2025

Custom Ontologies with the University of Southampton ODRL Editor

AUTHOR: CHRISTOPHER MAIDENS

Reviewers

Jaime Osvaldo Salas

Semih Yumusak

Paolo Pareti

1 Custom Ontology Creation Guidance

1.1 Introduction

The ODRL Editor is supplied with a default underpinning ontology (an integration of ODRL and DPV). This is used to drive the vocabulary made available to the user when creating new policies.

The ODRL editor also provides the user with the ability to add domain specific ontologies that may be used to tailor the expressivity of the vocabulary to express specific business requirements.

The editor will allow the user to load custom ontologies using either Turtle (ttl) or RDF/XML (xml) files.

The text below provides some detail on how the editor queries the underpinning ontologies to extract to relevant vocabulary. This is then used to provide insight into how custom ontologies can be created that will tailor the editor's behaviour.

Some simple examples are also given that can be used as a basis for simple definitions.

Ontology creation should follow basic analysis of the business process(es) and related vocabulary that subsequent policies will serve.

1.2 Adding a New Action

1.2.1 Action Discovery

The editor will retrieve the list of possible Actions from the base ontology and additional custom ontologies using the following SPARQL query

```
SELECT DISTINCT ?action ?label  ?sub_action ?sub_label
      WHERE {

          ?sub_action   (rdf:type /  rdfs:subClassOf?)   odrl:Action .
          ?sub_action ( rdfs:label | skos:prefLabel ) ?sub_label .

          ?sub_action   odrl:includedIn | rdfs:subClassOf   ?action .
          ?action    ( rdfs:label | skos:prefLabel )    ?label .

      }
  """
```

From this query a hierarchical list of possible Actions is provided to the editor user - interface and this will be formatted and displayed in the Rule dropdown elements.

1.2.2 Custom Actions

An example of adding an Action is shown below (this is presented through a TTL fragment)

```
:CorrelationAnalysis a odrl:Action , skos:Concept ;
        rdfs:isDefinedBy <http://example.org> ;
        rdfs:label "Correlation Analysis"@en ;
        odrl:includedIn dpv-owl:Analyse .
```

Some commentary on this is given below

In this example the user has declared *CorrelationAnalysis* as *rdf:type odrl:Action*

The user may well have declared it as *rdfs:SubClassOf odrl:Action* or indeed *rdfs:SubClassOf* another user defined *rdf:type odrl:Action*. This would depend on the user's business modelling requirements. Any of these formulations would be discovered by the editor.

CorrelationAnalysis is also includedIn *dpv-owl:Analyse. F*or more specific detail on odrl:includedIn see ODRL Vocabulary & Expression 2.2 however broadly this is creating usable ancestor relationships. That is "An Action (except for use and transfer) MUST

have one includedIn property value (of type Action) to transitively assert this Action that encompasses its operational semantics." (see Ref 2.4)

In summary the user has the following options for a new action:

- The user adds a triple {new_action} rdf:type odrl:Action.

- The user defines their own class of actions {user_defined_actions}, add the triple {user_defined_actions} rdfs:subClassOf odrl:Action, and the triple {new_action} rdf:type {user_defined_actions}.

- The user adds a triple {new_action} rdfs:subClassOf {another_action} where {another_action} is of type odrl:Action.

And also

- Same as above but odrl:includedIn instead of rdfs:subClassOf.

1.2.3 Action Property Discovery

For a given selected Action the editor will populate a list of possible left operands to be used in associated Refinements (if any).

```
SELECT DISTINCT ?property ?label
      WHERE {
        {class_uri} rdf:type?   /  ( odrl:includedIn? | skos:broader? | rdfs:subClassOf* )  ?class .
        ?property ( rdfs:domain | schema:domainIncludes | dcam:domainIncludes ) /
                                                ( skos:broader | rdfs:subClassOf)? ?class ;
                ( rdfs:label | skos:prefLabel ) ?label .
        FILTER ( ?property NOT IN ( odrl:includedIn, odrl:implies, odrl:hasPolicy) )
      }
"""
```

From this query a list of possible properties is provided to the editor user interface and this will be formatted and displayed in the left operand dropdown of the refinement dialogue

1.2.4 Custom Action Properties

An example of adding an Action with a custom action property is shown below (this is presented through a TTL fragment)

```
:GridBasedClustering a odrl:Action , skos:Concept , dpv-owl:Processing ;
          rdfs:isDefinedBy <http://example.org> ;
          rdfs:label "Grid-based Clustering"@en ;
          odrl:includedIn dpv-owl:Analyse .
:hasThresholdDensity rdfs:domain :GridBasedClustering.
```

Also for completeness

```
:hasThresholdDensity a odrl:LeftOperand , owl:NamedIndividual , skos:Concept ;
        rdfs:isDefinedBy <http://example.org> ;
        rdfs:label "has threshold density"@en .
```

Some commentary on this is given below

In this example the user has declared *:hasThresholdDensity rdfs:domain :GridBasedClustering*. (alongside the Action *GridBasedClustering*).

By placing this in the *rdfs:domain* of *GridBasedClustering* then *hasThresholdDensity* is found through the query above and will be displayed as an option in the refinement dialogue.

The query provides some additional flexibility in tailoring the precise property semantics (i.e. *(rdfs:domain | schema:domainIncludes | dcam:domainIncludes) (skos:broader | rdfs:subClassOf)?*) however the example given provides a simple direct method of declaration. *skos:broader* and it's inverse *skos:narrower*, provides additional approaches to clarifying hierarchical semantic relationships (see SKOS Simple Knowledge Organization System Namespace Document - HTML Variant, 18 August 2009 Recommendation Edition).

For ODRL the refinements for Actions are considered as type *odrl:constraint* (itself having a structure of { *odrl:leftOperand, odrl:Operator*, *odrl:rightOperand*}. For this reason, the typing of *hasThresholdDensity* is included above.

1.3 Adding a New Actor

1.3.1 Actor Discovery

The editor will retrieve the list of possible Actors from the base ontology and additional custom ontologies using the following SPARQL query

```
SELECT DISTINCT ?actor ?label ?sub_actor ?sub_label

      WHERE {
         ?actor rdfs:subClassOf* dvpowl:Entity .
         ?actor skos:prefLabel|rdfs:label ?label .

         ?sub_actor rdfs:subClassOf ?actor .
         ?sub_actor skos:prefLabel|rdfs:label ?sub_label .

      }
   """
```

From this query a hierarchical list of possible Actors is provided to the editor and this will be formatted and displayed in the Rule dropdown elements.

1.3.2 Custom Actors

An example of adding an Actor is shown below (this is presented through a TTL fragment)

```
:PartTimeEmployee a rdfs:Class , skos:Concept ;
        rdfs:subClassOf dpv-owl:Employee ;
        rdfs:label "part-time employee"@en ;
        skos:broader dpv-owl:Employee .
```

Some commentary on this is given below

In this example the user has declared *PartTimeEmployee* as *rdf:type rdfs:Class*

Further they declared *PartTimeEmployee* as *rdfs:subClassOf dpv-owl:Employee*

In this case this has also declared *PartTimeEmployee* as *skos:broader dpv-owl:Employee* (for more specific detail on the semantic hierarchical relation *skos:broader* (see SKOS Simple Knowledge Organization System Namespace Document - HTML Variant, 18 August 2009 Recommendation Edition)

The *rdfs:subClassOf dpv-owl:Employee* formulation would be discovered by the editor (as *dpv-owl:Employee* is itself a descendant of *dvpowl:Entity* via *Employee->HumanSubject->LegalEntity->Entity*).

1.3.3 Actor Property Discovery

For a given selected Actor the editor will populate a list of possible left operands to be used in associated Refinements (if any).

```
SELECT DISTINCT ?property ?label
      WHERE {
        {class_uri} rdf:type?   /  ( odrl:includedIn? | skos:broader? | rdfs:subClassOf* )  ?class .
        ?property ( rdfs:domain | schema:domainIncludes | dcam:domainIncludes ) /
                                                ( skos:broader | rdfs:subClassOf)? ?class ;
                ( rdfs:label | skos:prefLabel ) ?label .
        FILTER ( ?property NOT IN ( odrl:includedIn, odrl:implies, odrl:hasPolicy) )
      }
"""
```

From this query a list of possible properties is provided to the editor and this will be formatted and displayed in the left operand dropdown of the refinement dialogue

1.3.4 Custom Actor Properties

An example of adding an Actor is with a custom property is shown below (this is presented through a TTL fragment)

```
:ContractEmployee a rdfs:Class , skos:Concept ;
          rdfs:subClassOf dpv-owl:Employee ;
          rdfs:label "part-time employee"@en ;
          skos:broader dpv-owl:Employee .
:hasWorkRights rdfs:domain :ContractEmployee .
```

Also for completeness

```
: hasWorkRights a odrl:LeftOperand , owl:NamedIndividual , skos:Concept ;
          rdfs:isDefinedBy <http://example.org> ;
          rdfs:label "has work rights"@en .
```

Some commentary on this is given below

In this example the user has declared *:hasWorkRights rdfs:domain :ContractEmployee*. (alongside the Actor *ContractEmployee*).

By placing this in the *rdfs:domain* of *ContractEmployee* then *hasWorkRights* is found through the query above and will be displayed as an option in the refinement dialogue.

The query provides some additional flexibility in tailoring the precise property semantics (i.e. *(rdfs:domain | schema:domainIncludes | dcam:domainIncludes)* *(skos:broader | rdfs:subClassOf)?*) however the example given provides a simple direct method of declaration.

For ODRL the refinements for Actors are considered as type *odrl:constraint* (itself having a structure of {leftOperand, Operator, rightOperand}. For this reason the typing of *hasWorkRights* is included above.

1.4 Adding a New Purpose

1.4.1 Purpose Discovery

The editor will retrieve the list of possible Purposes from the base ontology and additional custom ontologies using the following SPARQL query

```
SELECT DISTINCT ?purpose ?label  ?sub_purpose ?sub_label
      WHERE {

          ?sub_purpose rdf:type dvpowl:Purpose .
          ?sub_purpose ( rdfs:label | skos:prefLabel ) ?sub_label.

          ?sub_purpose ( skos:broader | rdfs:subClassOf) ?purpose .
          ?purpose ( rdfs:label | skos:prefLabel ) ?label .


      }
  """
```

From this query a a hierarchical list of possible Purposes is provided to the editor and this will be formatted and displayed in the Rule dropdown elements.

1.4.2 Custom Purposes

An example of adding a Purpose is shown below (this is presented through a TTL fragment)

```
:MedicalResearch a dpv-owl:Purpose , skos:Concept ;
          rdfs:isDefinedBy <http://example.org> ;
          rdfs:label "Medical Research"@en ;
          skos:prefLabel "Medical Research"@en ;
          skos:broader dpv-owl:ScientificResearch .
```

Some commentary on this is given below

In this example the user has declared *MedicalResearch* as *rdf:type dpv-owl:Purpose* and so is discovered by the editor.

In this case this has also declared *MedicalResearch* as *skos:broader dpv-owl:Employee* (for more specific detail on the semantic hierarchical relation *skos:broader* see SKOS Simple Knowledge Organization System Namespace Document - HTML Variant, 18 August 2009 Recommendation Edition)

1.4.3 Purpose Property Discovery

For a given selected Purpose the editor will populate a list of possible left operands to be used in associated Refinements (if any).

```
SELECT DISTINCT ?property ?label
      WHERE {
        {class_uri} rdf:type?   /  ( odrl:includedIn? | skos:broader? | rdfs:subClassOf* )  ?class .
        ?property ( rdfs:domain | schema:domainIncludes | dcam:domainIncludes ) /
                                                ( skos:broader | rdfs:subClassOf)? ?class ;
                ( rdfs:label | skos:prefLabel ) ?label .
        FILTER ( ?property NOT IN ( odrl:includedIn, odrl:implies, odrl:hasPolicy) )
      }
"""
```

From this query a list of possible properties is provided to the editor and this will be formatted and displayed in the left operand dropdown of the refinement dialogue

1.4.4 Custom Purpose Properties

An example of adding an Purpose is with a custom property is shown below (this is presented through a TTL fragment)

```
:MedicalResearch a dpv-owl:Purpose , skos:Concept ;
        rdfs:isDefinedBy <http://example.org> ;
        rdfs:label "Medical Research"@en ;
        skos:prefLabel "Medical Research"@en ;
        skos:broader dpv-owl:ScientificResearch .
:hasSponsor rdfs:domain :MedicalResearch .
```

Also for completeness

```
: hasSponsor a odrl:LeftOperand , owl:NamedIndividual , skos:Concept ;
        rdfs:isDefinedBy <http://example.org> ;
        rdfs:label "is sponsored by"@en .
```

Some commentary on this is given below

In this example the user has declared *:hasSponsor rdfs:domain :MedicalResearch*. (alongside the Purpose *MedicalResearch*).

By placing this in the *rdfs:domain* of *MedicalResearch* then *hasSponsor* is found through the query above and will be displayed as an option in the refinement dialogue.

The query provides some additional flexibility in tailoring the precise property semantics (i.e. *(rdfs:domain | schema:domainIncludes | dcam:domainIncludes)* *(skos:broader | rdfs:subClassOf)?*) however the example given provides a simple direct method of declaration.

For ODRL the refinements for Purposes are considered as type *odrl:constraint* (itself having a structure of {leftOperand, Operator, rightOperand}. For this reason the typing of *hasSponsor* is included above.

1.5 Adding a New Constraint

1.5.1 Constraint Left Operand Discovery

The editor will retrieve the list of possible left operands for Constraints from the base ontology and additional custom ontologies using the following SPARQL query

```
SELECT DISTINCT ?leftoperand ?label
        WHERE {
          ?leftoperand rdf:type odrl:LeftOperand ;
                ( rdfs:label | skos:prefLabel ) ?label .
        }
    """
```

From this query a hierarchical list of possible left operands for Constraints is provided to the editor and this will be formatted and displayed in the Rule dropdown elements.

1.5.2 Custom Constraint Left Operands

An example of adding a left operand for a constraint is shown below (this is presented through a TTL fragment)

```
:hasThresholdDensity a odrl:LeftOperand , owl:NamedIndividual , skos:Concept ;
        rdfs:isDefinedBy <http://example.org> ;
        rdfs:label "has threshold density"@en .
```

Some commentary on this is given below

In this example the user has declared *hasThresholdDensity* as *rdf:type hasThresholdDensity* and so is discovered by the editor.

2 Creating Policies

2.1 Introduction

This section should be read in conjunction with ODRL Information Model 2.2 and the ODRL Editor User Interface instructions.

It is intended to provide guidance in using the ODRL fragment developed with University of Southampton and the EU UPCAST project. It is intended to further clarify how JSON-LD policy 'documents' will be created from the knowledge recorded in the editor user interface.

Detailed data references are described through JSON-LD fragments

Policy creation should follow basic analysis of the business process(es) and related vocabulary/grammar that subsequent policies will serve.

that the policy will serve.

2.2 The Target

A policy is associated with a ***single*** Target.

This is used to identify a specific data asset.

The Rules (see below) are used to define the Policy

2.3 Rules

The basic building block of a policy is the Rule.

There may be one or more Rules that taken together are used to define the Policy.

There are three Rule types:

- Permission
- Prohibition
- Obligation

The Rule Type 'Duty' is currently also displayed in the user interface (for future development) but this should not be used.

The effect of multiple Rules is cumulative (that is the effects are AND ed).

Each Rule is made up of the following elements

- Actor (see ref 2.3)
    - Defines who this rule applies to .
    - See "odrl:Assignee" and "@type" : "odrl:PartyCollection" .
    - The value is defined directly or in element "odrl:source" : *URI* when of "@type" : "odrl:PartyCollection".
    - There may be none, one or many "odrl:refinement" (of type "odrl:constraint" - see Constraint below).
- Action (see ref 2.4)
    - Defines what action (on the Target) this rule applies to
    - See "odrl:action".
    - The value is defined directly or in element "rdf:value" : *IRI* (if refinements defined for the action – see below).
    - There may be none, one or many "odrl:refinement" (of type "odrl:constraint" - see Constraint below).
    -
- Purpose
    - Defines why the action (on the Target) is being applied.
    - See "odrl:purpose" (https://www.w3.org/TR/odrl-vocab 4.5.19)

        - This is of class “odrl:leftOperand”
        - The value is defined directly “odrl:leftOperand”: *IRI*
    - There may be none, one or many “odrl:refinement” (of type “odrl:constraint” - see Constraint below)
    - Purpose descriptions and refinements (if any) are presented as Constraints. This is best illustrated in the JSON-LD example given below, where the Purpose ‘block’ can be seen as part of the set of constraints (where Purpose refinements are presented these are tied with the Purpose using “odrl:and” (https://www.w3.org/TR/odrl-vocab 3.17.3)
- Target
    - This is used to identify a specific data asset.
    - See “odrl:target” and “@type” : “odrl:AssetCollection” .
    - The value is defined directly or in element “odrl:source” : *IRI* (if refinements defined for the target – see below).
    - Each rule allows the user to add none, one or many “odrl:refinement” (of type “odrl:constraint” - see Constraint below) (to the fixed target) that apply to that rule.
    - The odrl:leftoperand “QUERY” may be used to specify a subset of policy Target.
- Constraints (see ref 2.5)
    - Defines constraints applied to the Rule as a whole.
    - See “odrl:constraint”
        - This has:
            - “odrl:leftOperand”: *IRI*
            - “odrl:operator”: *IRI*
            - “odrl: rightOperand”: *IRI*.
    - There may be none, one or many “odrl:constraint” .

2.4 A Simple Example Policy Represented in JSON-LD

```
 "@context": {
     "dash": "http://datashapes.org/dash#",
     "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
     "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
     "schema": "http://schema.org/",
     "sh": "http://www.w3.org/ns/shacl#",
     "xsd": "http://www.w3.org/2001/XMLSchema#",
     "ex": "http://example.ex/",
     "odrl": "http://www.w3.org/ns/odrl/2/"
   },
"@id": "https://eu-upcast.github.io/shapes/mint#2policy17446325067569091",
"@type": "odrl:Policy",
“odrl:permission”:
 [
   {
     “odrl:action”
     {
        “rdf:value” :”https://w3id.org/dpv/owl#Analyse",
        “odrl:refinement”:
        [
          {
             “odrl:leftOperand”: “https://eu-upcast.github.io/vocabulary/index-en.html#hasStartTime” ,
             “odrl:operator”: “http://www.w3.org/ns/odrl/2/eq”,
             “odrl:rightOperand”: “Today”
          },
          {
             “odrl:leftOperand”: “https://eu-upcast.github.io/vocabulary/index-en.html#hasFinishTime”,
             “odrl:operator:http”: “//www.w3.org/ns/odrl/2/eq”,
             “odrl:rightOperand”: “Tomorrow”
          }
        ]
     }
     “odrl:assignee”:
     {
        “@type” : “odrl:PartyCollection”,
        “odrl:source” : “https://w3id.org/dpv/owl#Student”,
        odrl:refinement:
        [
          {
             “odrl:leftOperand”: “https://w3id.org/dpv/owl#isRepresentativeFor”,
             “odrl:operator”:http: “//www.w3.org/ns/odrl/2/eq”,
             “odrl:rightOperand”: “University of Harvard”
          }
        ]
     }
     “odrl:target”;
     {
        “@type” : “odrl:AssetCollection”,
        “odrl:source” : “http://example.org/datasets/covid19Stats”,
        “odrl:refinement”:
        [
          {
             “odrl:leftOperand”: “QUERY”,
             “odrl:operator”: “http://www.w3.org/ns/odrl/2/eq”,
             “odrl:rightOperand”: “SELECT C1, C2 FROM COVID_DATA”
          }
        ]
     }
```

```
      “odrl:constraint”
      [
         “odrl:and”:
         [
           {
              “odrl:leftOperand”: “odrl:purpose”,
              “odrl:operator”:http: “//www.w3.org/ns/odrl/2/eq”,
              “odrl:rightOperand”: “https://w3id.org/dpv/owl#AcademicResearch”
           },
           {
              “odrl:leftOperand”: “https://eu-upcast.github.io/vocabulary/index-en.html#hasApprovalFrom”,
              “odrl:operator”:http: “//www.w3.org/ns/odrl/2/eq”,
              “odrl:rightOperand”: “The Government”
           }
         ],
         {
            “odrl:leftOperand”: “http://www.w3.org/ns/odrl/2/spatial”,
            “odrl:operator”:http: “//www.w3.org/ns/odrl/2/eq”,
            “odrl:rightOperand”: “USA”
         }
      ]
    }
  ]

  “odrl:prohibition”:
  [
    {
       “odrl:action”: “http://www.w3.org/ns/odrl/2/transfer”,
       “odrl:assignee”: “https://w3id.org/dpv/owl#Entity”,
       “odrl:target”: “http://example.org/datasets/covid19Stats”
       “odrl:constraint”:
       [
         {
            “odrl:leftOperand”: “odrl:purpose”,
            “odrl:operator”: “http://www.w3.org/ns/odrl/2/eq”,
            “odrl:rightOperand”: “https://w3id.org/dpv/owl#AcademicResearch”
         }
       ]
    }
  ]


}
```

This policy is presented in a tabular form as follows

Policy Description		
Target	"http://example.org/datasets/covid19Stats"	
Rule	Permission	
	Action - Analyse	
	Refinement	**Left Operand** - hasStartTime
		Operator – Eq
		Right Operand – "Today"
	Refinement	**Left Operand** - hasFinishTime
		Operator – Eq
		Right Operand – "Tomorrow"
	Actor - Student	
	Refinement	**Left Operand** - isRepresentativeFor
		Operator – Eq
		Right Operand – "University of Harvard"
	Purpose – Academic Research	
	Refinement	**Left Operand** - hasApprovalFrom
		Operator – Eq
		Right Operand – "The Government"
	Target Refinements	
	Refinement	**Left Operand** - QUERY
		Operator – Eq
		Right Operand – "SELECT C1, C2 FROM COVID_DATA"
	Constraints	
	Constraint	**Left Operand** – Spatial (area)
		Operator – Eq
		Right Operand – "USA"
Rule	Prohibition	
	Action - Transfer	
	Actor - Entity	
	Purpose – Academic Research	
	Target Refinements	
	Constraints	