHOULD* provide mechanisms that adequately address these three outcomes. That is, how the Action can be **supported**, or **ignored**, or the entire Policy is **invalid**.

If the `undefined` attribute is not explicitly set, its default value of `invalid` *SHALL* be used.

The processing model for undefined actions includes:

1. If a Policy has the `undefined` attribute of `support` and there are actions in the Policy now known to the processing system, then the processing system *MUST* support these actions as part of the valid Policy.
2. If a Policy has the `undefined` attribute of `ignore` and there are actions in the Policy now known to the processing system, then the processing system *MUST* disregard these actions as part of the valid Policy.
3. If a Policy has the `undefined` attribute of `invalid` and there are actions in the Policy now known to the processing system, then the processing system *MUST* process the entire Policy as invalid.
4. If a Policy has multiple `undefined` attribute values (for example, after a Policy merge) AND there are actions in the Policy now known to the processing system, then the entire Policy *MUST* be processed as invalid.

Example Use Case: A Policy of type Set states that the Asset can be *recorded*. The processing system does not understand this Action, and since the **undefined** attribute is **invalid**, then the entire Policy is deemed invalid.

Example 11

```
{
    "@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Set",
    "uid": "http://example.com/policy:8888",
    "undefined": "invalid",
    "permission": [{
        "target": "http://example.com/music/1999.mp3",
        "action": "http://example.com/ns/recorded"
        }]
    }]
}
```

3.1.5 Policy Inheritance

ODRL supports a simple inheritance mechanism in which a (child) Policy may inherit all the information structures of another (parent) Policy. The inheritance is aimed at *including* data structures between Policies.

The `inheritAllowed` attribute in the Policy entity is used to indicate if the Policy expression can be used in any inheritance relationship. If present, the value of the `inheritAllowed` attribute *MUST* take one of the following values:

- `true`: the Policy expression can be used for inheritance
- `false`: the Policy expression can not be used for inheritance

If the `inheritAllowed` attribute is not explicitly set, its default value of `true` will be used.

The following attributes *SHOULD* be used in a child Policy that is inheriting from a parent Policy in which that parent Policy *MUST* allow inheritance (via the `inheritAllowed` attribute) :

- `inheritFrom`: the identifier of the parent Policy from which this child Policy inherits from
- `inheritRelation`: the identifier of the relationship context of this inheritance structure

The `inheritFrom` association in the (child) Policy will uniquely identify (via a UID) the (parent) Policy from which the inheritance will be performed.

The `inheritRelation` attribute in the (child) Policy will uniquely identify (via a UID) the context of inheritance from the (parent) Policy. For example, this may indicate the business scenario, such as *subscription*, or prior arrangements between the Parties (that are not machine representable). Such terms *SHOULD* be defined in the ODRL vocabulary [*vocab-odrl*] or ODRL Profiles. For example, an Assigner and Assignee may have a historical arrangement related to the specific use of content they make available to each other. The business model (identified with a URI) is used in the `inheritRelation` attribute in their subsequent ODRL Policies they exchange. This will require the ODRL Policy to be interpreted with the additional information identified by the URI. For example, this may include additional permission Actions or constraints (etc) that is documented in their business model arrangement.

Both the `inheritFrom` association and `inheritRelation` attribute may be used independently.

The following restrictions apply when using inheritance:

- Single inheritance is only supported. (One parent Policy to one or more child Policy entities. No child Policy can inherit from two or more parent Policy entities.)
- Inheritance can be to any depth. (Multiple levels of children Policy entities.)
- Inheritance *MUST NOT* be circular.
- No state information is transferred from the parent Policy to the child Policy.

Example Use Case: Consider the below (parent) Policy that has been expressed primarily for inheritance purposes:

Example 12

```
    "@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Set",
    "uid": "http://example.com/policy:3333",
    "target": "http://example.com/asset:3333",
    "assigner": "http://example.com/boss:0001",
    "permission": [{
        "action": "odrl:use"
    }]
}
```

The below (child) Policy includes the inheritFrom attribute pointing to the above (parent) Policy. The (child) Policy also includes its own specific policy-level asset, and two Permission Rules.

Example 13

```
{
    "@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Agreement",
    "uid": "http://example.com/policy:4444",
    "inheritFrom": "http://example.com/policy:3333",
    "target": "http://example.com/asset:5555",
    "permission": [{
        "assignee": "http://example.com/guest:0001",
        "action": "odrl:display"
    }],
    "permission": [{
        "assignee": "http://example.com/guest:0002",
        "action": "odrl:print"
    }]
}
```

After the inheritance is performed - where the (parent) Policy information structures are added to the (child) Policy - the resultant Policy is shown below:

Example 14

```
{
    "@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Agreement",
    "uid": "http://example.com/policy:4444",
    "target": [
       "http://example.com/asset:5555",
       "http://example.com/asset:3333" ]
    "assigner": "http://example.com/boss:0001",
    "permission": [{
        "assignee": "http://example.com/guest:0001",
        "action": "odrl:display"
    }],
```

```
    "permission": [{
        "assignee": "http://example.com/guest:0002",
        "action": "odrl:print"
    }],
    "permission": [{
        "action": "odrl:use"
    }]
}
```

The processing model for ODRL Policy Inheritance includes:

1. A (child) Policy with an `inheritFrom` attribute *MUST* first verify that the (parent) Policy does not contain the `inheritAllowed` attribute with the value "false".
2. The (child) Policy *MUST* access the (parent) Policy and replicate the following in the (child) Policy:
    - All policy-level Assets, Parties, Actions.
    - All Permission and Prohibition Rules.
3. The (child) Policy can then be further expanded (into *atomic* Rules) by following the processing models defined in the Policy Structure section.

3.2 Asset

The Asset entity is the subject of an ODRL policy expression that Permissions and Prohibitions are applied to. The Asset entity can be any form of identifiable resource, such as data/information, content/media, applications, or services. Furthermore, it can be used to represent other `Asset` entities that are needed to undertake the Policy expression, such as with the Duty entity. The Asset entity is referred to by the Permission and/or Prohibition entities, and also by the Duty entity.

The Asset entity contains the following attribute:

- `uid`: the unique identification of the Asset (*REQUIRED*)
- `scope`: defines how the Asset shall be interpreted under different contexts. (*OPTIONAL*)

The identification of the Asset entity is a key foundation of the ODRL Policy language. In general, the `uid` *SHOULD* be a URI [*rfc3986*] representing the identifier for the Asset. There *MAY* also be other alternative identifers, or identification mechanisms that could be used. For example, to identify specific parts of an Asset, Media Fragments URI [*media-frags*] may be utilised. To identify groups or a range of Assets, then POWDER [*powder-primer*], URI Templates [*rfc6570*], or URI wildcards may be utilised.

There are some use cases where the ODRL Policy expression *MAY* be embedded inside the target Asset. In these cases, it *MAY* be more appropriate to provide, or infer, a link to the Asset entity (as the complete Asset `uid` may not be known at the time) through the local context. Use of such inference and context *MUST* be documented in the relevant ODRL Profile.

Since ODRL Policies could deal with any kind of asset, the ODRL Information Model does not provide additional metadata to describe Asset entities of particular media types. It is

recommended to use already existing metadata standards, such as Dublin Core Metadata Terms that are appropriate to the Asset type or purpose.

3.2.1 Relation

The Relation entity is an association class and can be used to link to an Asset from either Permission, Duty or Prohibition, indicating how the Asset *SHOULD* be utilised in respect to the entity that links to it.

The Relation entity contains the following attribute:

- `relation`: indicates the relationship of the Asset to the linked entity (*REQUIRED*)

The default value for the `relation` attribute is `target` which indicates that the Asset is the primary object to which the Permission, Duty or Prohibition actions apply.

Other values for the Relation entity *SHOULD* be defined in the ODRL Vocabulary [*vocab-odrl*] and ODRL Profiles.

3.2.2 Scope

The `scope` attribute *SHOULD* be used to indicate the context under which to interpret the Asset entity. The purpose of `scope` is to provide additional information about the Asset. For example, the Asset identifier may refer to a resource which many characteristics, but the scope may limit that to only one specific characteristic.

The `scope` attribute URI values *SHOULD* be defined in the ODRL vocabulary [*vocab-odrl*] and ODRL Profiles.

Example Use Case: The Party http//example.com/guest:0001 needs to display the target Asset http://example.com/asset:3333, but she needs to be granted with the Permission to do so. This request can be implemented through a Policy of the type Request asking for the Permission to display Asset http://example.com/asset:3333.

Example 15

```
{
 "@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Request",
    "uid": "http://example.com/policy:3333",
    "permission": [{
        "target": "http://example.com/asset:3333",
        "action": "odrl:display",
        "assignee": "http://example.com/guest:0001"
    }]
}
```

Example Use Case: The Party needs to define a Policy exploiting multiple Asset entities. The index Permission is granted to the target Asset http://example.com/archive1011. The collection Asset http://example.com/x/database specifies which database the index outcome should be stored in.

Example 16

```
{
 "@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Set",
    "uid": "http://example.com/policy:1011",
    "permission": [{
        "target": "http://example.com/archive1011",
        "x:collection": "http://example.com/x/database",
        "action": "odrl:index"
    }]
}
```

Example Use Case: The Policy includes a target Asset http://example.com/media-catalogue that has a scope of http://example.com/imt/jpeg (which, in this case, provides additional context on what characteristics the Asset *MUST* hold).

Example 17

```
{
"@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Offer",
    "uid": "http://example.com/policy:4444",
    "permission": [{
        "target": [{
                "uid": "http://example.com/media-catalogue",
                "scope": "http://example.com/imt/jpeg"
        }],
        "action": "odrl:use",
        "assigner": "http://example.com/user88"
    }]
}
```

3.3 Party

The Party entity is the object of an ODRL Policy and can be any form of identifiable entity, such as a person, group of people, organisation, or agent. An agent is a person or thing that takes an active Role or produces a specified effect. The Party performs (or does not perform) Actions or has a function in a Duty (i.e., assigns the Party to the Rule by associating it with the Role it plays in the Rule).

The Party entity contains the following attributes:

- `uid`: the unique identification of the Party (*REQUIRED*)
- `function`: defines the Roles fulfilled by a Party in relation to a Rule (*REQUIRED*)
- `scope`: defines how the Role shall be interpreted under different contexts. (*OPTIONAL*)

The ODRL Information Model does not provide additional metadata for the Party element. It is recommended to use already existing metadata standards, such as the W3C vCard Ontology [*vcard-rdf*] or FOAF [*foaf*], that are appropriate to the Party type or purpose.

The Role entity is used to associate the Party entity with the relevant Permission, Prohibition, and Duty entities.

3.3.1 Role

The Role entity is an association class and can be used to link to a Party from either Permission, Duty or Prohibition, indicating which Role the Party takes with respect to the entity that links to it.

The Role entity contains the following attributes:

- `function`: the functional role the Party takes (*REQUIRED*)

The `function` attribute *MUST* support the following values:

- `assigner`: indicates that the Party has assigned the associated Permission, Duty, or Prohibition. In other words, the Party grants a Permission or requires a Duty to be performed or states a Prohibition.
- `assignee`: indicates that the Party has been assigned the associated entity, i.e., they are granted a Permission or required to perform a Duty or have to adhere to a Prohibition.

Other values for the `function` attribute *SHOULD* be defined in the ODRL vocabulary [*vocab-odrl*] and ODRL Profiles.

3.3.2 Scope

The `scope` attribute *SHOULD* be used to indicate the context under which to interpret the Party entity. The purpose of `scope` is to provide additional information about the Party that may also reduce the extent of the Party identifier. For example, the Party identifier may refer to all users at a University, but the scope may limit that to only staff members at the University.

The `scope` attribute *SHOULD* take one of the following values:

- `individual`: indicates that the Party entity is a single individual. The linked Permission, Duty or Prohibition is applicable for that individual only.
- `group`: indicates that the Party entity represents a group. The group consisting of many individual members. The linked Permission, Duty or Prohibition is applicable for each member of that group. For example, a Permission to play a movie 5 times is valid for each Party member or the Duty to pay 3 EUR has to be fulfilled by each Party member.

Other URI values for the `scope` attribute *SHOULD* be defined in the ODRL vocabulary [*vocab-odrl*] and ODRL Profiles.

Example Use Case: The target Asset http://example.com/myPhotos:BdayParty are photos posted to a Social Network site by the Assigner and owner of the photos http://example.com/user44. The Assignee is a Party http://example.com/user44/friends,

and represents all the friends of the Assigner. The scope of the Assignee(s) has been set to http://example.com/people/age/18+ (which, in this case provides additional context on what characteristics the Assigee *MUST* hold).

Example 18

```
{
"@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Agreement",
    "uid": "http://example.com/policy:4444",
    "permission": [{
        "target": "http://example.com/myPhotos:BdayParty",
        "action": "odrl:display",
        "assigner": "http://example.com/user44",
        "assignee": [{
                "uid": "http://example.com/user44/friends",
                "scope": "http://example.com/people/age/18+"
        }]
    }]
}
```

3.4 Permission

The Permission entity indicates the Actions permitted to be performed on the Asset.

An ODRL policy expression *MAY* contain at least one Permission. It is important to verify the semantics of the Policy `type` attribute as this *SHOULD* indicate additional constraints on the Policy expression semantics.

The Permission entity has the following relations:

- `Asset`: the Permission entity *MUST* refer to an Asset (where at least one, and only one, `relation` value is `target`) on which the linked Action *SHOULD* be performed (*REQUIRED*)
- `Action`: the Permission entity *MUST* refer to *exactly one* Action that indicates the granted operation on the target Asset (*REQUIRED*) (Note: this is after the Rule has folowed the processing models defined in the Policy Structure section.)
- `Party`: the Permission *MAY* refer to one or more Party entities linked via the Role entity (*OPTIONAL*)
- `Constraint`: the Permission *MAY* refer to one or more Constraints which affect the validity of the Permission - it only becomes effective if all of the referred Constraints are true (*OPTIONAL*). For example; the Action play is only permitted for a certain period of time.
- `Duty`: the Permission *MAY* refer to one or more Duty entities that indicate a requirement that *SHOULD* be fulfilled in return for receiving the Permission (*OPTIONAL*)

Example Use Case: The ticket Policy expresses the play Permission for the target Asset http//example.com/game:9090, stating that the ticket is valid until the end of the year 2016. Any valid holder of this ticket may exercise this Permission.

Example 19

```
{
 "@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Ticket",
    "uid": "http://example.com/policy:9090",
    "permission": [{
        "target": "http://example.com/game:9090",
        "action": "odrl:play",
        "constraint": [{
            "leftOperand": "odrl:dateTime",
            "operator": "odrl:lteq",
            "rightOperand": "2016-12-31"
        }]
    }]
}
```

3.5 Prohibition

The Prohibition entity indicates the Actions that are prohibited to be performed on the Asset.

The Prohibition entity has the following relations:

- `asset`: the Prohibition entity *MUST* refer to an Asset (where at least one, and only one, `relation` value is `target`) on which the Action is prohibited (*REQUIRED*) (Note: this is after the Rule has folowed the processing models defined in the Policy Structure section.)
- `action`: the Prohibition entity *MUST* refer to *exactly one* Action that is prohibited (*REQUIRED*)
- `party`: the Prohibition *MAY* refer to one or more Party entities linked via the Role entity (see Section 2.3.1) (*OPTIONAL*)
- `constraint`: the Prohibition *MAY* refer to one or more Constraints which affect the validity of the Prohibition - it only becomes effective if all of the referred Constraints are true (*OPTIONAL*). For example; the Action display is only prohibited for a certain country.

Example Use Case: The owner and Assigner of a target Asset http://example.com/photoAlbum:55 needs an Agreement Policy expressing with both a Permission and a Prohibition. Then Assigner Party http://example.com/MyPix:55 assigns the Permission display to the Assignee Party http://example.com/assignee:55 at the same time they are prohibited from archiving the target Asset. Additionally, in case of any conflicts between Permissions and Prohibitions, the conflict attribute is set to `perm` indicating that the Permission entity will take precedence.

Example 20

```
{
"@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Agreement",
    "uid": "http://example.com/policy:5555",
    "conflict": "odrl:perm",
```

```
    "permission": [{
        "target": "http://example.com/photoAlbum:55",
        "action": "odrl:display",
        "assigner": "http://example.com/MyPix:55",
        "assignee": "http://example.com/assignee:55"
    }],
    "prohibition": [{
        "target": "http://example.com/photoAlbum:55",
        "action": "odrl:archive",
        "assigner": "http://example.com/MyPix:55",
        "assignee": "http://example.com/assignee:55"
    }]
}
```

3.6 Duty

A Permission entity *MAY* include a Duty entity that indicates a requirement that *MUST* be satisfied in order to undertake the Permission. A Duty *MUST* only be associated with a Permission.

The Duty entity contains the following attributes:

- `uid`: a unique identification for this Duty. Used to refer a single Duty to multiple Permission entities (*OPTIONAL*)

The `Duty` entity has the following relations:

- `Action`: indicates the requirement that *MUST* be performed (*REQUIRED*). Note: It is assumed that the assigned Party has the appropriate permissions to perform this action.
- `Party`: a Duty *MAY* refer to Party entities with different Roles. If no explicit Party is linked to as Assignee or Assigner, the Parties with the respective Roles are taken from the referring Permission. (*OPTIONAL*)
- `Asset`: a Duty entity *MAY* refer to an Asset (where at least one, and only one, `relation` value is `target`) related to satisfying the Duty. (*OPTIONAL*) For example, a `nextPolicy` Action must be linked to the identifier of a target Policy Asset.
- `Constraint`: a Duty *MAY* refer to one or more Constraints which affect the fulfillment of the Duty - it is only fulfilled if all of the referred Constraints are true (*OPTIONAL*). For example; the Duty Action payAmount is only fulfilled if a specific amount is paid.

If a Permission refers to several Duty entities, *all of them MUST* be satisfied for the Permission to be undertaken. If several Permission entities refer to one Duty, then the Duty only has to be satisfied *once* for all the Permission entities.

Even though a Duty entity is mandatory, the ODRL model does not specify any conditions on *WHEN* the Duty Action *MUST* be performed. Such business rules *MAY* be expressed through additional Constraints. For example, a Policy may state that you can play a music file for a payment of $5.00. This does not indicate when the $5.00 should be paid, as different business rules may apply such as montly invoicing.

Example Use Case: The Party http://example.com/assigner:sony makes an Offer to play the music file http://example.com/music/1999.mp3. There are two Duties that both must be satisfied for the `compensate` requirement. The first is the `payAmount` of $AUD5.00 and the second is the `event` of this http://www.w3.org/ns/odrl/2/policyUsage must not have occured yet.

Example 21

```
{
    "@context": {
     "odrl": "http://www.w3.org/ns/odrl/2/"
     },
     "type": "odrl:Offer",
     "uid": "http://example.com/policy:88",
     "permission": [{
         "assigner": "http://example.com/assigner:sony",
         "target": "http://example.com/music/1999.mp3",
         "action": "odrl:play",
         "duty": [{
             "action": "odrl:compensate",
             "constraint": [{
                 "leftOperand": "odrl:payAmount",
                 "operator": "odrl:eq",
                 "rightOperand": "5.00",
                 "unit": "http://cvx.iptc.org/iso4217a/AUD"
             }],
             "constraint": [{
                 "name": "odrl:event",
                 "operator": "odrl:lt",
                 "rightOperand": "odrl:policyUsage"
             }]
         }]
     }]
}
```

Example Use Case: The Party http://example.com/assigner:77 needs to set a Privacy Policy for some data about it. The target Asset http://example.com/personal-data:77 of this Policy is personal data, and the Assignee http://example.com/gov:health:au is allowed to distribute the Asset only for the purpose of contacting the subject of the Personal Data. The purpose value is taken from the P3P privacy purpose vocabulary. Additionally, the Party has stipulated that the Assignee must delete the Asset after a 30 day period, i.e., retention policy.

Example 22

```
{
"@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    },
    "type": "odrl:Privacy",
    "uid": "http://example.com/policy:7777",
    "permission": [{
        "target": "http://example.com/personal-data:77",
        "action": "odrl:distribute",
        "constraint": [{
            "name": "odrl:purpose",
            "operator": "odrl:eq",
            "rightOperand": "http://www.w3.org/2002/01/P3Pv1:contact"
        }],
        "assigner": "http://example.com/assigner:77",
```

```
        "assignee": "http://example.com/gov:health:au",
        "duty": [{
            "action": "odrl:delete",
            "target": "http://example.com/personal-data:77",
            "constraint": [{
                "leftOperand": "odrl:dateTime",
                "operator": "odrl:eq",
                "rightOperand": "P30D"
            }]
        }]
    }]
}
```

3.7 Action

The Action entity (when related to a Permission entity) indicates the operations (~~e.g., play~~, ~~copy, etc.~~) that the Assignee (i.e., the consumer) is *permitted* to perform on the related Asset linked to by Permission. When related to a Prohibition, the Action entity indicates the operations that the Assignee (again the consumer) is *prohibited* to perform on the Asset linked to by Prohibition. Analogously, when related to a Duty, it indicates the operation to be performed.

Action contains the following attribute:

- `name`: indicates the Action entity term (*REQUIRED*)

As its value, the `name` attribute *SHOULD* take one of a set of Action *names* which are formally defined in Profiles. The ODRL vocabulary [*vocab-odrl*] defines a standard set of potential terms that *MAY* be used. Communities will develop new (or extend existing) Profiles to capture additional and refined semantics.

Example Use Case: The Party needs to define an Offer policy granting a Permission about the target Asset http://example.com/music:1012, namely the permission to play the Asset.

Example 23

```
{
"@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    },
    "type": "odrl:Offer",
    "uid": "http://example.com/policy:1012",
    "permission": [{
            "target": "http://example.com/music:1012",
            "action": "odrl:play"
     }]
}
```

3.8 Constraint

The Constraint entity indicates limits and restrictions to the Permission, the Prohibition and the Duty entities. Constraints express a Rule for comparing two operands by one operator. If the two operands match the result of the Constraint is **true**.

For example, the "number of usages" (`leftOperand`) must be "smaller than" (`operator`) the "number 10" (`rightOperand`).

If multiple Constraint entities are linked to the same Permission, Prohibition, or Duty entity, then all of the Constraint entities *MUST* be satisfied. That is, all the Constraint entities are (boolean) *and*ed. In the case where the same Constraint is repeated, then these *MUST* be represented as a single Constraint entity using an appropriate `operator` value (for example, `isAnyOf`).

The `Constraint` entity contains the following attributes:

- `leftOperand`: a name that identifies the left operand of the operation (*REQUIRED*)
- `operator`: an operator function (*REQUIRED*)
- `rightOperand`: the right operand of the operation (*REQUIRED* if no rightOperandReference)
- `rightOperandReference`: the right operand of the operation (*REQUIRED* if no rightOperand)
- `dataType`: the datatype of the rightOperand (*OPTIONAL*)
- `unit`: the units of the rightOperand (*OPTIONAL*)
- `status`: the current value of the left operand (*OPTIONAL*)

The `leftOperand` identifies the left operand of the logic operation for the Constraint, it *SHOULD* include the entity it constrains and how its value for a comparison has to be retrieved/generated. It *MAY* include the datatype of the value and it *MAY* include references to resources for retrieving/generating the value.

The `operator` identifies the comparative operation such as "greater than" or "equal to".

The actual value of the constraint that is to be compared to the `leftOperand` is either represented with the `rightOperand` or `rightOperandReference`. The `rightOperand` represents the literal value (such as "10" or "http://example.com/c100") and `rightOperandReference` represents a URI that *MUST* be dereferenced to obtain the actual value. Only one of these *MUST* appear in the Constraint.

The `dataType` indicates the type of the `rightOperand/Reference`, such as "decimal" or "datetime" and the `unit` indicates the unit value of the `rightOperand/Reference`, such as "EU dollars".

The `status` provides the current value of the Constraint variable (i.e. current value of `leftOperand`). This is useful in cases where the current status of Constraints needs to be captured and expressed in the ODRL Information Model.

Example Use Case: The Party http://example.com/myPix:6161 wants to assign the Permission distribute directly to the potential buyer of the Permission who will pay 100 EUR for this grant. The distribute Permission is also constrained to a specific country, ~~i.e., Italy.~~ The potential Assignee may then distribute the target Asset http://example.com/wallpaper:1234 according to the nextPolicy target Asset linked directly from this Duty. In this case, the next Policy Asset stipulates that the potential Assignee may only offer the display Permission to downstream consumers.

Example 24

```
{
"@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Offer",
    "uid": "http://example.com/policy:6161",
    "permission": [{
        "target": "http://example.com/wallpaper:1234",
        "assigner": "http://example.com/myPix:6161",
        "action": "odrl:distribute",
        "constraint": [{
            "leftOperand": "odrl:spatial",
            "operator": "odrl:eq",
            "rightOperand": "http://www.itu.int/tML/tML-ISO-3166:it"
        }],
        "duty": [
            {
                "action": "odrl:compensate",
                "constraint": [{
                    "leftOperand": "odrl:payAmount",
                    "operator": "odrl:eq",
                    "rightOperand": "100.00",
                    "dataType": "http://www.w3.org/2001/XMLSchema#decimal",
                    "unit": "http://cvx.iptc.org/iso4217a/EUR"
                }]
            },
            {
                "action": "odrl:nextPolicy",
                "target": "http://example.com/policy:7171"
            }
        ]
    }].
}
```

Example 25

```
{
    "@context": {
    "odrl": "http://www.w3.org/ns/odrl/2/"
    }
    "type": "odrl:Set",
    "uid": "http://example.com/policy:7171",
    "permission": [{
        "target": "http://example.com/wallpaper:1234",
        "action": "odrl:display"
    }]
}
```

belongs to this example

3.8.1 Constraint Relations

Constraint objects *MAY* also be used as both of the values for the `leftOperand` and `rightOperand` of a Constraint expression. This supports more complex constraint relations and allows for two constraints to be compared and processed accordingly. The two constraints *MUST* be *atomic* Constraints, that is, not a constraint that includes constraint relations.

The applicable `operators` that can be used for constraint relations are:

- `or` - at least one of the Constraints (in the left or right operand) *MUST* be satisfied

- `xor` - only one of the Constraints (in the left or right operand) *MUST* be satisfied
- `and` - both of the Constraints (in the left or right operand) *MUST* be satisfied

The processing model for Constraint relations includes:

1. If the Constraint `leftOperand` refers to a Constraint object, then the `rightOperand` *MUST* also refer to a Constraint object.
2. The Constraint objects both *MUST* be *atomic* Constraints.
3. The Constraint `operator` *MUST* only be of the values; `or`, `xor`, `and`.
4. If the Constraint `operator` is `or` then at least one of the Constraints (in the left or right operands) *MUST* be satisfied for the entire Constraint to be satisfied.
5. If the Constraint `operator` is `xor` then at only one of the Constraints (in the left or right operands) *MUST* be satisfied for the entire Constraint to be satisfied.
6. If the Constraint `operator` is `and` then both of the Constraints (in the left and right operands) *MUST* be satisfied for the entire Constraint to be satisfied.

Example Use Case: The Policy below shows the use of Constraint relations. The Permission to play the target asset can either be a maximum of 100 times, or unlimited play until the end of 2017.

Example 26

```
{
    "@context": {
     "odrl": "http://www.w3.org/ns/odrl/2/"
     }
     "type": "odrl:Offer",
     "uid": "http://example.com/policy:88",
     "permission": [{
         "target": "http://example.com/book/1999.mp3",
         "assigner": "http://example.com/org/paisley-park",
         "action": "odrl:play",
         "constraint": [{
                 "leftOperand": "http://example.com/policy:88/C1",
                 "operator": "odrl:xor",
                 "rightOperand": "http://example.com/policy:88/C2"
             }],
     }],
    "constraint": {
        "@id": "http://example.com/policy:88/C1"
        "leftOperand": "odrl:count",
        "operator": "odrl:lteq",
        "rightOperand": "100"
    },
    "constraint": {
        "@id": "http://example.com/policy:88/C2"
        "leftOperand": "odrl:dateTime",
        "operator": "odrl:lteq",
        "rightOperand": "2017-12-31"
    }
}
```