Appendix

A Ontology semantics in first-order logic

The main concepts, their relationships and the associated constraints are introduced hereafter, for all five ontology categories, using first-order logic. For the sake of brevity, we omit the implementation of constraints in SHACL [2].

A.1 SAO semantics

We explain the relations of SAO concepts by referring to a formula in Table 20 using its equation number.

$\forall x.(IdentifiedConcept(x)$	$\Longrightarrow$	$owl:Thing(x))$	(1)
$\forall x.(SpecifiedConcept(x)$	$\Longrightarrow$	$(IdentifiedConcept(x) \wedge$ $\exists y.(Requirement(y) \wedge isSpecifiedBy(x,y)) \wedge$ $(System(x) \vee Function(x) \vee Interface(x))))$	(2)
$\forall x.(System(x)$	$\Longrightarrow$	$(SpecifiedConcept(x) \wedge$ $(\exists y.(Function(y) \wedge performs(x,y)) \vee$ $\exists y.(Interface(y) \wedge presents(x,y)) \vee$ $\exists y.(System(y) \wedge contains(x,y) \wedge (x \neq y)) \vee$ $\exists y.(State(y) \wedge hasState(x,y)))))$	(3)
$\forall x.(Function(x)$	$\Longrightarrow$	$(SpecifiedConcept(x) \wedge$ $(\exists y.(Flow(y) \wedge (ingests(x,y) \vee emits(x,y))) \vee$ $\exists y.(Item(y) \wedge (sends(x,y) \vee receives(x,y))) \vee$ $\exists y.(Item(y) \wedge sets(x,y)) \vee$ $(\exists y.(StateValue(y) \wedge setsTo(x,y) \wedge$ $\exists z.(State(z) \wedge sets(x,z)))) \vee$ $\exists y.(Function(y) \wedge invokes(x,y) \wedge (x \neq y)) \vee$ $\exists y.(StateValue(y) \wedge isAvailableIn(x,y)))))$	(4)
$\forall x.(Interface(x)$	$\Longrightarrow$	$(SpecifiedConcept(x) \wedge$ $\exists y.(TraversingConcept(y) \wedge transfers(x,y))))$	(5)
$\forall x.(TraversingConcept(x)$	$\Longrightarrow$	$(IdentifiedConcept(x) \wedge$ $\exists y.(Connection(y) \wedge traverses(x,y)) \wedge$ $(Flow(x) \vee Item(x))))$	(6)
$\forall x.(Item(x)$	$\Longrightarrow$	$(TraversingConcept(x) \wedge \neg Flow(x) \wedge$ $\exists y.(Item(y) \wedge isDecomposedTo(x,y) \wedge (x \neq y))))$	
$\forall x.(Flow(x)$	$\Longrightarrow$	$TraversingConcept(x) \wedge \neg Item(x))$	(8)
$\forall x.(Connection(x)$	$\Longrightarrow$	$(IdentifiedConcept(x) \wedge$ $\exists y_1.(Interface(y_1) \wedge \exists y_2.(Interface(y_2) \wedge$ $joins(x,y_1) \wedge joins(x,y_2) \wedge (y_1 \neq y_2)))))$	(9)
$\forall x.(StateConcept(x)$	$\Longrightarrow$	$(IdentifiedConcept(x) \wedge$ $(State(x) \vee StateValue(x) \vee StateSet(x)) \wedge$ $(\exists y.((Item(y) \vee (State(y)) \wedge$ $takesValuesFrom(y,x) \wedge StateSet(x)) \vee$ $\exists y.(System(y) \wedge hasState(y,x) \wedge State(x)) \vee$ $\exists y.(State(y) \wedge hasSubState(x,y) \wedge State(x) \wedge (x \neq y)) \vee$ $\exists y.(StateValue(y) \wedge belongsTo(y,x) \wedge StateSet(x)))))$	(10)

Table 20: SAO relations in first-order logic

An `IdentifiedConcept` subsumes any other class (1) and allows to uniquely identify high-level concepts. Those concepts that should be possible to specify in a requirement (`System`, `Function`, `Interface`) are subclasses of the `SpecifiedConcept` class (Figure 3). This is defined as an `IdentifiedConcept`, for which there can be a `Requirement` (class of RDO) referring to its behaviour (2).

A `System` denotes a group of interacting or interrelated entities that form a unified whole. It is associated to a `Function` (`performs` property) or to an `Interface` (`presents` property), or may `contain` several `System`s or can have a `State` (3). An operation or activity by some `System` is denoted by a `Function`. Therefore, any `Function` represents some intended system behaviour (e.g. acquire telemetry packet) with the following possibilities (4): (i) `emit` or `ingest` `Flow`s representing non-discrete exchanges of matter or energy (e.g. an event), (ii) `send` or `receive` `Item`s representing discrete exchanges of matter or information, (iii) `set` a `State`'s or an `Item`'s `StateValue`. A `Function` may `invoke` several `Function`s and can be made available in an identified `StateValue`.

An `Interface` denotes a set of mechanical, electrical, signal, or other contact points referring to some aspect of a `System`'s `Connection` or interaction with another `System`. It is the means to transfer a `TraversingConcept` (5), which may be either a `Flow` or an `Item` (6 and 8). An `Item` can be decomposed into other `Item`s (7). When two (or more) `Interface`s mate with each other, this denotes the existence of a `Connection` that `joins` them (9).

Finally, a `StateConcept` denotes the condition in which a `System` or an `Item` is at a specific point in time. `State` represents a `System`'s state and can have sub-states, whereas it takes possible values from a `StateSet` (as is also the case with `Item`). A `StateSet` represents a collection of possible states, i.e. any `StateValue` `belongsTo` a `StateSet` (10).

A.2 RBO semantics

$\forall x.(Requirement(x)$	$\Longrightarrow$	$(owl:Thing(x) \wedge$ $\exists y.(ReqDescriptor(y) \wedge hasClause(x,y) \wedge hasMain(x,y))))$	(11)
$\forall x.(Prefix(x)$	$\Longrightarrow$	$(ReqDescriptor(x) \wedge$ $\exists y,z.(Requirement(y) \wedge Placeholder(z) \wedge$ $isRelatedToPlaceholder(x,z) \wedge hasPrefix(y,x))))$	(12)
$\forall x.(Main(x)$	$\Longrightarrow$	$(ReqDescriptor(x) \wedge$ $\exists y,z.(Requirement(y) \wedge Placeholder(z) \wedge$ $isRelatedToPlaceholder(x,z) \wedge hasMain(y,x))))$	(13)
$\forall x.(Suffix(x)$	$\Longrightarrow$	$(ReqDescriptor(x) \wedge$ $\exists y,z.(Requirement(y) \wedge Placeholder(z) \wedge$ $isRelatedToPlaceholder(x,z) \wedge hasSuffix(y,x))))$	(14)

Table 21: RBO relations in first-order logic

A.3 RDO semantics

We explain the relations of RDO concepts by referring to a formula in Table 22 using its equation number.

$\forall x.(Requirement(x)$	$\Longrightarrow$	$(owl:Thing(x) \wedge$ $\exists y.(SpecifiedConcept(y) \wedge isSpecifiedBy(y,x))))$	(15)
$\forall x,y,z.(IdentifiedConcept(x) \wedge$ $Requirement(y) \wedge ReqDescriptor(z)) \wedge$ $hasMain(y,z) \wedge isRelatedToPlaceholder(z,x))$	$\Longrightarrow$	$isSpecifiedBy(x,y)))$	(16)
$\forall x.(FunctionalRequirement(x)$	$\Longrightarrow$	$(Requirement(x) \wedge$ $\exists y.(Function(y) \wedge isSpecifiedBy(y,x))))$	(17)
$\forall x.(InterfaceRequirement(x)$	$\Longrightarrow$	$(Requirement(x) \wedge$ $\exists y.(Interface(y) \wedge isSpecifiedBy(y,x))))$	(18)
$\forall x.(SystemRequirement(x)$	$\Longrightarrow$	$(Requirement(x) \wedge$ $\exists y.(System(y) \wedge isSpecifiedBy(y,x))))$	(19)

Table 22: RDO relations in first-order logic

A `Requirement` specifies an assertion about a `SpecifiedConcept` (i.e. `System`, `Function`, or `Interface`) that must hold true for any acceptable realization of that concept (15); `isSpecifiedBy` denotes an inference relationship between a `Requirement` and an `IdentifiedConcept` that comes from boilerplate definitions, such that the `IdentifiedConcept` is related to a placeholder of a `Main` clause boilerplate (16).

A `Requirement` that specifies a `Function` is a `FunctionalRequirement` (17). When specifying an `Interface`, it is an `InterfaceRequirement` (18) and when specifying a `System`, it is a `SystemRequirement` (19).

B Experiment specification

Orbit control system consists of the following:

- Sensor Processor system:
  - Measures the angular rate based on the four axes raw data coming from the Gyroscope sensor at a frequency of 16Hz.
  - Measures the Sun vector based on the raw data coming from the 2 Fine Sun Sensors at a frequency of 8Hz.
- Datafusion system:
  - It calculates the average rates based on the two latest 16 Hz angular rate measurements from the gyroscope sensor processing
  - It calculates the estimated quaternion of attitude, estimated rate and estimated direction of the Sun based on the average rates and Sun vector measurements at a frequency of 8Hz.
- Error Generator system: Error generation computes the Attitude and Rate Error between a reference quaternion of attitude, reference rate and the estimated quaternion of attitude and estimated rate.
- Guidance system: Guidance system estimates the reference quaternion of attitude and reference rate based on the estimated quaternion of attitude,

estimated rate and estimated direction of Sun. Guidance system also provides reference values to the Error Generator system.

- Commanding system:
  - If orbit control system is in ThrustingMode, commanding system converts a Control Torque Demand received from the Controller into individual thruster actuator commands.
  - If orbit control system is in ReactionWheelMode, commanding system converts a Control Torque Demand received from the Controller into individual reaction wheel actuator commands.
- Controller system: Controller system computes a Control Torque Demand based on the Attitude and Rate Error.

Orbit control system uses MIL-STD-1553 bus protocol for system communication.

C Semantic analyses

C.1 SHACL rules to detect incompleteness

Evaluation of internal completeness is based on identifying all requirements with at least one placeholder still being a class instead of having a concrete value (instance). External incompleteness is detected by finding all requirements with a placeholder value, whose direct or transitive `SAO:providesTo` and `SAO dependsOn` relations are missing. The former is supposed to encode the asymmetric producer/consumer relationships in SAO, while the latter express all direct relationships of SAO (e.g. `Function` should `dependsOn System`, when there is a `perform` relationship between them).

SHACL rule 1 was added to class `Requirement` to represent all requirements with at least one OWL object property whose range is a class (abstract value) instead of a class instance (concrete value). This rule classifies an instance of `Requirement` to class `IncompleteRequirement`, if the requirement is related to a placeholder that is not concrete, namely, is not instantiated, and therefore is a `OWL:Class`. Note that logic representation lacks some of the SHACL rule semantics, such as the transitive closure of `isRelatedToPlaceholder` property and deduction of multiple inferences.

SHACL rules 2 - 4 were also added to class `Requirement` to represent all external incomplete requirements, and classify an instance of `Requirement` to class `IncompleteRequirement`, if there are missing concepts within a requirements set, taking into account the explicit knowledge defined in the DSO ontologies. This explicit knowledge is described as SAO relations between concepts. Therefore, in order to detect external incompleteness, we have grouped all direct or intermediate relations to the following three detectable cases:

1. A requirement (i.e. $this) is related to a placeholder which `dependsOn` another concept (i.e. ?val), which is missing from any other requirement in the requirements set (Rule 2). Relation `dependsOn` expresses all directed graphs in SAO, such that in an ordered pair of vertices (x, y), y `dependsOn` x.

2. A requirement (i.e. $this) is related to a placeholder which another concept `providesTo`. The latter is missing from any other requirement in the requirements set (Rule 3). Relation `providesTo` expresses the asymmetric producer/consumer relationships in SAO, for example in the case where a `SpecifiedConcept providesTo` another `SpecifiedConcept` a `TraversingConcept` to consume, through a connection that joins two systems together.

3. A requirement (i.e. $this) is related to a `StateValue` which `belongsTo` a `StateSet` which has additional states, which are missing from any other requirement in the requirements set(Rule 4). Relation `belongsTo` expresses the SAO relationship of a `StateConcept`, which takes value from a `StateSet`.

SPARQL query of SHACL rule

```
CONSTRUCT {
?this a ReqDo:IncompleteRequirement .
?this RBO:isMissing ?e .
}
WHERE {
?this RDO:hasClause ?x .
?x (RBO:isRelatedToPlaceholder)+ ?e .
?e a owl:Class .
}
```

Rule 1: Internal incompleteness requirement definition

C.2 SHACL rules to detect inconsistencies

This inference of conflicts between specified requirements takes place through grouping all relations to the following two types of relations:

1. Whenever two values of boilerplate placeholders are semantically equivalent (same OWL properties), they are characterized by the `isSimilarTo` relation, which is symmetric and transitive.

2. Two values of boilerplate placeholders contradict with each other, if they differ in at least one OWL property. Then, they are characterized by the `contradictsWith` relationship, which is symmetric.

The SHACL rules that implement these analyses are based on the knowledge coming from the SAO relations. SHACL rules 5 - 7 were added to class `Requirement` to represent all inconsistent requirements, and classify an instance of `Requirement` to class `InconsistentRequirement`, if in terms of used boilerplates, they are assigned to the same placeholders, for the same subject, different concept instances or use contradicting words or different quantifiers (e.g. numbers or number units):

SPARQL query of SHACL rule

```
CONSTRUCT {
$this a RDO:IncompleteRequirement .
$this RDO:isMissingProducer ?val .
}
WHERE {
{$this ?p1 ?x . ?p1 rdfs:subPropertyOf* RDO:hasClause}
{?x ?p2 ?e . ?p2 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder}
{?x RBO:isRelatedToAction ?action .}
?subject RDO:isSubjectOfRequirement $this .
FILTER (?e != ?subject && ?e != ?action)
?e rdf:type/rdfs:subClassOf* SAO:TraversingConcept .
?e SAO:dependsOn+ ?val .
FILTER NOT EXISTS {
{?val rdf:type/rdfs:subClassOf* EAO:Environment .}
UNION
{
?env SAO:performs ?val .
?env rdf:type/rdfs:subClassOf* EAO:Environment .
}
UNION
{
?env SAO:performs ?env_function .
?env rdf:type/rdfs:subClassOf* EAO:Environment .
?env_function ?pl ?val .
FILTER (STRBEFORE(str(?pl), "#") = "http://delab.csd.auth.gr/ontologies/2018/SAO") .
}
}
FILTER (?val != ?subject)
FILTER NOT EXISTS {
{?req  ?p3 ?y . ?p3 rdfs:subPropertyOf* RDO:hasClause}
{?y ?p4 ?e1 . ?p4 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder}
FILTER ($this != ?req) .
?e1 SAO:dependsOn+ ?val .
}
FILTER EXISTS {
BIND (CONCAT(CONCAT(strafter(str(?action),"#"), "_"), strafter(str(?e),"#")) AS ?functionName) .
BIND (URI(CONCAT ("http://delab.csd.auth.gr/ontologies/2018/RDO-instances#", ?functionName))
AS ?function) .
FILTER (?val != ?function) .
}
}
```

Rule 2: External incompleteness due to missing concept

1. SHACL rule 5 returns pairs of requirements that possibly contradict with each other, because the first one (i.e. `Requirement` $this) defines an action (i.e. `Function` ?a1) for a subject (i.e. `System` ?s), whereas the second `Requirement` ?r2 through `hasClause` ?c2 is related to a TraversingConcept (i.e. `Flow` or `Item` ?t) of the same subject ?s, but for a contradicting action (i.e. `Function` ?a2).

2. SHACL rule 6 returns pairs of requirements that possibly contradict with each other, because the first one (i.e. `Requirement` $this) defines a placeholder ?inconsistent_placeholder1 for a subject (i.e. `System` ?s), whereas the second `Requirement` ?r2 uses a contradicting placeholder (i.e. ?inconsistent_placeholder2) instance for the same subject ?s.

3. SHACL rule 7 returns pairs of requirements that possibly contradict with each other, because the first one (i.e. `Requirement` $this) defines a quantifier ?q1 for a subject (i.e. `System` ?s), whereas the second `Requirement` ?r2 uses a contradicting quantifier (i.e. ?q2) for the same subject ?s.

SPARQL query of SHACL rule

```
CONSTRUCT {
$this a RDO:IncompleteRequirement .
$this  RDO:isMissingConcept ?val .
}
WHERE {
{$this ?p1 ?x . ?p1 rdfs:subPropertyOf* RDO:hasClause}
{?x ?p2 ?e . ?p2 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder}
?val SAO:providesTo ?e .
FILTER NOT EXISTS {
{?val rdf:type/rdfs:subClassOf* EAO:Environment .}
UNION
{
?env SAO:performs ?val .
?env rdf:type/rdfs:subClassOf* EAO:Environment .
}
UNION
{
?env SAO:performs ?env_function .
?env rdf:type/rdfs:subClassOf* EAO:Environment .
?env_function ?pl ?val .
FILTER (STRBEFORE(str(?pl), "#") = "http://delab.csd.auth.gr/ontologies/2018/SAO") .
}
}
FILTER NOT EXISTS {
{?req  ?p3 ?y . ?p3 rdfs:subPropertyOf* RDO:hasClause}
{?y ?p4 ?val . ?p4 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder}
FILTER ($this != ?req) .
} .
FILTER (?val != ?e) .
}
```

Rule 3: External incompleteness due to missing producer

SPARQL query of SHACL rule

```
CONSTRUCT {
$this a RDO:IncompleteRequirement .
$this  RDO:isMissingState ?val .
}
WHERE {
{$this ?p1 ?x . ?p1 rdfs:subPropertyOf* RDO:hasClause}
{?x ?p2 ?e . ?p2 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder}
?e SAO:belongsTo ?s .
?val SAO:belongsTo ?s .
FILTER NOT EXISTS {
{$req ?p3 ?y . ?p3 rdfs:subPropertyOf* RDO:hasClause}
{?y ?p4 ?val . ?p4 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder}
FILTER ($this != ?req) .
} .
FILTER (?val != ?e) .
}
```

Rule 4: External incompleteness due to missing state

C.3 SHACL rules to detect ambiguity

SHACL rule 8 was added to class `Requirement` to represent all ambiguous requirements, and classify an instance of `Requirement` to class `AmbiguousRequirement`,

if `Requirement` $this is related to a placeholder ?p1, where other instances (i.e. ?p2) of more concrete subclasses of placeholder ?p1 exist.

C.4 SHACL rules to detect noise

SHACL rule 9 was added to class `Requirement` to represent all noisy requirements, and classify an instance of `Requirement` to class `NoisyRequirement`, if `Requirement` $this has in the same clause (i.e. ?boilerplate) a placeholder (i.e. ?placeholder), which is not an instance of a domain ontology class, but instance of class `RBO:Entity`, `SAO:IdentifiedConcept`, `RBO:ComplexEntity` or `RBO:SimpleEntity`.

C.5 SHACL rules to detect opacity

To detect opacity we focus on tracing the `dependsOn` relations (as opposed to the `subClassOf` relations that are taced for detecting noise). Specifically, this analysis works by identidying those requirements having a boilerplate placeholder value without existing `dependsOn` relationship with the requirement's subject.

SHACL rule 10 was added to class `Requirement` to represent all opaque requirements, and classify an instance of `Requirement` to class `OpaqueRequirement`, if `Requirement` $this is related to a placeholder (i.e. ?placeholder), which is not relevant (i.e. `dependOn`) on the requirement's subject ?s.

C.6 SHACL rules to detect redundancy

SHACL rules 11 and 12 were added to class `Requirement` to represent all redundant requirements, and classify an instance of `Requirement` to class `RedundantRequirement`, if a pair of requirements have in their boilerplate placeholders for the same subject similar concept instances or same quantifiers:

1. SHACL rule 11 returns pairs of requirements that are possibly redundant, because the first one (i.e. `Requirement` $this) for subject ?s, has a non-quantifier placeholder (i.e. ?redundant_placeholder1) in its clause, and exists a second `Requirement` ?r2 that for the same subject ?s, also has a non-quantifier placeholder (i.e. ?redundant_placeholder2) which is equal or `similarTo` placeholder ?redundant_placeholder1.

2. SHACL rule 12 returns pairs of requirements that are possibly redundant, because the first one (i.e. `Requirement` $this) for subject ?s, has a quantifier ?q1 in its clause, while a second `Requirement` ?r2, for the same subject ?s, has a quantifier placeholder ?q2 which is equal or `similarTo` placeholder ?q1.

SPARQL query of SHACL rule

```
CONSTRUCT {
$this a RDO:InconsistentRequirement .
$this RDO:isInconsistentTo ?r2 .
}
WHERE {
$this ?p1 ?c1 .
?p1 rdfs:subPropertyOf* RDO:hasClause .
{
{
?c1 ?pr1 ?s .
FILTER (?pr1 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
OPTIONAL {?c1 RBO:isRelatedToQuantifier ?q1}
}
UNION {
?c1 ?pl1 ?placeholder1 .
?pl1 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder .
?placeholder1 ?pr1 ?s .
FILTER (?pr1 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
OPTIONAL {?placeholder1 RBO:isRelatedToQuantifier ?q1}
}
}
?c1 RBO:isRelatedToAction ?a1 .
?c1 RBO:isRelatedToTraversingConcept ?t .
?r2 a RDO:Requirement .
FILTER ($this != ?r2)
?r2 ?p2 ?c2 .
?p2 rdfs:subPropertyOf* RDO:hasClause .
{
{
?c2 ?pr2 ?s .
FILTER (?pr2 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
OPTIONAL{?c2 RBO:isRelatedToQuantifier ?q2}
}
UNION {
?c2 ?pl2 ?placeholder2 .
?pl2 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder .
?placeholder2 ?pr2 ?s .
FILTER (?pr2 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
OPTIONAL{?placeholder2 RBO:isRelatedToQuantifier ?q2 }
}
}
?c2 RBO:isRelatedToAction ?a2 .
?c2 RBO:isRelatedToTraversingConcept ?t .
?a1 RBO:contradictsWith ?a2 .
OPTIONAL{
FILTER (
(EXISTS{?q2 (RBO:isSimilarTo|^RBO:isSimilarTo) ?q1 .})
||
(?q2=?q1)
)
}
}
```

Rule 5: Inconsistency due to contradicting actions

SPARQL query of SHACL rule

```
CONSTRUCT {
$this a RDO:InconsistentRequirement .
$this RDO:isInconsistentTo ?r2 .
}
WHERE {
$this ?p1 ?c1 .
?p1 rdfs:subPropertyOf* RDO:hasClause .
{
{
?c1 ?pr1 ?s .
FILTER (?pr1 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
?c1 RBO:isRelatedToQuantifier ?q1 .
}
UNION {
?c1 ?pl1 ?placeholder1 .
?pl1 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder .
?placeholder1 ?pr1 ?s .
FILTER (?pr1 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
?placeholder1 RBO:isRelatedToQuantifier ?q1 .
}
}
?c1 RBO:isRelatedToAction ?a1 .
?c1 RBO:isRelatedToTraversingConcept ?t .
?c1 ?pr5 ?inconsistent_placeholder1 .
?pr5 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder .
FILTER ((?pr5 != RBO:isRelatedToTraversingConcept) && (?pr5 != ?pr1) && (?pr5 != RBO:isRelatedToAction))
?r2 a RDO:Requirement .
FILTER ($this != ?r2)
?r2 ?p2 ?c2 .
?p2 rdfs:subPropertyOf* RDO:hasClause .
{
{
?c2 ?pr2 ?s .
FILTER (?pr2 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
?c2 RBO:isRelatedToQuantifier ?q2 .
}
UNION {
?c2 ?pl2 ?placeholder2 .
?pl2 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder .
?placeholder2 ?pr2 ?s .
FILTER (?pr2 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
?placeholder2 RBO:isRelatedToQuantifier ?q2 .
}
}
?c2 RBO:isRelatedToAction ?a2 .
?c2 RBO:isRelatedToTraversingConcept ?t .
?c2 ?pr5 ?inconsistent_placeholder2 .
FILTER ((?pr5 != RBO:isRelatedToTraversingConcept) && (?pr5 != ?pr2))
FILTER (
(EXISTS{?a1 (RBO:isSimilarTo|^RBO:isSimilarTo) ?a2 .})
||
(?a1=?a2)
)
FILTER (
(EXISTS{?q1 (RBO:isSimilarTo|^RBO:isSimilarTo) ?q2 .})
||
(?q1=?q2)
)
FILTER (
(EXISTS{?inconsistent_placeholder2 RBO:contradictsWith ?inconsistent_placeholder1 .})
||
(?inconsistent_placeholder2!=?inconsistent_placeholder1)
)
}
```

Rule 6: Inconsistency due to contradicting placeholders

SPARQL query of SHACL rule

```
CONSTRUCT {
?this a RDO:InconsistentRequirement .
$this RDO:isInconsistentTo ?r2 .
}
WHERE {
$this ?p1 ?c1 .
?p1 rdfs:subPropertyOf* RDO:hasClause .
{
{
?c1 ?pr1 ?s  .
FILTER (?pr1 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
?c1 RBO:isRelatedToQuantifier ?q1 .
}
UNION {
?c1 ?pl1 ?placeholder1 .
?pl1 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder .
?placeholder1 ?pr1 ?s  .
FILTER (?pr1 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
?placeholder1 RBO:isRelatedToQuantifier ?q1 .
}
}
?c1 RBO:isRelatedToAction ?a1 .
?c1 RBO:isRelatedToTraversingConcept ?t .
?r2 a RDO:Requirement .
FILTER ($this != ?r2)
?r2 ?p2 ?c2 .
?p2 rdfs:subPropertyOf* RDO:hasClause .
{
{
?c2 ?pr2 ?s  .
FILTER (?pr2 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
?c2 RBO:isRelatedToQuantifier ?q2 .
}
UNION {
?c2 ?pl2 ?placeholder2 .
?pl2 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder .
?placeholder2 ?pr2 ?s  .
FILTER (?pr2 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
?placeholder2 RBO:isRelatedToQuantifier ?q2 .
}
}
?c2 RBO:isRelatedToAction ?a2 .
?c2 RBO:isRelatedToTraversingConcept ?t .
FILTER (
(EXISTS{?a1 (RBO:isSimilarTo|^RBO:isSimilarTo) ?a2 .})
||
(?a1=?a2)
)
?q2 RBO:contradictsWith ?q1 .
}
```

Rule 7: Inconsistency due to contradicting quantifier

SPARQL query of SHACL rule

```
CONSTRUCT {
$this a RDO:AmbiguousRequirement .
$this RDO:hasAmbiguousConcept ?c1 .
$this RDO:hasAmbiguousConcept2 ?c2 .
}
WHERE {
{$this ?p1 ?x . ?p1 rdfs:subPropertyOf* RDO:hasClause}
{?x ?p2 ?e1 . ?p2 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder}
?e1 rdf:type ?c1 .
?c1 rdfs:subClassOf* SAO:IdentifiedConcept .
?c2 rdfs:subClassOf ?c1 .
FILTER (?c2 != rdfs:Class) .
?e2 rdf:type ?c2 .
FILTER (?e2 != ?e1) .
}
```

Rule 8: Ambiguous requirement

SPARQL query of SHACL rule

```
CONSTRUCT {
$this a RDO:NoisyRequirement .
$this RDO:hasNoisyConcept ?placeholder .
}
WHERE {
$this rdf:type RDO:Requirement .
$this (RDO:hasMain|RDO:hasPrefix|RDO:hasSuffix) ?boilerplate .
FILTER EXISTS {
{
?boilerplate ?p1 ?placeholder .
?p1 rdfs:subPropertyOf RBO:isRelatedToPlaceholder .
}
UNION
{
?boilerplate ?p1 ?v1 .
?p1 rdfs:subPropertyOf RBO:isRelatedToPlaceholder .
?v1 ?p2 ?placeholder .
?p2 rdfs:subPropertyOf RBO:isRelatedToPlaceholder .
}
UNION
{
?boilerplate ?p1 ?v1.
?p1 rdfs:subPropertyOf RBO:isRelatedToPlaceholder .
?v1 ?p2 ?v2 .
?p2 rdfs:subPropertyOf RBO:isRelatedToPlaceholder .
?v2 ?p3 ?placeholder .
?p3 rdfs:subPropertyOf RBO:isRelatedToPlaceholder .
}
}
?placeholder rdf:type/rdfs:subClassOf* RBO:Entity .
FILTER EXISTS {
{?placeholder rdf:type/rdfs:subClassOf SAO:IdentifiedConcept .}
UNION
{?placeholder rdf:type/rdfs:subClassOf RBO:ComplexEntity .}
UNION
{?placeholder rdf:type/rdfs:subClassOf RBO:SimpleEntity .}
}
}
```

Rule 9: Noisy requirement

SPARQL query of SHACL rule

```
CONSTRUCT {
$this a RDO:OpaqueRequirement .
$this RDO:hasOpaqueConcept ?placeholder .
}
WHERE {
$this rdf:type RDO:Requirement .
$this (RDO:hasMain|RDO:hasPrefix|RDO:hasSuffix) ?boilerplate .
OPTIONAL{
{
?boilerplate ?p1 ?placeholder .
?p1 rdfs:subPropertyOf RBO:isRelatedToPlaceholder .
}
UNION
{
?boilerplate ?p1 ?v1 .
?p1 rdfs:subPropertyOf RBO:isRelatedToPlaceholder .
?v1 ?p2 ?placeholder .
?p2 rdfs:subPropertyOf RBO:isRelatedToPlaceholder .
}
UNION
{
?boilerplate ?p1 ?v1.
?p1 rdfs:subPropertyOf RBO:isRelatedToPlaceholder .
?v1 ?p2 ?v2 .
?p2 rdfs:subPropertyOf RBO:isRelatedToPlaceholder .
?v2 ?p3 ?placeholder .
?p3 rdfs:subPropertyOf RBO:isRelatedToPlaceholder .
}
}
?placeholder rdf:type/rdfs:subClassOf* RBO:Entity .
?s RDO:isSubjectOfRequirement $this .
FILTER (?placeholder != ?s)
FILTER NOT EXISTS {
?placeholder SAO:dependsOn+ ?s .
}
}
```

Rule 10: Opaque requirement

SPARQL query of SHACL rule

```
CONSTRUCT {
?this a RDO:RedundantRequirement .
}
WHERE {
$this ?p1 ?c1 .
?p1 rdfs:subPropertyOf* RDO:hasClause .
{
{
?c1 ?pr1 ?s  .
FILTER (?pr1 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
OPTIONAL{?c1 RBO:isRelatedToQuantifier ?q1 .}
}
UNION {
?c1 ?pl1 ?placeholder1 .
?pl1 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder .
?placeholder1 ?pr1 ?s  .
FILTER (?pr1 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
OPTIONAL{?placeholder1 RBO:isRelatedToQuantifier ?q1 .}
}
}
?c1 RBO:isRelatedToAction ?a1 .
?c1 RBO:isRelatedToTraversingConcept ?t .
?c1 ?pr5 ?redundant_placeholder1 .
?pr5 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder .
FILTER ((?pr5 != RBO:isRelatedToTraversingConcept) && (?pr5 != ?pr1) && (?pr5 != RBO:isRelatedToAction))
?r2 a RDO:Requirement .
FILTER ($this != ?r2)
?r2 ?p2 ?c2 .
?p2 rdfs:subPropertyOf* RDO:hasClause .
{
{
?c2 ?pr2 ?s  .
FILTER (?pr2 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
OPTIONAL{?c2 RBO:isRelatedToQuantifier ?q2 .}
}
UNION {
?c2 ?pl2 ?placeholder2 .
?pl2 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder .
?placeholder2 ?pr2 ?s  .
FILTER (?pr2 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
OPTIONAL{?placeholder2 RBO:isRelatedToQuantifier ?q2 .}
}
}
?c2 RBO:isRelatedToAction ?a2 .
?c2 RBO:isRelatedToTraversingConcept ?t .
?c2 ?pr5 ?redundant_placeholder2 .
FILTER ((?pr5 != RBO:isRelatedToTraversingConcept) && (?pr5 != ?pr2))
FILTER (
(EXISTS{?a1 (RBO:isSimilarTo|^RBO:isSimilarTo) ?a2 .})
||
(?a1=?a2)
)
FILTER (
(EXISTS{?q1 (RBO:isSimilarTo|^RBO:isSimilarTo) ?q2 .})
||
(?q1=?q2)
)
FILTER (
(EXISTS{?redundant_placeholder2 (RBO:isSimilarTo|^RBO:isSimilarTo) ?redundant_placeholder1 .})
||
(?redundant_placeholder2 =?redundant_placeholder1)
)
}
```

Rule 11: Redundancy due to similar placeholders

SPARQL query of SHACL rule

```
CONSTRUCT {
$this a RDO:RedundantRequirement .
}
WHERE {
$this ?p1 ?c1 .
?p1 rdfs:subPropertyOf* RDO:hasClause .
{
{
?c1 ?pr1 ?s  .
FILTER (?pr1 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
OPTIONAL{?c1 RBO:isRelatedToQuantifier ?q1 .}
}
UNION {
?c1 ?pl1 ?placeholder1 .
?pl1 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder .
?placeholder1 ?pr1 ?s  .
FILTER (?pr1 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
OPTIONAL{?placeholder1 RBO:isRelatedToQuantifier ?q1 .}
}
}
?c1 RBO:isRelatedToAction ?a1 .
?c1 RBO:isRelatedToTraversingConcept ?t .
?r2 a RDO:Requirement .
FILTER ($this != ?r2)
?r2 ?p2 ?c2 .
?p2 rdfs:subPropertyOf* RDO:hasClause .
{
{
?c2 ?pr2 ?s  .
FILTER (?pr2 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
OPTIONAL{?c2 RBO:isRelatedToQuantifier ?q2 .}
}
UNION {
?c2 ?pl2 ?placeholder2 .
?pl2 rdfs:subPropertyOf* RBO:isRelatedToPlaceholder .
?placeholder2 ?pr2 ?s  .
FILTER (?pr2 IN (RBO:isRelatedToSubject, RBO:isRelatedToStimulus))
OPTIONAL{?placeholder2 RBO:isRelatedToQuantifier ?q2 .}
}
}
?c2 RBO:isRelatedToAction ?a2 .
?c2 RBO:isRelatedToTraversingConcept ?t .
FILTER (
(EXISTS{?a1 (RBO:isSimilarTo|^RBO:isSimilarTo) ?a2 .})
||
(?a1=?a2)
)
FILTER (
(EXISTS{?q2 (RBO:isSimilarTo|^RBO:isSimilarTo) ?q1 .})
||
(?q2=?q1)
)
}
```

Rule 12: Redundancy due to same quantifier