# OceanLink Core Ontology Patterns

Draft Version

*Contributors:*
ADILA KRISNADHI — Wright State University
ROBERT ARKO — Columbia University
CYNTHIA CHANDLER — Woods Hole Oceanographic Institution
MICHELLE CHEATHAM — Wright State University
TIMOTHY FININ — University of Maryland, Baltimore County
PASCAL HITZLER — Wright State University
KRZYSZTOF JANOWICZ — University of California, Santa Barbara
THOMAS NAROCK — University of Maryland, Baltimore County
LISA RAYMOND — Woods Hole Oceanographic Institution
ADAM SHEPHERD — Woods Hole Oceanographic Institution
PETER WIEBE — Woods Hole Oceanographic Institution

*Document Date:* January 8, 2014

This work was supported by the National Science Foundation under award 1354778 *EAGER: Collaborative Research: EarthCube Building Blocks, Leveraging Semantics and Linked Data for Geoscience Data Sharing and Discovery*. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

# Contents

**Pascal's Notes (still work in progress):**

Changes resulting from discussions with Krzysztof:

- Section **??** added to provide alternative formalization.
- Alternative modeling of birth as event in Section **??**.
- Added information object.
- Added information object to most patterns.
- Added Trajectory pattern
- Remodeled cruise pattern using event and trajectory.
- remodeled vessel pattern using qudt as basis.
- several minor corrections.

Further comments from discussions between Pascal and Krzysztof:

- rethink all property names called hasXYZ
- lifting shortcuts to more generic level
- we should make pictures more consistent: avoid parallel arrows to the same range
- use local (scoped) domain and range restrictions in the axiomatizations
- hasDescription should be mapped to dc
- rethink colors etc. in pictures.

Several additional comments/changes resulting from discussions with Krzysztof

# 0 Overview

## 0.1 About This Document

This refers to Chapter 5, Section 5.3

This document collects all core (ontology) patterns that will serve two major roles. First, they will act as the basis of the integration layer to which various data repositories within the OceanLink project will be mapped. Second, they are used to guide the navigational and retrieval components of the integrated user interface. The content of this document is derived from the results of the project meeting in Baltimore on November 5-8, 2013 attended by all of the editors and content contributors, as well as subsequent online communications.

In general, each core pattern in this document is presented as a separate chapter. Each chapter is then organized according to the following sequence.

1. An informal description is given, possibly with some visual depiction of the pattern.
2. An axiomatization of the pattern is presented in the form of a set of axioms. Each axiom can either be written using Description Logic (DL) notation or Datalog notation. The axiomatization formalizes the semantic relationships that are implicit in the informal description.
3. A collection of *views* for the pattern is given, if any. Here, a view is simply a DL axiom or rule whose sole purpose is to ease the user task of expressing certain important queries. Note that although views are given as axioms, they do not impose constraints (i.e., ontological commitments) to the pattern. In a sense, a view here is simply a shortcut for queries and analogous to the similar notion from databases.
4. Miscellaneous remarks if necessary. It may also contain alternative proposals (we will settle on a single version before moving on in the project).

## 0.2 Namespace

We adopt the following convention regarding the namespace for all patterns.

1. The base namespace is given by the URI: `http://schema.ocean-data.org`

   If needed, the preferred prefix for this base namespace is `od`
2. Each pattern has its own URI under the base namespace. So, for a pattern `pattern-name`, the URI is `http://schema.ocean-data.org/pattern-name`

   This URI acts as the ontology IRI of the OWL file of the pattern. Versioning is accommodated through version IRI of the form `http://schema.ocean-data.org/pattern-name/version-number`.
3. The URI of a class name, property name, and individual name is given as a hashed URI of the following forms:

   - `http://schema.ocean-data.org/pattern-name#ClassName`
   - `http://schema.ocean-data.org/pattern-name#propertyName`
   - `http://schema.ocean-data.org/pattern-name#individual_name`

   where `pattern-name` is the pattern in which the (class/property/individual) name is **originally** defined. We do not put an explicit versioning information on the URI of the (class/property/individual) name above. This helps us to obtain more stable URIs. The above names are assumed to correspond to the latest/current version of the pattern. Further details on dealing with versioning will be made in the future.

4. It is perfectly possible to have the same name fragment defined in two different patterns. For instance, we have a property called startsAt occurring in `agent-role` (Section 2) and `personal-info-item` (Section 5) patterns. In this case, we consider startsAt in `agent-role` to be a different property from the one in `personal-info-item` because their URIs are actually different. This avoids any direct inter-relationship between them, unless we explicitly specify it through axiomatization (which, in this case, we don't). Note that if a name fragment occurs in a pattern because of inheritance/subclassing, then it will retain its original URI. For example, the startsAt property occurring in `person-name` (Section 6) and `person-affiliation` (Section 9) patterns is still represented using its original URI as defined in the `personal-info-item` pattern.

5. In pattern descriptions and axiomatizations, we are usually able to recognize which pattern that defines a class/property/individual name. Thus, to simplify the notation, we typically drop the URI parts of a class/property/individual name up to the hash character, i.e., only take the name fragment of the full URI. Font for the (class/property/individual) names is also not necessarily a typewriter font (can be Serif or other font families).

6. Some external namespaces are used for some of the class/property/individual names. In that case, we use a prefixed format for the URI for those names. In this document, the external namespaces (and its prefix) are as follows:

   - `http://www.w3.org/2000/01/rdf-schema#` with prefix **rdfs**:
   - `http://www.w3.org/2002/07/owl#` with prefix **owl**:
   - `http://www.w3.org/2006/time#` with prefix **time**:

Note that there exist some terms in our vocabulary that are similar to terms in existing (external) vocabulary. For example, our vocabulary introduces a class Person which is similar to the class foaf:Person (i.e., `http://xmlns.com/foaf/0.1/Person`). In such a situation, one can argue that it is better to reuse the term from an existing, external vocabulary directly. However, we think that this would imply a strong ontological commitment to that external vocabulary which is something that we do not necessarily want. By using our own namespace, we avoid this issue and only when necessary will we make a link/mapping/alignment to an existing vocabulary.

## 0.3 Notations

### 0.3.1 Graphical Notation

Each pattern is visualized as a graph structure as depicted in Fig. 0.1. The nodes of the graph are labeled either with a *class name* (orange- and blue-colored nodes), *individual name* (yellow-colored nodes), *RDF literal* (yellow-colored nodes) and *RDF datatype* (green-colored nodes). Individuals and literals (yellow-colored nodes) are defined in the current pattern (the pattern descibed by the graph), unless stated otherwise. Likewise, classes represented by orange-colored nodes are defined in the current pattern Meanwhile, classes represented by blue colored nodes are defined in some external patterns. We indicate the external pattern name by stating it explicitly or by writing the class name enclosed with angled brackets. The latter actually indicates that the class name also acts as the external pattern name that defines it. For example, in Fig. 0.1, ClassName2 also indicates the name of a pattern that defines it, which, by our convention, is class-name2.

The edges of the graph are labeled with *property names*. Properties defined in the current pattern are represented by standard black arrows. Properties defined in some external pattern are represented by dotted black arrows. We usually omit the information regarding the external pattern name from the edge representing such a property since this is always clear by inspecting the nodes connected by the edge. Finally, special properties such as `rdfs:subClassOf` or `rdf:type` are represented by dotted blue arrows.

The reader should note here that if a class is defined in an external pattern (i.e., blue colored nodes), then there are almost always additional details (other related names, axioms, etc.) which are not explicitly spelled out in the current pattern. Typically, we only care about the class name and other details are not important for the current pattern. However, there are occasions in which some of the external details are

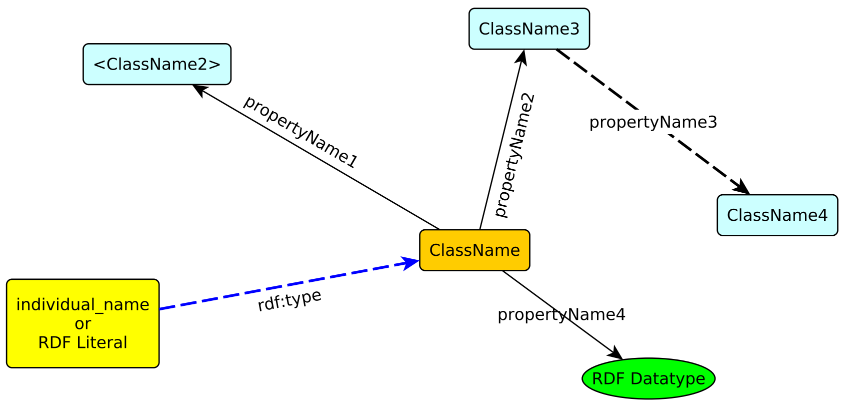

Figure 0.1: Graphical Notation for a pattern

important in the current pattern, e.g., if we want to assert some axiom involving both names defined in the current pattern and some class names in the inner part of the external pattern. In such a situation, we may visualize parts of the external pattern into the current pattern, hence the dotted black arrows are necessary.

### 0.3.2 Notation for Axiomatization

Axiomatization of each pattern consist of a set of logical axioms that is formed using signature symbols which are taken from *class names*, *property names*, and *individual names*. Here, datatypes are viewed as classes and literals are viewed as individuals. Each logical axiom is either given as a description logic (DL) axiom or a Datalog rule (with equality) as defined in the sequel. In general, we would prefer to use DL notation over rule notation since the former immediately corresponds to OWL syntax. The latter is sometimes needed if it is felt that using it would ease the understanding of the intuitive meaning of the axiom or the improve the readability. Most rules appearing in this document are actually quite straightforward to translate into DL notation.

#### DL notation

An *individual name* semantically represents a particular (named) individual in the object domain. Classes and properties are defined inductively as follows. A *property* is an expression of one of the following form (given together with the semantics):

- $S$ for any property name $S$ — a particular set of pairs of individuals from the object domain (i.e., a binary relation over the object domain);
- $R^-$ where $R$ is some property — the inverse relation of $R$.

A *class* is an expression of one of the following form (given together with the semantics):

- $A$, for any class name $A$ — represents a particular set of individuals
- $\top$, a special class — equivalent to `owl:Thing`
- $\bot$, a special class — equivalent to `owl:Nothing`
- $\{a\}$, where $a$ is an individual name — equivalent to `owl:oneOf`$(a)$
- $C \sqcap D$, where $C, D$ are some classes — the intersection of $C$ and $D$
- $\exists R.C$, where $R$ is some property and $C$ is some class — $\delta$ is an individual that belongs to the class $\exists R.C$, if and only if there exists some individual $\delta'$ such that the pair $(\delta, \delta')$ belongs to $R$ and $\delta'$ belongs to $C$;

- ($\leqslant m\ R.C$), where $m$ is a natural number, $R$ is some property and $C$ is some class — $\delta$ is an individual that belongs to the class ($\leq m\ R.C$), if and only if there exists at most $m$ distinct individuals $\delta_1, \ldots, \delta_m$ such that for every $i = 1, \ldots, m$, $(\delta, \delta_i)$ belongs to $R$ and $\delta_i$ belongs to $C$;
- ($=m\ R.C$), where $m$ is a natural number, $R$ is some property and $C$ is some class — $\delta$ is an individual that belongs to the class ($= m\ R.C$), if and only if there exists exactly $m$ distinct individuals $\delta_1, \ldots, \delta_m$ such that for every $i = 1, \ldots, m$, $(\delta, \delta_i)$ belongs to $R$ and $\delta_i$ belongs to $C$;

A *DL axiom* is a statement of the form $C \sqsubseteq D$, $C \equiv D$, and $R_1 \circ \cdots \circ R_n \sqsubseteq S$ where $n \geq 1$, $C, D$ are classes and $R_i$'s and $S$ are properties. $C \sqsubseteq D$ means that $C$ is a subclass of $D$ (i.e., subset relationship). $C \equiv D$ means that $C$ is an equivalent class of $D$ (i.e., class equivalence/equality). $R_1 \circ R_n \sqsubseteq S$ means that the relation that is obtained by composing $R_1, \ldots, R_n$ is a subproperty of $S$ (i.e., subset relationship between sets of pairs). In case $n = 1$, we have a simple subproperty relationship.

**Datalog/Rule notation**

An *atom* is an expression of the form either $C(x)$, $R(x, y)$, or $x = y$ where $C$ is a class name, $R$ is a property name, and $x, y$ are either individual names or variables. A *rule* is a statement of the form $B_1 \wedge \cdots \wedge B_m \rightarrow H$ where $m \geq 0$, and $H$ and all $B_i$'s are atoms. The conjunction of $B_i$'s is called the *body*, while $H$ is called the *head* of the rule. If $m = 0$, we say that the rule is a *fact*. A rule/fact is *ground* if no variable appears in it. We assume that the rule is *safe*, i.e., each variable occurring in the head occurs somewhere in the body. We sometimes use an expression of the form $B_1 \wedge \cdots \wedge B_m \rightarrow H_1 \wedge \cdots \wedge H_n$ as an abbreviation of a set of $n$ rules: $B_1 \wedge \cdots \wedge B_m \rightarrow H_1, \ldots, B_1 \wedge \cdots \wedge B_m \rightarrow H_n$. Semantics of a rule is a first-order implication in which all variables in the rule are universally quantified over individuals in the object domain and equality is interpreted as the standard equality relation over the object domain (i.e., $x = y$ iff $x$ and $y$ are the same element of the object domain).

**Domain and range restrictions of properties**

Since properties are semantically a binary relation over the object domain, when a property is defined within a particular pattern, we often need to assert classes that cover the domain and range of that property. For example, let $\langle PAT \rangle$ be a pattern in which $R$ is a property and we assert that the class $A$ cover its domain and the class $B$ covers its range. One obvious way to do it then is by asserting the rule $R(x, y) \rightarrow A(x) \wedge B(y)$ in $\langle PAT \rangle$ which is essentially an abbreviation of two rules: the domain restriction $R(x, y) \rightarrow A(x)$ (or $\exists R.\top \sqsubseteq A$ in DL notation) and the range restriction $R(x, y) \rightarrow B(y)$ (or $\exists R^-.\top \sqsubseteq B$ in DL notation). These two rules assert that for all individuals $x$ and $y$, whenever $x$ is connected to $y$ through $R$, then it must be the case that both $x$ belongs to $A$ and $y$ belongs to $B$.

For the purpose of reusability of patterns, however, the above rules force a very strong ontological commitment since they may prevent the reuse of $R$ in other patterns, especially those which does not (wish to) use $A$ or $B$ in it. As an alternative, we will frequently use a *guarded domain restriction* which, in the context of the earlier example, will be of the form $R(x, y) \wedge B(y) \rightarrow A(x)$ (or $\exists R.B \sqsubseteq A$ in DL notation), and a *guarded range restriction* which will be of the form $R(x, y) \wedge A(x) \rightarrow B(y)$ (or $\exists R^-.A \sqsubseteq B$ in DL notation). It is easy to see that if $R$ is used in some other patterns, two individuals connected through $R$ need not be forced to belong $A$ and $B$, respectively. Note that if $R$ is a data property (i.e., a property whose range is a datatype), then its inverse is not supported by OWL 2. For such a property, we will simply express the unguarded range restriction (for which we can simply use `rdfs:range` construct).

## 0.4 Notes for the Group

1. Are there any loose ends, i.e. additional things which need to be modeled before we can start working on mappings and user interface?
2. If possible, rethink class and property names – can we make better naming choices?
3. We need a list of all roles:

- in a cruise;
- in a funding award;
- in an organization (that is relevant for ocean link data)
- in a program (if any)
- in a vessel (if any)

4. We need more details on the external vocabulary for:
    - Vessel ID type (if necessary)
    - File format (MIME types?) for DataFileSet
5. More thinking is needed on modeling properties related to time information (e.g., arrival at ports, vessel's commissioning year, etc.), especially on how difficult it would be in mapping data to patterns. For example, presumably data about a vessel's commissioning year would simply provide a year value, but the pattern models this as an instance of time:Instant which then may be associated with some year value in its description.
6. It might be necessary to put a specific section talking about data-to-pattern alignment issues. For example, the data may contain a property hasAffiliation, but the core patterns model affiliation as a specialization of ⟨PersonalInfoItem⟩ pattern. Thus, a query about affiliation would be translated into a complex expression involving some patterns which might be translated into a simple expression on the data side. This means that there can be two kinds of views:
    - User view: a rule with atomic head and possibly complex body
    - Data view (?): a "rule" with possibly complex head but atomic body.

## 0.5 Further Issues Beyond the General Pattern Modeling

1. We need to produce OWL/RDF versions of course.
2. Which views do we need for the OceanLink application?
3. Which additional extensions of the patterns do we need for OceanLink to work?
4. Provide, separately, mappings from our patterns to established patterns and ontologies.
5. How do we put this work into papers? One option is to target an "In-Use Track" paper for ISWC2014 (deadline probably in May 2014), describing the overall project outcome. Shall we try to publish some of the patterns separately at some other conference?

# 1 ⟨Agent⟩ Pattern

## 1.1 Description

**Pattern URI:** `http://schema.ocean-data.org/agent`

The ⟨Agent⟩ pattern is a very simplistic pattern that is intended to cover instances of both Person and Organization. However, subclass relationships between Person and Agent as well as between Organization and Agent are not axiomatized here: they are done in the subclasses instead. Furthermore, this pattern also does not force an Agent to be either a Person or an Organization, i.e., the existence of an Agent that is neither of those two is allowed. This pattern also states that an Agent may assume some AgentRole.

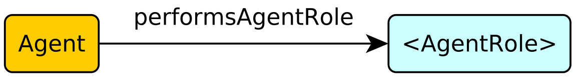

Figure 1.1: ⟨Agent⟩ pattern

## 1.2 Axiomatization

Guarded domain and range restrictions

$$\mathsf{performsAgentRole}(x,y) \wedge \mathsf{Agent}(x) \rightarrow \mathsf{AgentRole}(y) \tag{1.1}$$
$$\mathsf{performsAgentRole}(x,y) \wedge \mathsf{AgentRole}(y) \rightarrow \mathsf{Agent}(x) \tag{1.2}$$

## 1.3 Views

# 2 ⟨AgentRole⟩ pattern

## 2.1 Description

**Pattern URI:** `http://schema.ocean-data.org/agent-role`

The ⟨AgentRole⟩ pattern describes a role that may be performed by an agent within a particular context, i.e., in an organization, a cruise, a project, etc. Such a role is temporally restricted, i.e., an AgentRole starts at one time:Instant and ends at one time:Instant. Here, the granularity of time is left open. Further, we do not collapse starting and ending time:Instants into one time:Interval because, although using time:Interval allows us to ensure the temporal ordering already from OWL Time ontology, such a collapsing would mean that starting and ending times have to be explicitly specified, something which is not required here. So, temporal ordering must be ensured through other mechanisms outside this pattern.

An AgentRole is performed by exactly one Agent and the property performsAgentRole from the ⟨Agent⟩ pattern is the inverse of this relationship. An AgentRole has exactly one AgentRoleType and is a role in exactly one context (we simply use owl:Thing to cover such a context). Also, if an AgentRole has some AgentRoleType and is a role in something, then that thing must provide that AgentRoleType.

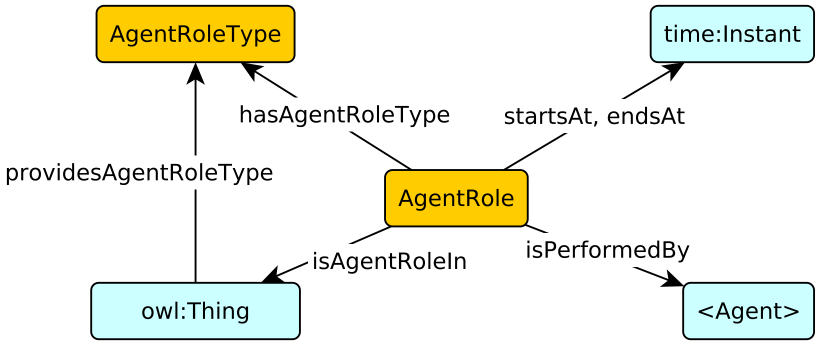

Figure 2.1: ⟨AgentRole⟩ pattern

## 2.2 Axiomatization

We first assert the domain and range restrictions of the properties in this pattern. Specifically for the isAgentRoleIn property, since it ranges over all individuals, range restriction is not needed and its domain restriction is unguarded. For the providesAgentRoleType property, it is the other way around: domain restriction is not needed while its range restriction is unguarded.

$$\text{isPerformedBy}(x,y) \land \text{AgentRole}(x) \to \text{Agent}(y) \tag{2.1}$$
$$\text{isPerformedBy}(x,y) \land \text{Agent}(y) \to \text{AgentRole}(x) \tag{2.2}$$
$$\text{hasAgentRoleType}(x,y) \land \text{AgentRoleType}(y) \to \text{AgentRole}(x) \tag{2.3}$$
$$\text{hasAgentRoleType}(x,y) \land \text{AgentRole}(x) \to \text{AgentRoleType}(y) \tag{2.4}$$
$$\text{startsAt}(x,y) \land \text{time:Instant}(y) \to \text{AgentRole}(x) \tag{2.5}$$
$$\text{startsAt}(x,y) \land \text{AgentRole}(x) \to \text{time:Instant}(y) \tag{2.6}$$
$$\text{endsAt}(x,y) \land \text{time:Instant}(y) \to \text{AgentRole}(x) \tag{2.7}$$
$$\text{endsAt}(x,y) \land \text{AgentRole}(x) \to \text{time:Instant}(y) \tag{2.8}$$

$$\mathsf{isAgentRoleIn}(x,y) \rightarrow \mathsf{AgentRole}(x) \tag{2.9}$$
$$\mathsf{providesAgentRoleType}(x,y) \rightarrow \mathsf{AgentRoleType}(y) \tag{2.10}$$

Next, an AgentRole is performed exactly by one Agent, has exactly one starting time and one ending time, and has exactly one AgentRoleType. Here, AgentRoleType instances are URIs provided by a controlled vocabulary specified later. Also, an AgentRole is an agent role in exactly one thing.

$$\mathsf{AgentRole} \sqsubseteq (=1\ \mathsf{isPerformedBy.Agent}) \tag{2.11}$$
$$\mathsf{AgentRole} \sqsubseteq (=1\ \mathsf{startsAt.time{:}Instant}) \sqcap (=1\ \mathsf{endsAt.time{:}Instant}) \tag{2.12}$$
$$\mathsf{AgentRole} \sqsubseteq (=1\ \mathsf{hasAgentRoleType.AgentRoleType}) \tag{2.13}$$
$$\mathsf{AgentRole} \sqsubseteq (=1\ \mathsf{isAgentRoleIn}.\top) \tag{2.14}$$

Finally, the agent role type of an agent role is provided by the instance such that agent role is a role in it.

$$\mathsf{isAgentRoleIn}(x,y) \wedge \mathsf{hasAgentRoleType}(x,z) \rightarrow \mathsf{providesAgentRoleType}(y,z) \tag{2.15}$$

In this pattern, specific roles (such as PI for projects) are identified by individuals (controlled vocabulary) in the AgentRoleType class, e.g. as follows.

```
_:x          ol:isPerformedBy         ol:tomNarock.
_:x          ol:isAgentRoleIn         ol:OceanLink.
ol:OceanLink ol:providesAgentRoleType ol:PI.
_:x          ol:hasAgentRoleType      ol:PI.
```

Note that our axiomatization does identify the type of roles by means of individuals (using controlled vocabulary) as fillers of the hasAgentRoleType property. For example, e.g., we can axiomatize that each FundingAward (see Chapter 10) provides the PI role below and together with our axiomatization above, this would classify as PI role all agent roles that are a role in a funding award and for which ol:PI is the filler of hasAgentRoleType property.

$$\mathsf{FundingAward} \sqsubseteq \exists \mathsf{providesAgentRoleType.ol{:}PI} \tag{2.16}$$

## 2.3 Views

The following rule provides the property performsAgentRole as a view for this pattern. This property itself is also defined, not as a view, in ⟨Agent⟩ pattern.

$$\mathsf{isPerformedBy}(x,y) \rightarrow \mathsf{performsAgentRole}(y,x) \tag{2.17}$$

# 3 ⟨Event⟩ Pattern Stub

## 3.1 Description

**Pattern URI:** `http://schema.ocean-data.org/event`

This is a simple pattern stub describing events. The modeling approach is based on the Simple Event Model (SEM). Essentially, an event has an event type and occurs at some place and some time. The time information is generic, i.e., can be a time point (time:Instant) or an interval (time:Interval). An activity may happen during an event. Such an activity has a certain activity type and has at least an agent as actor. This pattern is going to be used by $\langle Person \rangle$ pattern (Section 4) to model birth event. The ⟨Cruise⟩ micro-ontology (Section 14) also uses this pattern.

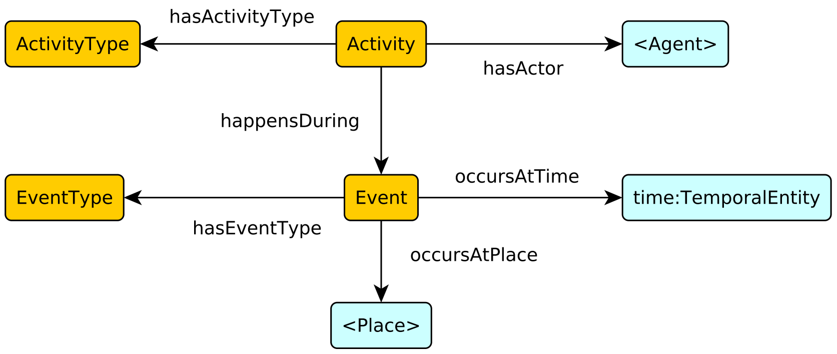

Figure 3.1: ⟨Event⟩ pattern stub

## 3.2 Axiomatization

Guarded domain and range restriction.

$$\mathsf{hasActivityType}(x,y) \wedge \mathsf{ActivityType}(y) \rightarrow \mathsf{Activity}(x) \tag{3.1}$$
$$\mathsf{hasActivityType}(x,y) \wedge \mathsf{Activity}(x) \rightarrow \mathsf{ActivityType}(y) \tag{3.2}$$
$$\mathsf{hasActor}(x,y) \wedge \mathsf{Agent}(y) \rightarrow \mathsf{Activity}(x) \tag{3.3}$$
$$\mathsf{hasActor}(x,y) \wedge \mathsf{Activity}(x) \rightarrow \mathsf{Agent}(y) \tag{3.4}$$
$$\mathsf{happensDuring}(x,y) \wedge \mathsf{Event}(y) \rightarrow \mathsf{Activity}(x) \tag{3.5}$$
$$\mathsf{happensDuring}(x,y) \wedge \mathsf{Activity}(x) \rightarrow \mathsf{Event}(y) \tag{3.6}$$
$$\mathsf{hasEventType}(x,y) \wedge \mathsf{EventType}(y) \rightarrow \mathsf{Event}(x) \tag{3.7}$$
$$\mathsf{hasEventType}(x,y) \wedge \mathsf{Event}(x) \rightarrow \mathsf{EventType}(y) \tag{3.8}$$
$$\mathsf{occursAtTime}(x,y) \wedge \mathsf{time{:}TemporalEntity}(y) \rightarrow \mathsf{Event}(x) \tag{3.9}$$
$$\mathsf{occursAtTime}(x,y) \wedge \mathsf{Event}(x) \rightarrow \mathsf{time{:}TemporalEntity}(y) \tag{3.10}$$
$$\mathsf{occursAtPlace}(x,y) \wedge \mathsf{Place}(y) \rightarrow \mathsf{Event}(x) \tag{3.11}$$
$$\mathsf{occursAtPlace}(x,y) \wedge \mathsf{Event}(x) \rightarrow \mathsf{Place}(y) \tag{3.12}$$

Every activity has exactly one activity type, has some actor and happens during some event. Every event has exactly one event type, occurs at some place and some time.

$$\mathsf{Activity} \sqsubseteq (=1\ \mathsf{hasActivityType}.\mathsf{ActivityType}) \tag{3.13}$$
$$\mathsf{Activity} \sqsubseteq \exists \mathsf{hasActor}.\mathsf{Agent} \tag{3.14}$$
$$\mathsf{Activity} \sqsubseteq \exists \mathsf{happensDuring}.\mathsf{Event} \tag{3.15}$$
$$\mathsf{Event} \sqsubseteq (=1\ \mathsf{hasEventType}.\mathsf{EventType}) \tag{3.16}$$
$$\mathsf{Event} \sqsubseteq \exists \mathsf{occursAtPlace}.\mathsf{Place} \tag{3.17}$$
$$\mathsf{Event} \sqsubseteq \exists \mathsf{occursAtTime}.\mathsf{time{:}TemporalEntity} \tag{3.18}$$

## 3.3 Views

# 4 ⟨Person⟩ Pattern

## 4.1 Description

**Pattern URI:** `http://schema.ocean-data.org/person`

The ⟨Person⟩ pattern describes people (i.e., human beings). Its main class, Person, is a subclass of Agent, hence an instance of Person may be related to an AgentRole through the property performsAgentRole.

When modeling Person, we notice that attributes of a Person are not always permanent: even a person's name may change during his lifetime. The only thing that arguably never changes is birthday. Note that the day/time a person dies also does not change, but for a living person, this information is always unknown. So, we assert that every person has a birthday and birthplace, and to incorporate other non-permanent attributes, we assert that a person may have some credentials and some personal information items which cover things which are time-dependent, e.g., name, address, etc. In this pattern, in particular, we reuse event pattern stub from Section 3 to model a person's birth as an event. The event pattern stub employs the modeling approach from The Simple Event Model (SEM). This is depicted in Figure 4.1.

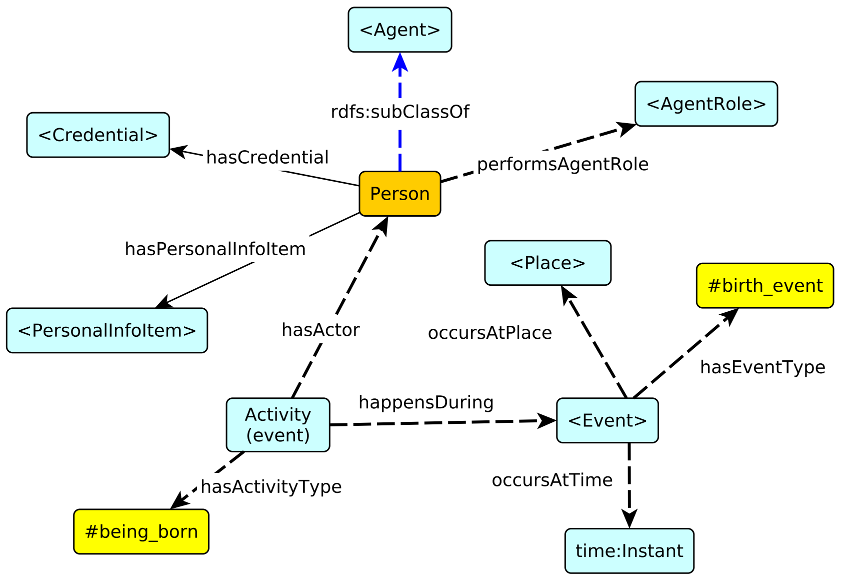

Figure 4.1: ⟨Person⟩ pattern: modeling birth as an event

## 4.2 Axiomatization

**Axioms included from other patterns:** all in Section 1.2; all in Section 3.2.

Every person is an agent, as stated here.

$$\text{Person} \sqsubseteq \text{Agent} \tag{4.1}$$

Next, we axiomatize the domain and range restrictions for the properties in this pattern. Note that the guarded domain and range restrictions for properties from external patterns are already defined in their respective patterns.

$$\mathsf{hasPersonalInfoItem}(x,y) \wedge \mathsf{PersonalInfoItem}(y) \rightarrow \mathsf{Person}(x) \tag{4.2}$$
$$\mathsf{hasPersonalInfoItem}(x,y) \wedge \mathsf{Person}(x) \rightarrow \mathsf{PersonalInfoItem}(y) \tag{4.3}$$
$$\mathsf{hasCredential}(x,y) \wedge \mathsf{Credential}(y) \rightarrow \mathsf{Person}(x) \tag{4.4}$$
$$\mathsf{hasCredential}(x,y) \wedge \mathsf{Person}(x) \rightarrow \mathsf{Credential}(y) \tag{4.5}$$

A person is the actor of a PersonBeingBornActivity. A PersonBeingBornActivity is an activity which has the `#being_born` activity type, has exactly one person actor. Furthermore, there is exactly one BirthEvent during which it happens. A BirthEvent itself is an event which has the `#birth_event` event type and occurs at exactly one time and exactly one place. Note that a PersonBeingBornActivity may happen during other events that are not BirthEvent.

$$\begin{aligned}\mathsf{PersonBeingBornActivity} \equiv{} & \mathsf{Activity} \sqcap \exists \mathsf{hasActivityType}.\{\#\mathtt{being\_born}\} \\ & \sqcap (=1\ \mathsf{hasActor}.\mathsf{Person}) \sqcap (=1\ \mathsf{happensDuring}.\mathsf{BirthEvent}) \qquad (4.6)\\ \mathsf{BirthEvent} \equiv{} & \mathsf{Event} \sqcap \exists \mathsf{hasEventType}.\{\#\mathtt{birth\_event}\} \sqcap (=1\ \mathsf{occursAtTime}.\mathsf{time{:}Instant}) \\ & \sqcap\ (=1\ \mathsf{occursAtPlace}.\mathsf{Place}) \qquad (4.7)\\ \mathsf{Person} \sqsubseteq{} & (=1\ \mathsf{hasActor}^{-}.\mathsf{PersonBeingBornActivity}) \qquad (4.8)\end{aligned}$$

Finally, we assert that a person cannot be an organization. Note that the class Organization does not appear directly in the ⟨Person⟩ pattern.

$$\mathsf{Person} \sqcap \mathsf{Organization} \sqsubseteq \bot \tag{4.9}$$

## 4.3 Views

The following rules provide hasBirthday and hasBirthplace as a view for this pattern.

$$\begin{aligned}&\mathsf{Person}(x) \wedge \mathsf{hasActor}(y,x) \wedge \mathsf{hasActivityType}(y,\#\mathtt{being\_born}) \wedge \mathsf{happensDuring}(y,z)\\ &\quad \wedge \mathsf{hasEventType}(z,\#\mathtt{birth\_event}) \wedge \mathsf{occursAtTime}(z,t) \rightarrow \mathsf{hasBirthday}(x,t) \qquad (4.10)\\ &\mathsf{Person}(x) \wedge \mathsf{hasActor}(y,x) \wedge \mathsf{hasActivityType}(y,\#\mathtt{being\_born}) \wedge \mathsf{happensDuring}(y,z)\\ &\quad \wedge \mathsf{hasEventType}(z,\#\mathtt{birth\_event}) \wedge \mathsf{occursAtPlace}(z,w) \rightarrow \mathsf{hasBirthplace}(x,w) \qquad (4.11)\end{aligned}$$

# 5 ⟨PersonalInfoItem⟩ Pattern

## 5.1 Description

**Pattern URI:** `http://schema.ocean-data.org/personal-info-item`

The ⟨PersonalInfoItem⟩ pattern encapsulates any time-dependent attribute of a Person, for example, name, address, nationality, etc. A PersonalInfoItem starts at exactly one time:Instant and also ends at exactly one time:Instant. Currently, a correct temporal ordering is not enforced by this pattern. A PersonalInfoItem is a personal info item of exactly one Person and the property hasPersonalInfoItem from the ⟨Person⟩ pattern is the inverse of this relationship.

The classes PersonalInfoType and PersonalInfoValue capture the kind of personal info item that we want to describe together with the values allowed for it. For example, `#nationality` is a PersonalInfoType and its allowed PersonalInfoValues are `#american`, `#german`, `#indonesian`, etc. This pattern asserts that a PersonalInfoItem has exactly one PersonalInfoType and exactly one PersonalInfoValue, and furthermore, this PersonalInfoValue has to be one of the allowed values for the PersonalInfoType.

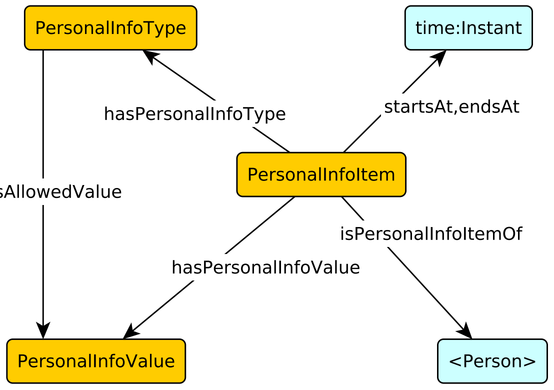

Figure 5.1: ⟨PersonalInfoItem⟩ pattern

## 5.2 Axiomatization

We first axiomatize the guarded domain and range restrictions for properties in this pattern. Note that startsAt and endsAt properties here are different from the ones in Section 2.2 since their URIs are different.

$$\mathsf{isPersonalInfoItemOf}(x,y) \land \mathsf{Person}(y) \to \mathsf{PersonalInfoItem}(y) \tag{5.1}$$
$$\mathsf{isPersonalInfoItemOf}(x,y) \land \mathsf{PersonalInfoItem}(x) \to \mathsf{Person}(y) \tag{5.2}$$
$$\mathsf{startsAt}(x,y)\mathsf{time{:}Instant}(y)\land \to \mathsf{PersonalInfoItem}(x) \tag{5.3}$$
$$\mathsf{startsAt}(x,y) \land \mathsf{PersonalInfoItem}(x) \to \mathsf{time{:}Instant}(y) \tag{5.4}$$
$$\mathsf{endsAt}(x,y) \land \mathsf{time{:}Instant}(y) \to \mathsf{PersonalInfoItem}(x) \tag{5.5}$$
$$\mathsf{endsAt}(x,y) \land \mathsf{PersonalInfoItem}(x) \to \mathsf{time{:}Instant}(y) \tag{5.6}$$
$$\mathsf{hasPersonalInfoType}(x,y) \land \mathsf{PersonalInfoType}(y) \to \mathsf{PersonalInfoItem}(x) \tag{5.7}$$
$$\mathsf{hasPersonalInfoType}(x,y) \land \mathsf{PersonalInfoItem}(x) \to \mathsf{PersonalInfoType}(x) \tag{5.8}$$
$$\mathsf{hasPersonalInfoValue}(x,y) \land \mathsf{PersonalInfoValue}(y) \to \mathsf{PersonalInfoItem}(x) \tag{5.9}$$
$$\mathsf{hasPersonalInfoValue}(x,y) \land \mathsf{PersonalInfoItem}(x) \to \mathsf{PersonalInfoValue}(y) \tag{5.10}$$

$$\mathsf{hasAllowedValue}(x, y) \land \mathsf{PersonalInfoValue}(y) \to \mathsf{PersonalInfoType}(x) \tag{5.11}$$
$$\mathsf{hasAllowedValue}(x, y) \land \mathsf{PersonalInfoType}(x) \to \mathsf{PersonalInfoValue}(y) \tag{5.12}$$

Every PersonalInfoItem is a personal information item of exactly one person. Every PersonalInfoItem must have exactly one starting time, one ending time, one PersonalInfoType and one PersonalInfoValue. Note here that PersonalInfoValue may contain controlled vocabularies, literal values, or even be specialized through a separate pattern.

$$\mathsf{PersonalInfoItem} \sqsubseteq (=1\ \mathsf{isPersonalInfoItemOf.Person}) \tag{5.13}$$
$$\mathsf{PersonalInfoItem} \sqsubseteq (=1\ \mathsf{startsAt.time{:}Instant}) \sqcap (=1\ \mathsf{endsAt.time{:}Instant}) \tag{5.14}$$
$$\mathsf{PersonalInfoItem} \sqsubseteq (=1\ \mathsf{hasPersonalInfoType.PersonalInfoType}) \tag{5.15}$$
$$\mathsf{PersonalInfoItem} \sqsubseteq (=1\ \mathsf{hasPersonalInfoValue.PersonalInfoValue}) \tag{5.16}$$

Every PersonalInfoType has at least one choice of allowed PersonalInfoValue.

$$\mathsf{PersonalInfoType} \sqsubseteq \exists \mathsf{hasAllowedValue.PersonalInfoValue} \tag{5.17}$$

The personal info value of any personal info item must be some allowed value of the corresponding personal info type.

$$\mathsf{hasPersonalInfoType}(x, y) \land \mathsf{hasPersonalInfoValue}(x, z) \to \mathsf{hasAllowedValue}(y, z) \tag{5.18}$$

## 5.3 Views

The property hasPersonalInfoItem is provided as a view for this pattern by the following rule. In the ⟨Person⟩ pattern, this property is defined not as a view.

$$\mathsf{isPersonalInfoItemOf}(x, y) \to \mathsf{hasPersonalInfoItem}(y, x) \tag{5.19}$$

## 5.4 Notes

The following is an example (i.e., not necessarily a part of what we need for OceanLink) of a specialization of PersonalInfoItem for nationality. Here, #`nationality` is an instance of PersonalInfoType and Nationality is a subclass of PersonalInfoValue. Thus, the following axioms must be satisfied.

$$\mathsf{PersonalInfoType}(\#\texttt{nationality})$$
$$\mathsf{Nationality} \sqsubseteq \mathsf{PersonalInfoValue}$$
$$\mathsf{hasAllowedValue}(\#\texttt{nationality}, y) \to \mathsf{Nationality}(y)$$

# 6 ⟨PersonName⟩ Pattern

## 6.1 Description

The ⟨PersonName⟩ pattern is a specialization of the ⟨PersonalInfoItem⟩ pattern that describes a person's name. We use #`person_name` as an instance of PersonalInfoType to represent the personal info type of name. Then, we use the class PersonName to represent all kinds of person names, each of which corresponds to three string values: the full name, the first/given name, and the family name.

We are aware that names and their usages are very culturally dependent. Making a versatile pattern for person names which is applicable in a multitude of cultural, national, and legislative contexts is a formidable challenge in its own right, and we consider this out of scope for our current purposes. In this sense, our person name pattern is a stub pattern, i.e. it is not fully developed. The intended usage is that fullNameAsString points to the full official name of the person, transcribed to latin letters, while firstOrGivenName and familyOrSurname point to the corresponding two parts of the name as that person would usually give them. While this is not a satisfactory solution for many contexts, it will serve our purpose for now – and we understand that a more sophisticated solution, which is out of scope for us at this stage, would be preferable.

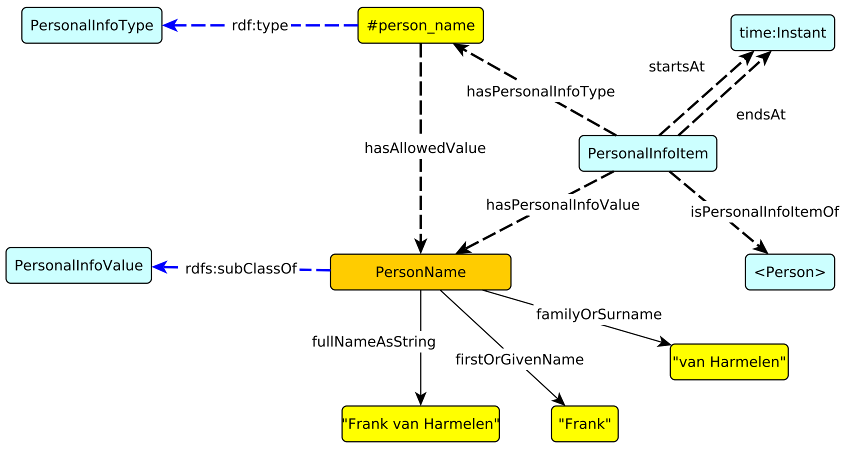

Figure 6.1: ⟨PersonName⟩ pattern

## 6.2 Axiomatization

**Axioms included from other patterns:** (5.1), (5.2), (5.3), (5.4), (5.5), (5.6), (5.7), (5.8), (5.9), (5.10), (5.11), (5.12), (5.13), (5.14), (5.15), (5.16), (5.17), (5.18). All of these are because ⟨PersonName⟩ is a specialization of ⟨PersonalInfoItem⟩.

In addition to the included axioms above, specializing ⟨PersonalInfoItem⟩ into ⟨PersonName⟩ is also realized by asserting a controlled vocabulary #`person_name` as a particular PersonalInfoType, setting the class PersonName as a subclass of PersonalInfoValue and ensuring allowed values of the type #`person_name` are only taken from PersonName.

$$\mathsf{PersonalInfoType}(\#\texttt{person\_name}) \tag{6.1}$$

$$\mathsf{PersonName} \sqsubseteq \mathsf{PersonalInfoValue} \tag{6.2}$$

$$\mathsf{hasAllowedValue}(\#\texttt{person\_name}, y) \rightarrow \mathsf{PersonName}(y) \tag{6.3}$$

Note that the ground fact given by (9.1) is equivalent to the following RDF triple:

$$\#\texttt{person\_name}\ \texttt{rdf:type}\ \texttt{PersonalInfoType}.$$

Next, we assert domain and range restrictions for fullNameAsString, firstOrGivenName, and familyOrSurname properties. Note that these three properties are data properties, hence, as noted in Section 0.3.2, we use the unguarded variant of range restriction here.

$$\mathsf{fullNameAsString}(x,y) \wedge \mathsf{xsd{:}string}(y) \to \mathsf{PersonName}(x) \tag{6.4}$$
$$\mathsf{fullNameAsString}(x,y) \to \mathsf{xsd{:}string}(y) \tag{6.5}$$
$$\mathsf{firstOrGivenName}(x,y) \wedge \mathsf{xsd{:}string}(y) \to \mathsf{PersonName}(x) \tag{6.6}$$
$$\mathsf{firstOrGivenName}(x,y) \to \mathsf{xsd{:}string}(y) \tag{6.7}$$
$$\mathsf{familyOrSurname}(x,y) \wedge \mathsf{xsd{:}string}(y) \to \mathsf{PersonName}(x) \tag{6.8}$$
$$\mathsf{familyOrSurname}(x,y) \to \mathsf{xsd{:}string}(y) \tag{6.9}$$

Finally, every PersonName has exactly 1 full name string, at most one first/given name and at most one family/surname.

$$\begin{aligned}\mathsf{PersonName} \sqsubseteq\ & (=1\ \mathsf{fullNameAsString.xsd{:}string}) \sqcap (\leqslant 1\ \mathsf{firstOrGivenName.xsd{:}string}) \\ & \sqcap\ (\leqslant 1\ \mathsf{familyOrSurname.xsd{:}string})\end{aligned} \tag{6.10}$$

**Remarks:** Together with the ⟨Person⟩ pattern (Section 4), this pattern does not model the constraint that every person has to have a name at any given time. Furthermore, this pattern does not model the uniqueness of a person's full name.

## 6.3 Views

**Views included from other patterns:** (5.19).

In addition, the properties personGivenName, personSurname, and hasCanonicalName are views for the ⟨PersonName⟩ pattern by the following rules. Note that hasCanonicalName is also part of the InformationObject pattern stub (Section 7).

$$\mathsf{hasPersonalInfoItem}(w,x) \wedge \mathsf{hasPersonalInfoValue}(x,y) \wedge \mathsf{PersonName}(y) \to \mathsf{hasPNameInstance}(w,y) \tag{6.11}$$
$$\mathsf{hasPNameInstance}(x,y) \wedge \mathsf{firstOrGivenName}(y,z) \to \mathsf{hasGivenName}(x,z) \tag{6.12}$$
$$\mathsf{hasPNameInstance}(x,y) \wedge \mathsf{familyOrSurname}(y,z) \to \mathsf{hasSurname}(x,z) \tag{6.13}$$
$$\mathsf{hasPNameInstance}(x,y) \wedge \mathsf{fullNameAsString}(y,z) \to \mathsf{hasFullName}(x,z) \tag{6.14}$$
$$\mathsf{hasFullName}(x,y) \to \mathsf{hasCanonicalName}(x,y) \tag{6.15}$$

# 7 ⟨InformationObject⟩ Pattern Stub

## 7.1 Description

We use the template in Figure 7.1 as a pattern stub to indicate additional information given for an object. Essentially, any thing can be described by an information object related to the thing, the information object in turn carries adornment such as descriptions or webpages, we indicate throughout this document, which adornments can be carried. This use of information objects follows good modeling practice.

Everything can have at most one information object associated with it. At a later stage, we will be able to develop information objects into a full pattern.

Currently, this pattern stub provides a number of properties whose range is either strings or URIs. The property hasDescription provides a simple string description of any instance. The properties hasWebpage and rdfs:seeAlso provide URL information that can be used to find further information. The property hasCanonicalName provides the canonical name of an instance as string. The property alsoKnownAs provides a string that can be used as an alternative name aside from the one provided by hasCanonicalName.

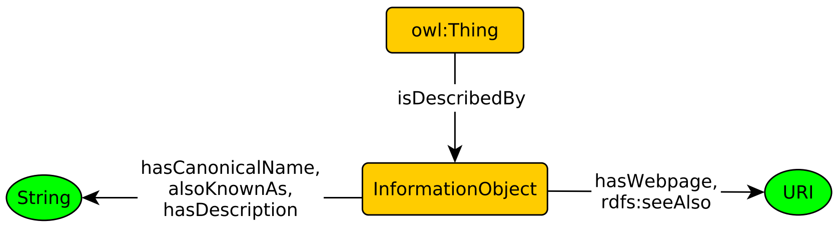

Figure 7.1: Information object stub

## 7.2 Axiomatization

For InformationObject, we only assert that everything can only be described by at most one InformationObject, while any InformationObject describes exactly one thing. The remaining properties are data properties. So, we assert guarded domain restriction and unguarded range restrictions. For rdfs:seeAlso, we assert an unguarded range restriction only.

$$\mathsf{isDescribedBy}(x,y) \rightarrow \mathsf{InformationObject}(y) \tag{7.1}$$

$$\top \sqsubseteq (\leqslant 1\ \mathsf{isDescribedBy}.\mathsf{InformationObject}) \tag{7.2}$$

$$\mathsf{InformationObject} \sqsubseteq (=1\ \mathsf{isDescribedBy}^{-}.\top) \tag{7.3}$$

$$\mathsf{hasCanonicalName}(x,y) \land \mathsf{xsd{:}string}(y) \rightarrow \mathsf{InformationObject}(x) \tag{7.4}$$

$$\mathsf{hasCanonicalName}(x,y) \rightarrow \mathsf{xsd{:}string}(y) \tag{7.5}$$

$$\mathsf{alsoKnownAs}(x,y) \land \mathsf{xsd{:}string}(y) \rightarrow \mathsf{InformationObject}(x) \tag{7.6}$$

$$\mathsf{alsoKnownAs}(x,y) \rightarrow \mathsf{xsd{:}string}(y) \tag{7.7}$$

$$\mathsf{hasDescription}(x,y) \land \mathsf{xsd{:}string}(y) \rightarrow \mathsf{InformationObject}(x) \tag{7.8}$$

$$\mathsf{hasDescription}(x,y) \rightarrow \mathsf{xsd{:}string}(y) \tag{7.9}$$

$$\mathsf{hasWebpage}(x,y) \land \mathsf{xsd{:}anyURI}(y) \rightarrow \mathsf{InformationObject}(x) \tag{7.10}$$

$$\mathsf{hasWebpage}(x,y) \rightarrow \mathsf{xsd{:}anyURI}(y) \tag{7.11}$$

$$\mathsf{rdfs{:}seeAlso}(x,y) \rightarrow \mathsf{xsd{:}anyURI}(y) \tag{7.12}$$

# 8 Organization Pattern

## 8.1 Description

This pattern describes organizations, and like Person, Organization is a subclass of Agent, hence inherits all of its properties and its axiomatization. This pattern also employs ⟨InformationObject⟩ stub, and adds an axiomatization that enforces an Organization to have exactly one official name. An Organization may have a sub-organization. Also, an Organization provides some AgentRoleType which is a class within the ⟨AgentRole⟩ pattern, expressed by reusing the providesAgentRoleType property from the ⟨AgentRole⟩ pattern, and hence, inheriting the axiomatization from that pattern that is relevant for this particular property.

Let us briefly elaborate a bit on this to understand the difference between providesAgentRoleType and performsAgentRole. The latter, performsAgentRole is inherited from the Agent pattern. It indicates that an organization (being an agent) can assume a role, e.g. in a second organization. This role, in turn would have to be of a type provided by this second organization through the providesAgentRoleType property.

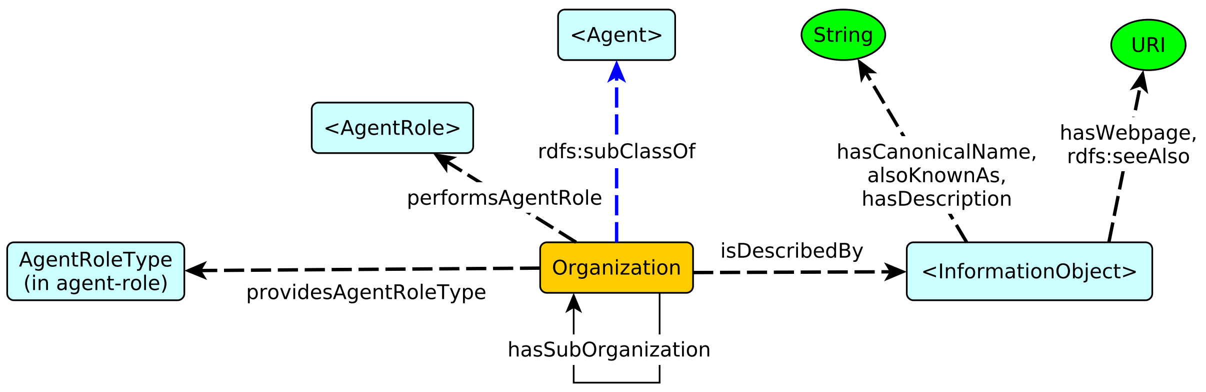

Figure 8.1: ⟨Organization⟩ pattern with Information Object

## 8.2 Axiomatization

**Axioms included from other patterns:** (1.1), (1.2), (4.9), (2.10), (7.1), (7.2), (7.3), (7.4), (7.5), (7.6), (7.7), (7.8), (7.9), (7.10), (7.11), (7.12).

Axioms (1.1) and (1.2) are included since Organization is a subclass of Agent. Axiom (4.9) is included because we wish to enforce that Organization is disjoint with Person. Axiom (2.10) is included because the providesAgentRoleType property from the ⟨AgentRole⟩ pattern is used. Axioms (7.1), (7.2), (7.3), (7.4), (7.5), (7.6), (7.7), (7.8), (7.9), (7.10), (7.11), (7.12) are included since ⟨Organization⟩ reuses the ⟨InformationObject⟩ pattern stub.

In addition, we assert the guarded domain and range restriction for hasSubOrganization property.

$$\text{hasSubOrganization}(x, y) \land \text{Organization}(y) \rightarrow \text{Organization}(x) \tag{8.1}$$
$$\text{hasSubOrganization}(x, y) \land \text{Organization}(x) \rightarrow \text{Organization}(y) \tag{8.2}$$

We also ensure that an Organization provides some AgentRoleType, and is described by some InformationObject which only has exactly one canonical name.

$$\mathsf{Organization} \sqsubseteq \exists \mathsf{providesAgentRoleType}.\mathsf{AgentRoleType} \tag{8.3}$$

$$\mathsf{Organization} \sqsubseteq \exists \mathsf{isDescribedBy}.\mathsf{InformationObject} \tag{8.4}$$

$$\mathsf{InformationObject} \sqcap \exists \mathsf{isDescribedBy}^{-}.\mathsf{Organization} \sqsubseteq (= 1\ \mathsf{hasCanonicalName}.\mathsf{xsd{:}string}) \tag{8.5}$$

## 8.3 Views

We have the following useful views.

$$\mathsf{Organization}(x) \land \mathsf{isDescribedBy}(x, y) \land \mathsf{hasCanonicalName}(y, z) \to \mathsf{hasOfficialName}(x, z) \tag{8.6}$$

$$\mathsf{hasSubOrganization}(x, y) \to \mathsf{subOrganizationOf}(y, x) \tag{8.7}$$

# 9 ⟨PersonAffiliation⟩ Pattern

## 9.1 Description

The ⟨PersonalInfoItem⟩ pattern is a specialization of the ⟨PersonalInfoItem⟩ pattern that describes a person's affiliation. We use `#person_affiliation` as an instance of PersonalInfoType to represent the personal info type of affiliation. Then, we use the class PersonAffiliation to represent all kinds of person's affiliation. A person's affiliation includes, e.g., organizations. The subclass relationship with PersonAffiliation is, however, not asserted here, but rather in the subclass' pattern.

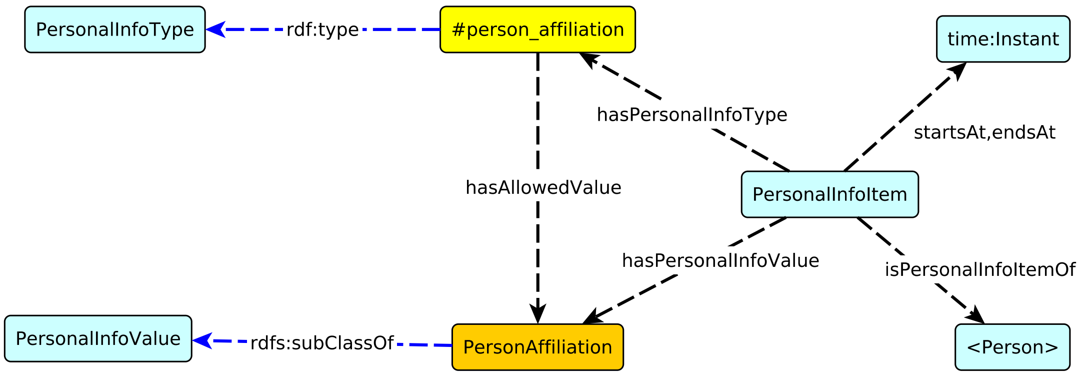

Figure 9.1: ⟨PersonAffiliation⟩ pattern

## 9.2 Axiomatization

**Axioms included from other patterns:** (5.1), (5.2), (5.3), (5.4), (5.5), (5.6), (5.7), (5.8), (5.9), (5.10), (5.11), (5.12), (5.13), (5.14), (5.15), (5.16), (5.17), (5.18).

In addition to the included axioms above, specializing ⟨PersonalInfoItem⟩ into ⟨PersonAffiliation⟩ is also realized by asserting a controlled vocabulary `#person_affiliation` as a particular PersonalInfoType, setting the class PersonAffiliation as a subclass of PersonalInfoValue and ensuring allowed values of the type `#person_affiliation` are only taken from PersonName.

$$\mathsf{PersonalInfoType}(\#\texttt{person\_affiliation}) \tag{9.1}$$

$$\mathsf{PersonAffiliation} \sqsubseteq \mathsf{PersonalInfoValue} \tag{9.2}$$

$$\mathsf{hasAllowedValue}(\#\texttt{person\_affiliation}, y) \rightarrow \mathsf{PersonAffiliation}(y) \tag{9.3}$$

**Remark:** In most cases, when a person is affiliated to an organization, this would entail the membership of that person to the organization which can be modeled through ⟨AgentRole⟩ pattern (Section 2). However, this relationship is not asserted here.

## 9.3 Views

Using (5.19), we can make hasAffiliation as a view:

$$\mathsf{hasPersonalInfoItem}(x, y) \wedge \mathsf{hasPersonalInfoValue}(y, z) \wedge \mathsf{PersonAffiliation}(z) \rightarrow \mathsf{hasAffiliation}(x, z) \tag{9.4}$$

# 10 ⟨FundingAward⟩ pattern

## 10.1 Description

The ⟨FundingAward⟩ pattern describes the awards that fund all kinds of ocean science research activities. ⟨InformationObject⟩ pattern stub is reused by this pattern through subclassing. Such a subclass of InformationObject has additional constraint that each FundingAward must have exactly one official/canonical name. To that subclass of InformationObject, we attach two additional pieces of information: exactly one unique string value that identifies the FundingAward; and 0 or more (short) description document.

Each FundingAward also has exactly one starting and ending date, both of which are time:Instants. It provides 0 or more AgentRoleTypes and at most one award amount. At least one Agent funds the award. Finally, we introduce isFundedBy property that can be used to connect anything to a funding award if the funding award funds it.

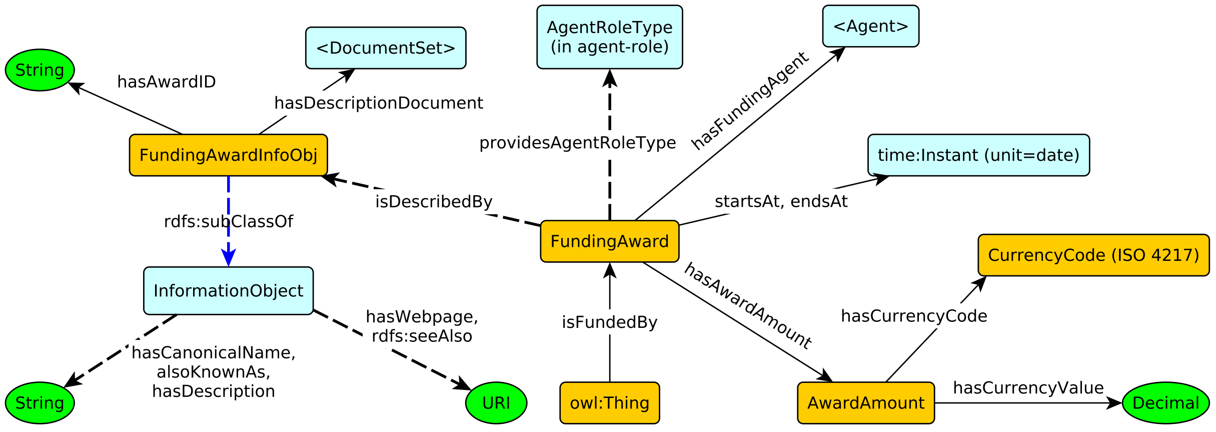

Figure 10.1: ⟨FundingAward⟩ pattern with information object

## 10.2 Axiomatization

**Axioms included from other patterns:** (2.10) — due to the use of providesAgentRoleType property; (7.1), (7.2), (7.3), (7.4), (7.5), (7.6), (7.7), (7.8), (7.9), (7.10), (7.11), (7.12) — due to FundingAwardInfoObj being a subclass of InformationObject.

We assert domain and range restrictions for properties defined in this pattern. Note that startsAt, endsAt properties here are different from the one in Section 2 and 5 as they have different URIs. In addition, isDescribedBy is given a guarded domain and range restrictions that are specific for ⟨FundingAward⟩ pattern. Also, isFundedBy property has only an unguarded range restriction and no domain restriction, whereas data properties hasAwardID and hasCurrencyValue only have unguarded range restriction.

$$\mathsf{isFundedBy}(x,y) \rightarrow \mathsf{FundingAward}(y) \tag{10.1}$$
$$\mathsf{startsAt}(x,y) \wedge \mathsf{time{:}Instant}(y) \rightarrow \mathsf{FundingAward}(x) \tag{10.2}$$
$$\mathsf{startsAt}(x,y) \wedge \mathsf{FundingAward}(x) \rightarrow \mathsf{time{:}Instant}(y) \tag{10.3}$$
$$\mathsf{endsAt}(x,y) \wedge \mathsf{time{:}Instant}(y) \rightarrow \mathsf{FundingAward}(x) \tag{10.4}$$
$$\mathsf{endsAt}(x,y) \wedge \mathsf{FundingAward}(x) \rightarrow \mathsf{time{:}Instant}(y) \tag{10.5}$$

$$\mathsf{isDescribedBy}(x,y) \wedge \mathsf{FundingAwardInfoObj}(y) \rightarrow \mathsf{FundingAward}(x) \tag{10.6}$$
$$\mathsf{isDescribedBy}(x,y) \wedge \mathsf{FundingAward}(x) \rightarrow \mathsf{FundingAwardInfoObj}(y) \tag{10.7}$$
$$\mathsf{hasAwardID}(x,y) \wedge \mathsf{xsd{:}string}(y) \rightarrow \mathsf{FundingAwardInfoObj}(x) \tag{10.8}$$
$$\mathsf{hasAwardID}(x,y) \rightarrow \mathsf{xsd{:}string}(y) \tag{10.9}$$
$$\mathsf{hasDescriptionDocument}(x,y) \wedge \mathsf{DocumentSet}(y) \rightarrow \mathsf{FundingAwardInfoObj}(x) \tag{10.10}$$
$$\mathsf{hasDescriptionDocument}(x,y) \wedge \mathsf{FundingAwardInfoObj}(x) \rightarrow \mathsf{DocumentSet}(y) \tag{10.11}$$
$$\mathsf{hasFundingAgent}(x,y) \wedge \mathsf{Agent}(y) \rightarrow \mathsf{FundingAward}(x) \tag{10.12}$$
$$\mathsf{hasFundingAgent}(x,y) \wedge \mathsf{FundingAward}(x) \rightarrow \mathsf{Agent}(y) \tag{10.13}$$
$$\mathsf{hasAwardAmount}(x,y) \wedge \mathsf{AwardAmount}(y) \rightarrow \mathsf{FundingAward}(x) \tag{10.14}$$
$$\mathsf{hasAwardAmount}(x,y) \wedge \mathsf{FundingAward}(x) \rightarrow \mathsf{AwardAmount}(y) \tag{10.15}$$
$$\mathsf{hasCurrencyValue}(x,y) \wedge \mathsf{xsd{:}decimal}(y) \rightarrow \mathsf{AwardAmount}(x) \tag{10.16}$$
$$\mathsf{hasCurrencyValue}(x,y) \rightarrow \mathsf{xsd{:}decimal}(x) \tag{10.17}$$
$$\mathsf{hasCurrencyCode}(x,y) \wedge \mathsf{CurrencyCode}(y) \rightarrow \mathsf{AwardAmount}(x) \tag{10.18}$$
$$\mathsf{hasCurrencyCode}(x,y) \wedge \mathsf{AwardAmount}(x) \rightarrow \mathsf{CurrencyCode}(y) \tag{10.19}$$

To ensure a subclass relationship between FundingAwardInfoObj and InformationObject, we assert:

$$\mathsf{FundingAwardInfoObj} \sqsubseteq \mathsf{InformationObject} \tag{10.20}$$

We also assert that each funding award is described by a FundingAwardInfoObj (exactly one due to Axiom (7.2)) which has exactly one award identifier that is unique, 0 or more description document, and exactly one canonical name. The funding award has exactly one starting time, exactly one ending time, at least one funding agent, and at most one award amount. The award amount corresponds to exactly one currency value and one currency code.

$$\mathsf{FundingAward} \sqsubseteq \exists \mathsf{isDescribedBy}.\mathsf{FundingAwardInfoObj} \tag{10.21}$$
$$\mathsf{FundingAwardInfoObj} \sqsubseteq (=1\ \mathsf{hasAwardID}.\mathsf{xsd{:}string}) \sqcap (=1\ \mathsf{hasCanonicalName}.\mathsf{xsd{:}string}) \tag{10.22}$$
$$\top \sqsubseteq (\leqslant 1\ \mathsf{hasAwardID}^{-}.\top) \tag{10.23}$$
$$\mathsf{FundingAward} \sqsubseteq (=1\ \mathsf{startsAt}.\mathsf{time{:}Instant}) \sqcap (=1\ \mathsf{endsAt}.\mathsf{time{:}Instant}) \tag{10.24}$$
$$\mathsf{FundingAward} \sqsubseteq \exists \mathsf{hasFundingAgent}.\mathsf{Agent} \tag{10.25}$$
$$\mathsf{FundingAward} \sqsubseteq (\leqslant 1\ \mathsf{hasAwardAmount}.\mathsf{AwardAmount}) \tag{10.26}$$
$$\mathsf{AwardAmount} \sqsubseteq (=1\ \mathsf{hasCurrencyValue}.\mathsf{xsd{:}decimal}) \sqcap (=1\ \mathsf{hasCurrencyCode}.\mathsf{CurrencyCode}) \tag{10.27}$$

**Remarks:** Note that hasAwardID is seen as an OWL data property (as the range is xsd:string which is a datatype). However, OWL 2 DL does not allow one to express an inverse functional data property. This makes axiom (10.23) not enforceable in the OWL translation of this pattern's axiomatization.

## 10.3 Views

The official name of a funding award.

$$\mathsf{FundingAward}(x) \wedge \mathsf{isDescribedBy}(x,y) \wedge \mathsf{hasCanonicalName}(y,z) \rightarrow \mathsf{hasOfficialName}(x,z) \tag{10.28}$$

# 11 ⟨Program⟩ Pattern Stub

## 11.1 Description

A program (in the oceanographic sense) is a loose collection of things, including cruises, funding awards, activities, events, which are loosely grouped together. We make use of the information object and add only one additional property for associating a program to a thing.

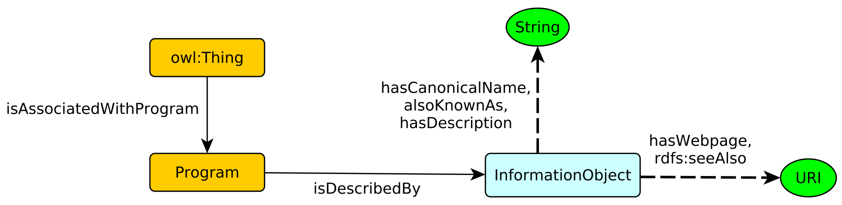

Figure 11.1: ⟨Program⟩ pattern

## 11.2 Axiomatization

$$\mathsf{isAssociatedWithProgram}(x,y) \rightarrow \mathsf{Program}(y) \tag{11.1}$$

$$\mathsf{Program} \sqsubseteq (=1\ \mathsf{isDescribedBy}.\mathsf{InformationObject}) \tag{11.2}$$

# 12 ⟨Trajectory⟩ pattern

## 12.1 Description

We reuse the trajectory pattern from [Hu et al., 2013], it will be modified as needed for our cruise pattern. A depiction of the pattern is given in Figure 12.1. We only differ slightly in that we make the nextFix property part of the core pattern description itself, while in [Hu et al., 2013] it was defined in terms of axiomatization. We assume that for the purposes of OceanLink it is more important to have this property directly available for mapping.

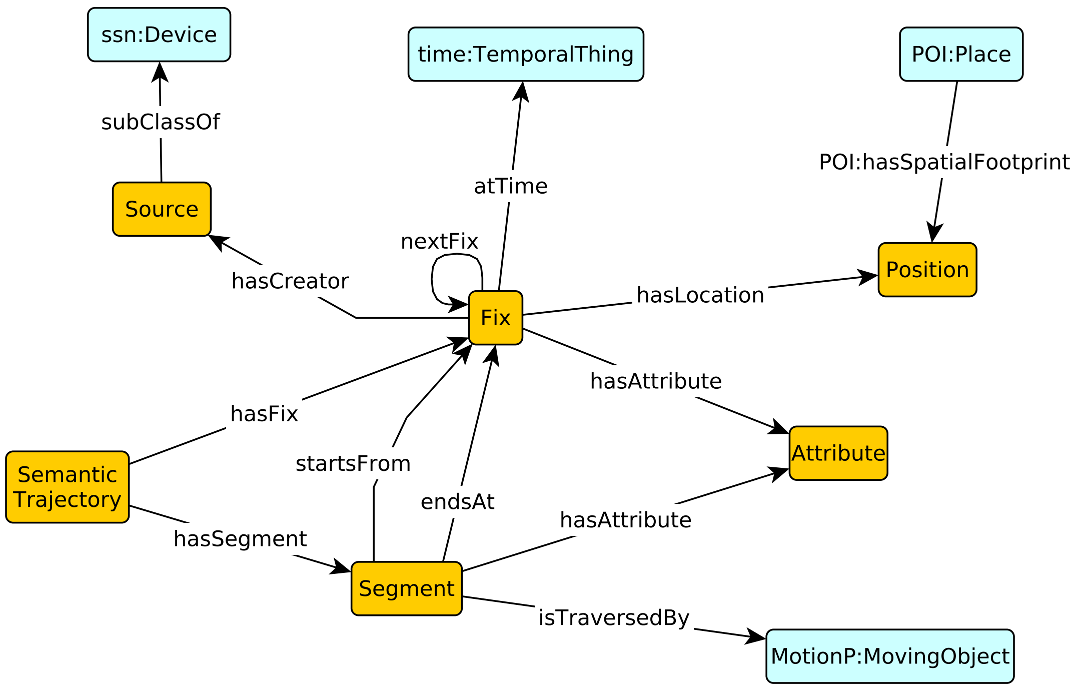

Figure 12.1: ⟨Trajectory⟩ pattern

The axiomatization of the pattern can be taken directly from [Hu et al., 2013], however at this stage only part of this pattern will be relevant to OceanLink: See Section 14.

In [Hu et al., 2013] it was mentioned that GeoSPARQL can be used to describe such trajectories.

## 12.2 Remarks

Note: Sort relationship to ⟨Geometry⟩, in particular using GeoSPARQL. Look into Web Feature Service (WFS).

## 13 ⟨Place⟩ and ⟨Geometry⟩ patterns

*For the moment, we simply use a stub/placeholder plus the generic information object. We model* Port *as a subclass of* Place*. In the future, we have to look at Gazetteer, GeoSPARQL and Web Feature Service (WFS) in modeling these two patterns.*

# 14 ⟨Cruise⟩ Micro-ontology

## 14.1 Description

### 14.1.1 Core Part of ⟨Cruise⟩

⟨Cruise⟩ describes ocean science cruises. Intuitively, the notion of ocean science cruise is rather too specific since one can obviously also think of sight-seeing cruises, pleasure cruises, or even science cruises which are not used for ocean science purposes. In this context, to develop a pattern that is highly reusable, the generic notion of cruise would be a better candidate than ocean science cruise. However, for the purpose of OceanLink project, rather than developing such a pattern, we opt to consider ⟨Cruise⟩ as a micro-ontology that is obtained by combining and reusing existing patterns. This is done through established modeling practices while keeping the amount of abstract ontological commitments to a minimum. To simplify the presentation, some of the classes and relations from reused patterns are shown as imports, while they will be aligned by local classes and roles.

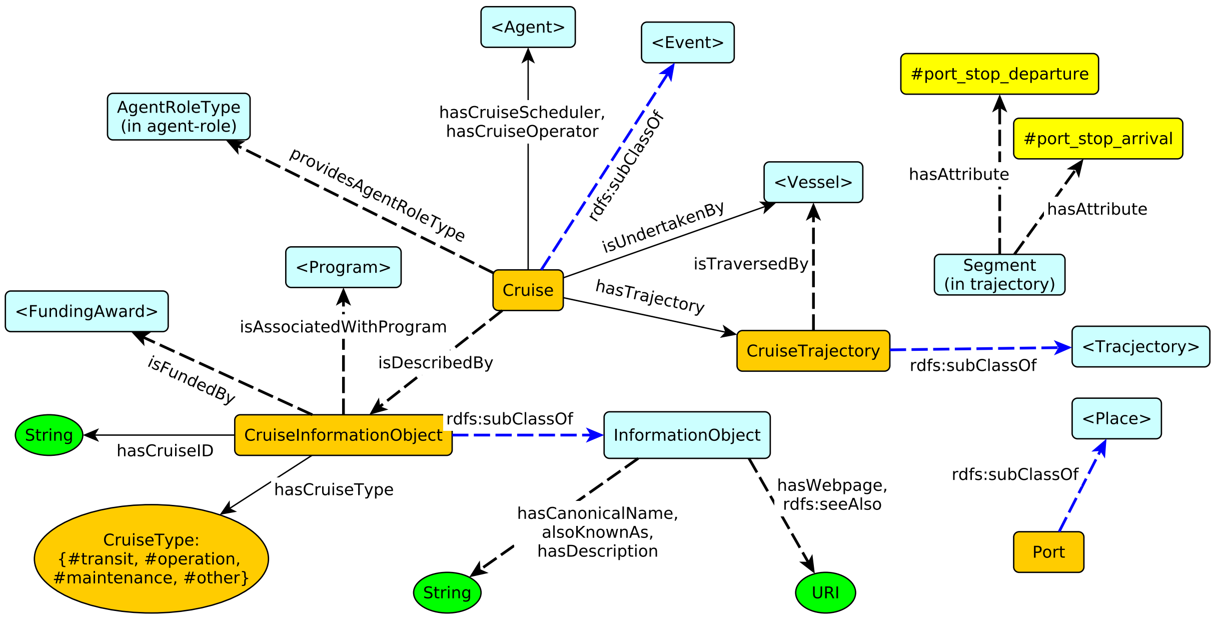

Figure 14.1: ⟨Cruise⟩ pattern

First, a Cruise has exactly one information object associated to it. Information objects (and their realizations) form one of the most commonly used ontology design patterns established in previous work. The information object describes the cruise in terms of: a cruise ID (a string) which is unique for each cruise, i.e., no two cruises share the same cruise ID (i.e., hasCruiseID is both *functional* and *inverse functional*); exactly one cruise type which is either 'transit', 'operational', 'maintenance' or 'other'; the official name and alternative names; textual descriptions of the cruise; associated Websites; associated funding awards; and associated programs. In the future the information object can be aligned to the W3C Prov-O.

A Cruise can have (0 or more) operators and schedulers, all of which are Agents. A Cruise can also provide (0 or more) AgentRoleTypes. Also, a Cruise provides a role type called "chief scientist" and has at least one FundingAward associated with it.

Cruise are events which distinguishes them from the information objects mentioned above, as well as physical objects, such as vessels. Each Cruise is undertaken by exactly one Vessel. This follows well established modeling practice in making the relation between events, objects, actors, and places explicit. Note that the aforementioned cruise scheduler, cruise operator, and chief scientist can be considered as actors for the cruise.

A cruise also has exactly one track which is modeled using the Trajectory pattern (see Section 12 and Section 14.1.2). The trajectory pattern is multi-granular and thus can be used to model the path along which the vessel is traveling as well as certain landmarks such as used for the port sequence. The trajectory pattern also takes care of modeling time stamps, locations, fixes, the source that established these fixes, and segments between fixes. It also provides hooks to other patterns to model Points Of Interests, transportation, sensors, and so forth. The trajectory pattern provides the basic vocabulary and OWL axiomatization to model the port sequence, port fixes, arrival times, and departure times. Therefore, the only required additions is some controled vocabulary to denote special fixes, #portStopArrival and #portStopDeparture, and the Port class as subclass of Place.

### 14.1.2 Trajectories for Cruises

While the full trajectory pattern can be used, at this stage of OceanLink only the part depicted in Figure 14.2 appears to be relevant.

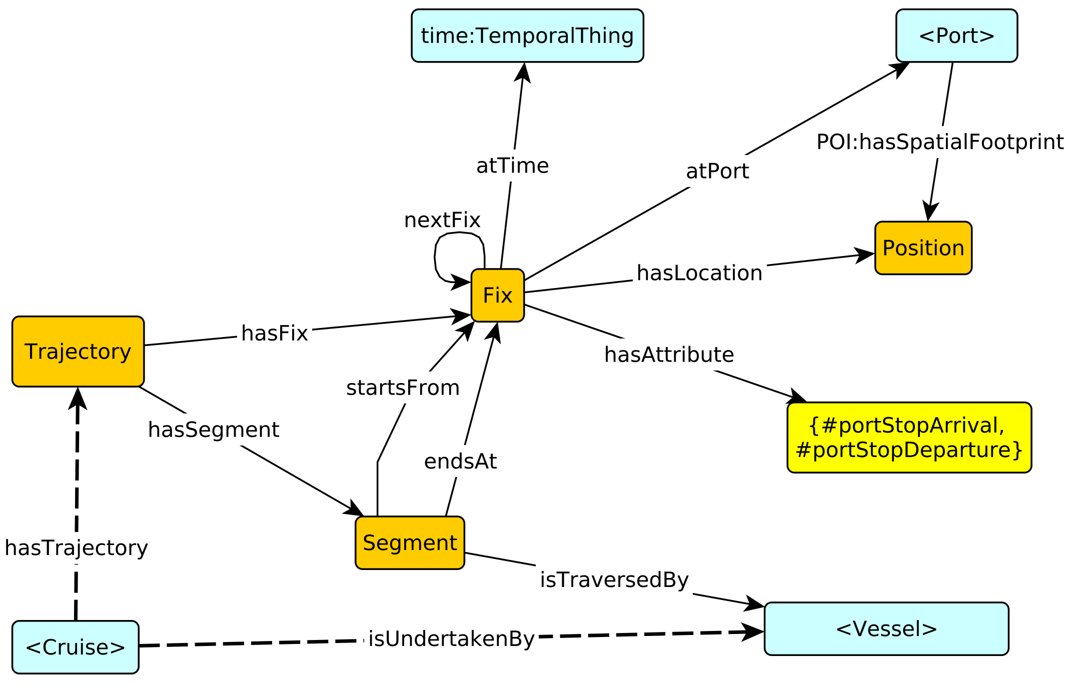

Figure 14.2: ⟨Trajectory⟩ pattern specialised for cruises

In real scenarios, often only properties of the fixes will be known, in particular their locations, and temporal extension, whether they are at ports, whether they are arrival or departure fixes from ports, and possibly in which sequence the fixes occured. The traversing vessel will usually also be known, however the trajectory pattern attaches it to trajectory segments which will usually not be explicitly represented, i.e. for mappings the shortcut participatesIn will be relevant.

Some axiomatization, in particular relating the attributes to the atPort property will be added, as well as several suitable views.

## 14.2 Axiomatization

**Axioms included from other patterns:** (7.1), (7.2), (7.3), (7.4), (7.5), (7.6), (7.7), (7.8), (7.9), (7.10), (7.11), (7.12) — since CruiseInfoObj is a subclass of InformationObject; axioms for trajectory from [Hu et al., 2013]; (2.10) — due to the use of providesAgentRoleType property; (10.1) — due to the use of isFundedBy property; (11.1) due to the use of isAssociatedWithProgram property.

Guarded domain and/or range restrictions for isDescribedBy, hasCruiseID, hasCruiseType, hasCruiseScheduler, hasCruiseOperator, isUndertakenBy, hasTrajectory.

$$\mathsf{isDescribedBy}(x,y) \land \mathsf{CruiseInfoObj}(y) \rightarrow \mathsf{Cruise}(x) \tag{14.1}$$
$$\mathsf{isDescribedBy}(x,y) \land \mathsf{Cruise}(x) \rightarrow \mathsf{CruiseInfoObj}(y) \tag{14.2}$$
$$\mathsf{hasCruiseID}(x,y) \land \mathsf{xsd{:}string}(y) \rightarrow \mathsf{CruiseInfoObj}(x) \tag{14.3}$$
$$\mathsf{hasCruiseID}(x,y) \rightarrow \mathsf{xsd{:}string}(y) \tag{14.4}$$
$$\mathsf{hasCruiseType}(x,y) \land \mathsf{CruiseType}(y) \rightarrow \mathsf{CruiseInfoObj}(x) \tag{14.5}$$
$$\mathsf{hasCruiseType}(x,y) \land \mathsf{CruiseInfoObj}(x) \rightarrow \mathsf{CruiseType}(y) \tag{14.6}$$
$$\mathsf{hasCruiseScheduler}(x,y) \land \mathsf{Agent}(y) \rightarrow \mathsf{Cruise}(x) \tag{14.7}$$
$$\mathsf{hasCruiseScheduler}(x,y) \land \mathsf{Cruise}(x) \rightarrow \mathsf{Agent}(y) \tag{14.8}$$
$$\mathsf{hasCruiseOperator}(x,y) \land \mathsf{Agent}(y) \rightarrow \mathsf{Cruise}(x) \tag{14.9}$$
$$\mathsf{hasCruiseOperator}(x,y) \land \mathsf{Cruise}(x) \rightarrow \mathsf{Agent}(y) \tag{14.10}$$
$$\mathsf{isUndertakenBy}(x,y) \land \mathsf{Vessel}(y) \rightarrow \mathsf{Cruise}(x) \tag{14.11}$$
$$\mathsf{isUndertakenBy}(x,y) \land \mathsf{Cruise}(x) \rightarrow \mathsf{Vessel}(y) \tag{14.12}$$
$$\mathsf{hasTrajectory}(x,y) \land \mathsf{Trajectory}(y) \rightarrow \mathsf{Cruise}(x) \tag{14.13}$$
$$\mathsf{hasTrajectory}(x,y) \land \mathsf{Cruise}(x) \rightarrow \mathsf{Trajectory}(y) \tag{14.14}$$

A Cruise is an event. It exactly has one trajectory, undertaken by exactly one Vessel and is described by exactly one CruiseInfoObj.

$$\mathsf{Cruise} \sqsubseteq \mathsf{Event} \tag{14.15}$$
$$\mathsf{Cruise} \sqsubseteq (=1\ \mathsf{isDescribedBy.CruiseInfoObj}) \tag{14.16}$$
$$\mathsf{Cruise} \sqsubseteq (=1\ \mathsf{isUndertakenBy.Vessel}) \tag{14.17}$$
$$\mathsf{Cruise} \sqsubseteq (=1\ \mathsf{hasTrajectory.Trajectory}) \tag{14.18}$$

A CruiseInfoObj is an InformationObject with additional properties indicating: a unique cruise ID and a cruise type. A cruise type is either #`transit`, #`operational`, #`maintenance`, or #`other`.

$$\mathsf{CruiseInfoObj} \sqsubseteq \mathsf{InformationObject} \tag{14.19}$$
$$\mathsf{CruiseInfoObj} \sqsubseteq (=1\ \mathsf{hasCruiseID.xsd{:}string}) \sqcap (=1\ \mathsf{hasCruiseType.CruiseType}) \tag{14.20}$$
$$\top \sqsubseteq (\leqslant 1 \mathsf{hasCruiseID}^{-}.\top) \tag{14.21}$$
$$\mathsf{all\text{-}different}(\#\mathtt{transit}, \#\mathtt{operational}, \#\mathtt{maintenance}, \#\mathtt{other}) \tag{14.22}$$
$$\mathsf{CruiseType} \equiv \{\#\mathtt{transit}, \#\mathtt{operational}, \#\mathtt{maintenance}, \#\mathtt{other}\} \tag{14.23}$$

The cruise type is #`operational` iff the cruise provides a #`chief_scientist` role and is funded by some FundingAward.

$$\begin{aligned}&\mathsf{Cruise} \sqcap \exists \mathsf{providesAgentRoleType}.\{\#\mathtt{chief\_scientist}\} \sqcap \exists \mathsf{isDescribedBy}.\exists \mathsf{isFundedBy.FundingAward}\\ &\equiv \mathsf{Cruise} \sqcap \exists \mathsf{isDescribedBy}.\exists \mathsf{hasCruiseType}.\{\#\mathtt{operational}\}\end{aligned} \tag{14.24}$$

**Remarks:** Note that hasCruiseID is seen as an OWL data property (as the range is xsd:string which is a datatype). However, OWL 2 DL does not allow one to express an inverse functional data property. This makes axiom (14.21) not enforceable in the OWL translation of this pattern's axiomatization.

We now details the assertions for cruise trajectories. This mostly follows axiomatization in [Hu et al., 2013] with suitable adjustments.

### 14.3 Views

Views for AgentRole?

Cruise start date, end date, start port, end port, port sequence.

## 15 ⟨Deployment⟩ Pattern

*We discussed that we do not need this yet, as we look only at cruises as deployments for now.*

# 16 ⟨Vessel⟩ pattern

## 16.1 Description

The ⟨Vessel⟩ pattern describes vessels on which cruises are carried out. First, each Vessel has exactly one VesselInfoObject which is a kind of InformationObject that stores data about identifiers of the vessel as well as about official/canonical name, a web page, etc. Each VesselID stores an identifier of a particular vessel. Such a VesselID has exactly one VesselIDScheme, one string value, and one Agent as the vessel ID provider. For any particular vessel ID scheme, a vessel can have at most one vessel ID. Also, for each vessel, there is at least one vessel ID scheme with which the vessel ID is defined (i.e., there cannot be a vessel having no vessel ID whatsoever for any scheme). No two vessels share the same vessel ID with the same scheme. Here, the IMO number is a particular vessel ID scheme, provided by International Maritime Organization (IMO). A Vessel has 0 or more platform types which are terms taken from the SEAVOC L06 vocabulary. Additionally, a Vessel has at least one Agent as its owner. Finally, each Vessel may be described in terms of observable properties modeled as Quantity. These quantities have an associated QuantityKind corresponding to length, beam, draft, and displacement. Each quantity also has a unit and a numerical value. These parts will be aligned to the well established *QUDT – Quantities, Units, Dimensions and Data Types Ontologies*. This improves interoperability and also allows us to add additional information, such as the uncertainty of the numerical value, in the future.

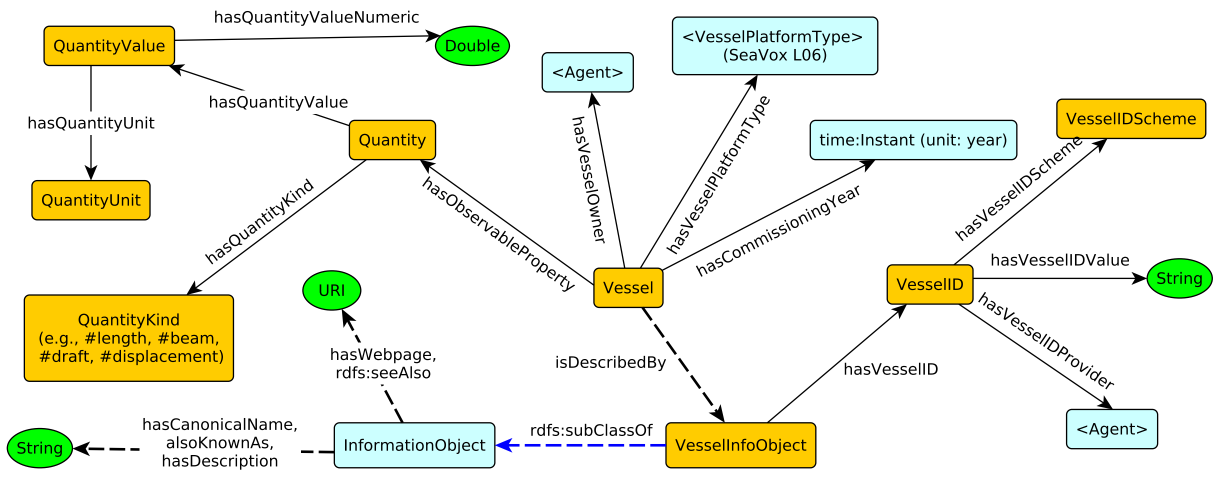

Figure 16.1: ⟨Vessel⟩ pattern

## 16.2 Axiomatization

**Axioms included from other patterns:** (7.1), (7.2), (7.3), (7.4), (7.5), (7.6), (7.7), (7.8), (7.9), (7.10), (7.11), (7.12) — since VesselInfoObject is a subclass of InformationObject;

We assert the guarded domain and/or range restrictions for isDescribedBy, hasVesselID, hasVesselIDProvider, hasVesselIDScheme, hasVesselIDValue, hasCommissioningYear, hasVesselPlatformType, hasVesselOwner, hasObservableProperty, hasQuantityValue, hasQuantityValueNumeric, hasQuantityUnit properties.

$$\mathsf{isDescribedBy}(x,y) \land \mathsf{VesselInfoObject}(y) \rightarrow \mathsf{Vessel}(x) \tag{16.1}$$
$$\mathsf{isDescribedBy}(x,y) \land \mathsf{Vessel}(x) \rightarrow \mathsf{VesselInfoObject}(y) \tag{16.2}$$
$$\mathsf{hasVesselID}(x,y) \land \mathsf{VesselID}(y) \rightarrow \mathsf{VesselInfoObject}(x) \tag{16.3}$$
$$\mathsf{hasVesselID}(x,y) \land \mathsf{VesselInfoObject}(x) \rightarrow \mathsf{VesselID}(y) \tag{16.4}$$
$$\mathsf{hasVesselIDScheme}(x,y) \land \mathsf{VesselIDScheme}(y) \rightarrow \mathsf{VesselID}(x) \tag{16.5}$$
$$\mathsf{hasVesselIDScheme}(x,y) \land \mathsf{VesselID}(x) \rightarrow \mathsf{VesselIDScheme}(y) \tag{16.6}$$
$$\mathsf{hasVesselIDValue}(x,y) \land \mathsf{xsd{:}string}(y) \rightarrow \mathsf{VesselID}(x) \tag{16.7}$$
$$\mathsf{hasVesselIDValue}(x,y) \rightarrow \mathsf{xsd{:}string}(y) \tag{16.8}$$
$$\mathsf{hasVesselIDProvider}(x,y) \land \mathsf{Agent}(y) \rightarrow \mathsf{VesselID}(x) \tag{16.9}$$
$$\mathsf{hasVesselIDProvider}(x,y) \land \mathsf{VesselID}(x) \rightarrow \mathsf{Agent}(y) \tag{16.10}$$
$$\mathsf{hasCommissioningYear}(x,y) \land \mathsf{time{:}Instant}(y) \rightarrow \mathsf{Vessel}(x) \tag{16.11}$$
$$\mathsf{hasCommissioningYear}(x,y) \land \mathsf{Vessel}(x) \rightarrow \mathsf{time{:}Instant}(y) \tag{16.12}$$
$$\mathsf{hasVesselPlatformType}(x,y) \land \mathsf{VesselPlatformType}(y) \rightarrow \mathsf{Vessel}(x) \tag{16.13}$$
$$\mathsf{hasVesselPlatformType}(x,y) \land \mathsf{Vessel}(x) \rightarrow \mathsf{VesselPlatformType}(y) \tag{16.14}$$
$$\mathsf{hasVesselOwner}(x,y) \land \mathsf{Agent}(y) \rightarrow \mathsf{Vessel}(x) \tag{16.15}$$
$$\mathsf{hasVesselOwner}(x,y) \land \mathsf{Vessel}(x) \rightarrow \mathsf{Agent}(y) \tag{16.16}$$
$$\mathsf{hasObservableProperty}(x,y) \land \mathsf{Quantity}(y) \rightarrow \mathsf{Vessel}(x) \tag{16.17}$$
$$\mathsf{hasObservableProperty}(x,y) \land \mathsf{Vessel}(x) \rightarrow \mathsf{Quantity}(y) \tag{16.18}$$
$$\mathsf{hasQuantityKind}(x,y) \land \mathsf{QuantityKind}(x) \rightarrow \mathsf{Quantity}(x) \tag{16.19}$$
$$\mathsf{hasQuantityKind}(x,y) \land \mathsf{Quantity}(x) \rightarrow \mathsf{QuantityKind}(y) \tag{16.20}$$
$$\mathsf{hasQuantityValue}(x,y) \land \mathsf{QuantityValue}(x) \rightarrow \mathsf{Quantity}(x) \tag{16.21}$$
$$\mathsf{hasQuantityValue}(x,y) \land \mathsf{Quantity}(x) \rightarrow \mathsf{QuantityValue}(y) \tag{16.22}$$
$$\mathsf{hasQuantityUnit}(x,y) \land \mathsf{QuantityUnit}(x) \rightarrow \mathsf{QuantityValue}(x) \tag{16.23}$$
$$\mathsf{hasQuantityUnit}(x,y) \land \mathsf{QuantityValue}(x) \rightarrow \mathsf{QuantityUnit}(y) \tag{16.24}$$
$$\mathsf{hasQuantityValueNumeric}(x,y) \land \mathsf{xsd{:}double}(y) \rightarrow \mathsf{QuantityValue}(x) \tag{16.25}$$
$$\mathsf{hasQuantityValueNumeric}(x,y) \rightarrow \mathsf{xsd{:}double}(y) \tag{16.26}$$

Each vessel has exactly one vessel information object (the functionality is enforced by axiom (7.2)), at least one agent as its owner, and exactly one commissioning year.

$$\mathsf{Vessel} \sqsubseteq \exists \mathsf{isDescribedBy}.\mathsf{VesselInfoObject} \tag{16.27}$$
$$\mathsf{Vessel} \sqsubseteq \exists \mathsf{hasVesselOwner}.\mathsf{Agent} \tag{16.28}$$
$$\mathsf{Vessel} \sqsubseteq (=1\ \mathsf{hasCommissioningYear}.\mathsf{time{:}Instant}) \tag{16.29}$$

A vessel information object is an information object and has at least one vessel ID. Every vessel ID has exactly one vessel ID scheme and one vessel ID value; and may have at most one vessel ID provider.

$$\mathsf{VesselInfoObject} \sqsubseteq \mathsf{InformationObject} \sqcap \exists \mathsf{hasVesselID}.\mathsf{VesselID} \tag{16.30}$$
$$\mathsf{VesselID} \sqsubseteq (=1\ \mathsf{hasVesselIDScheme}.\mathsf{VesselIDScheme}) \tag{16.31}$$
$$\mathsf{VesselID} \sqsubseteq (=1\ \mathsf{hasVesselIDValue}.\mathsf{xsd{:}string}) \tag{16.32}$$
$$\mathsf{VesselID} \sqsubseteq (\leqslant 1\ \mathsf{hasVesselIDProvider}.\mathsf{Agent}) \tag{16.33}$$

Further, for each vessel ID scheme, the vessel information object may only have at most one vessel ID[1]. An

[1]Rule (16.34) is not yet written into OWL file; OWL translation is needed

example of vessel ID scheme is IMO number (provided by International Maritime Organization).

$$\mathsf{VesselInfoObject}(w) \land \mathsf{VesselIDScheme}(x) \land \mathsf{hasVesselID}(w, y_1) \land \mathsf{hasVesselID}(w, y_2) \land \mathsf{hasVesselIDScheme}(y_1, x) \land \mathsf{hasVesselIDScheme}(y_2, x) \rightarrow y_1 = y_2 \tag{16.34}$$

Each vessel may have information on its length, beam, draft, and displacement.

$$\mathsf{Quantity} \sqsubseteq (=1\ \mathsf{hasQuantityKind.QuantityKind}) \sqcap (=1\ \mathsf{hasQuantityValue.QuantityValue}) \tag{16.35}$$

$$\{\#\texttt{length},\#\texttt{beam}, \#\texttt{draft}, \#\texttt{displacement}\} \sqsubseteq \mathsf{QuantityKind} \tag{16.36}$$

$$\mathsf{alldifferent}(\#\texttt{length}, \#\texttt{beam}, \#\texttt{draft}, \#\texttt{displacement}) \tag{16.37}$$

$$\mathsf{QuantityValue} \sqsubseteq (=1\ \mathsf{hasQuantityUnit.QuantityUnit}) \sqcap (=1\ \mathsf{hasQuantityValueNumeric.xsd{:}double}) \tag{16.38}$$

$$\mathsf{Vessel} \sqsubseteq (=1\ \mathsf{hasObservableProperty}.\exists \mathsf{hasQuantityKind}.\{\#\texttt{length}\}) \tag{16.39}$$

$$\mathsf{Vessel} \sqsubseteq (=1\ \mathsf{hasObservableProperty}.\exists \mathsf{hasQuantityKind}.\{\#\texttt{beam}\}) \tag{16.40}$$

$$\mathsf{Vessel} \sqsubseteq (=1\ \mathsf{hasObservableProperty}.\exists \mathsf{hasQuantityKind}.\{\#\texttt{draft}\}) \tag{16.41}$$

$$\mathsf{Vessel} \sqsubseteq (=1\ \mathsf{hasObservableProperty}.\exists \mathsf{hasQuantityKind}.\{\#\texttt{displacement}\}) \tag{16.42}$$

A Vessel has 0 or more platform types. Currently, platform types are terms taken from the SEAVOX L06 vocabulary. There are 78 platform types in total, each is represented by an individual name of the form $\texttt{seavoxl06\_type}_s$ where $s$ is the key string. The hasSeaVoXL06Key property relates each of those individuals to the corresponding SeaVoXL06 key string literal while the hasSeaVoXL06Term relates it to the corresponding SeaVoXL06 term string literal.

$$\mathsf{SeaVoXL06} \sqsubseteq \mathsf{VesselPlatformType} \tag{16.43}$$

$$\mathsf{SeaVoXL06} \sqsubseteq (=1\ \mathsf{hasSeaVoXL06Key.xsd{:}string}) \sqcap (=1\ \mathsf{hasSeaVoXL06Term.xsd{:}string}) \tag{16.44}$$

The list of key and term for SeaVoXL06 platform types is given in Table 16.1.

## 16.3 Views

No known view is currently associated with this pattern.

| Key | Term | Key | Term |
|---|---|---|---|
| 0 | unknown | 47 | float |
| 10 | land or seafloor | 48 | mooring |
| 11 | fixed benthic node | 49 | surface ice buoy |
| 12 | sea bed vehicle | 50 | buoyant aircraft |
| 13 | beach/intertidal zone structure | 51 | free-rising balloon |
| 14 | land/onshore structure | 52 | free-floating balloon |
| 15 | land/onshore vehicle | 53 | tethered balloon |
| 16 | offshore structure | 54 | airship |
| 17 | coastal structure | 60 | non-buoyant aircraft |
| 18 | river station | 61 | research aeroplane |
| 19 | mesocosm bag | 62 | aeroplane |
| 20 | submersible | 63 | rocket |
| 21 | propelled manned submersible | 64 | geostationary orbiting satellite |
| 22 | propelled unmanned submersible | 65 | orbiting satellite |
| 23 | towed unmanned submersible | 66 | manned spacecraft |
| 24 | drifting manned submersible | 67 | helicopter |
| 25 | autonomous underwater vehicle | 68 | satellite |
| 26 | lowered unmanned submersible | 69 | autogyro |
| 27 | sub-surface gliders | 6A | glider |
| 30 | ship | 6B | kite |
| 31 | research vessel | 6C | parachute |
| 32 | vessel of opportunity | 6Z | spacecraft |
| 33 | self-propelled small boat | 70 | organism |
| 34 | vessel at fixed position | 71 | human |
| 35 | vessel of opportunity on fixed route | 72 | diver |
| 36 | fishing vessel | 73 | flightless bird |
| 37 | self-propelled boat | 74 | seabird and duck |
| 38 | man-powered boat | 75 | cetacean |
| 39 | naval vessel | 76 | fish |
| 3A | man-powered small boat | 77 | land-sea mammals |
| 3B | autonomous surface water vehicle | 90 | cryosphere |
| 3C | surface gliders | 91 | ice island |
| 3Z | surface vessel | 92 | ice shelf |
| 41 | moored surface buoy | 93 | pack ice |
| 42 | drifting surface float | 94 | drift ice |
| 43 | subsurface mooring | 95 | amphibious vehicle |
| 44 | drifting subsurface float | 96 | amphibious crawler |
| 45 | fixed subsurface vertical profiler | 9A | DUKW |
| 46 | drifting subsurface profiling float | 9B | hovercraft |

Table 16.1: SeaVoXL06 key-term pairs

# 17 ⟨RepositoryObject⟩ Pattern

## 17.1 Description

The ⟨RepositoryObject⟩ pattern (Figure 17.1) provides an abstraction of a repository object which can be a DataFileSet, a DocumentSet, etc., though we do not restrict repository objects only to these three types. Note that the names of subclass of RepositoryObject above seems to indicate that a repository object is a container. In fact, Those aforementioned subclasses do act as a container: DataFileSet contains data files, while DocumentSet contains documents (see Section 18) and 23). This is, however, not enforced in this pattern and thus lessening the ontological commitment.

In essence, a repository object is a kind of information object (through subclassing to InformationObject), hence inherits all properties an information object can have. In addition to the properties defined in InformationObject, a repository object has additional properties and features as specified below.

A repository object has exactly one identifier information which consist of a scheme (e.g., DOI) and a string value. A repository object may have (0 or more) keywords (as strings), science themes (as strings), descriptions (as strings). A repository object may have some date information about when it is created, issued and/or released. A repository object belongs to 0 or more repositories. We do not model a ⟨Repository⟩ pattern and simply assume it to be defined (in the simplest case, it would consist of a repository class and an attached information object). A repository object has exactly one non-empty list of agents as its originators, modeled by reusing AgentRole pattern whereby the repository object provides "originator" as an agent role type. A list is used here to reflect the significance of originator ordering. As of now, this is the only known agent role type that is provided by a repository object. Other agent role types can still be added in the future. A repository object is funded by 0 or more funding awards and associated with 0 or more programs. Furthermore, a repository object originates from 0 or more cruises. A repository object may also be related to (0 or more) other repository objects under some particular relationship type. The relationship types are currently taken from the DataCite vocabulary and they are used as subproperties of hasRepositoryObjectRelationTo. This, however, does not preclude the use of other relationship types if necessary.

## 17.2 Axiomatization

**Axioms included from other patterns:** (2.1), (2.2), (2.3), (2.4), (2.6), (2.8), (2.9), (2.10), (2.11), (2.12), (2.13), (2.14), (2.15), (7.1), (7.2), (7.3), (7.4), (7.5), (7.6), (7.7), (7.8), (7.9), (7.10), (7.11), (7.12), (10.1), (11.1).

Guarded domain and range restrictions. Note that the range restriction of data properties is unguarded.

$$\mathsf{hasAbstract}(x,y) \land \mathsf{xsd{:}string}(y) \to \mathsf{RepositoryObject}(x) \tag{17.1}$$
$$\mathsf{hasAbstract}(x,y) \to \mathsf{xsd{:}string}(y) \tag{17.2}$$
$$\mathsf{hasKeyword}(x,y) \land \mathsf{xsd{:}string}(y) \to \mathsf{RepositoryObject}(x) \tag{17.3}$$
$$\mathsf{hasKeyword}(x,y) \to \mathsf{xsd{:}string}(y) \tag{17.4}$$
$$\mathsf{hasSubject}(x,y) \land \mathsf{xsd{:}string}(y) \to \mathsf{RepositoryObject}(x) \tag{17.5}$$
$$\mathsf{hasSubject}(x,y) \to \mathsf{xsd{:}string}(y) \tag{17.6}$$
$$\mathsf{hasScienceTheme}(x,y) \land \mathsf{xsd{:}string}(y) \to \mathsf{RepositoryObject}(x) \tag{17.7}$$
$$\mathsf{hasScienceTheme}(x,y) \to \mathsf{xsd{:}string}(y) \tag{17.8}$$
$$\mathsf{hasLanguageSpecifier}(x,y) \land \mathsf{xsd{:}string}(y) \to \mathsf{RepositoryObject}(x) \tag{17.9}$$
$$\mathsf{hasLanguageSpecifier}(x,y) \to \mathsf{xsd{:}string}(y) \tag{17.10}$$
$$\mathsf{hasRepositoryObjectTitle}(x,y) \land \mathsf{xsd{:}string}(y) \to \mathsf{RepositoryObject}(x) \tag{17.11}$$
$$\mathsf{hasRepositoryObjectTitle}(x,y) \to \mathsf{xsd{:}string}(y) \tag{17.12}$$

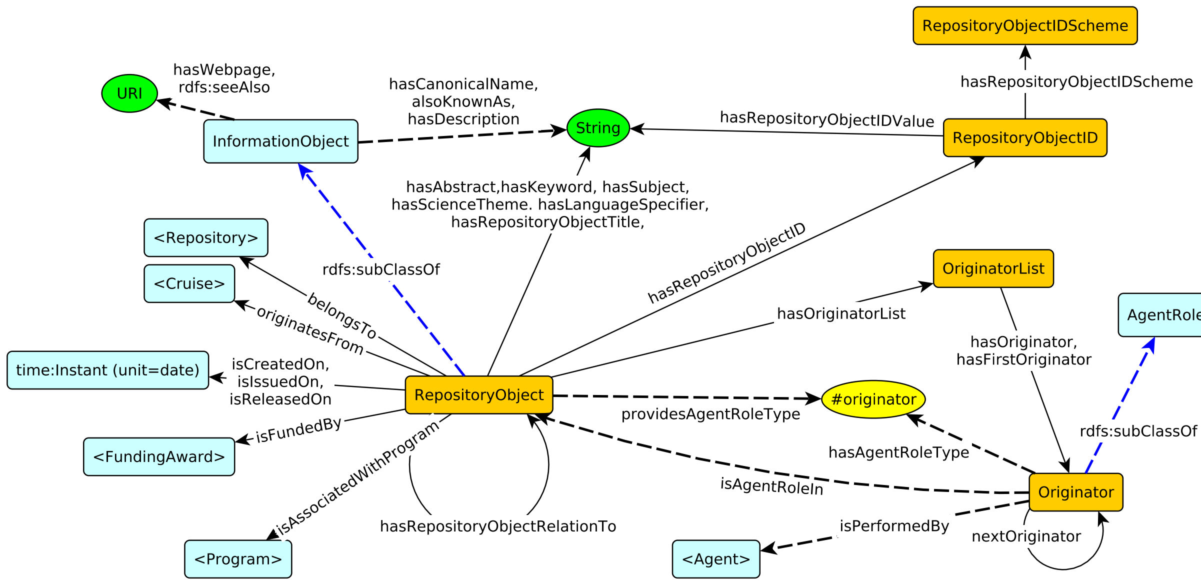

Figure 17.1: ⟨RepositoryObject⟩ pattern

$$\mathsf{hasRepositoryObjectID}(x,y) \wedge \mathsf{RepositoryObjectID}(y) \rightarrow \mathsf{RepositoryObject}(x) \tag{17.13}$$
$$\mathsf{hasRepositoryObjectID}(x,y) \wedge \mathsf{RepositoryObject}(x) \rightarrow \mathsf{RepositoryObjectID}(y) \tag{17.14}$$
$$\mathsf{hasRepositoryObjectIDScheme}(x,y) \wedge \mathsf{RepositoryObjectIDScheme}(y) \rightarrow \mathsf{RepositoryObjectID}(x) \tag{17.15}$$
$$\mathsf{hasRepositoryObjectIDScheme}(x,y) \wedge \mathsf{RepositoryObjectID}(x) \rightarrow \mathsf{RepositoryObjectIDScheme}(y) \tag{17.16}$$
$$\mathsf{hasRepositoryObjectIDValue}(x,y) \wedge \mathsf{xsd{:}string}(y) \rightarrow \mathsf{RepositoryObjectID}(x) \tag{17.17}$$
$$\mathsf{hasRepositoryObjectIDValue}(x,y) \rightarrow \mathsf{xsd{:}string}(y) \tag{17.18}$$
$$\mathsf{hasOriginatorList}(x,y) \wedge \mathsf{OriginatorList}(y) \rightarrow \mathsf{RepositoryObject}(x) \tag{17.19}$$
$$\mathsf{hasOriginatorList}(x,y) \wedge \mathsf{RepositoryObject}(x) \rightarrow \mathsf{OriginatorList}(y) \tag{17.20}$$
$$\mathsf{hasOriginator}(x,y) \wedge \mathsf{Originator}(y) \rightarrow \mathsf{OriginatorList}(x) \tag{17.21}$$
$$\mathsf{hasOriginator}(x,y) \wedge \mathsf{OriginatorList}(x) \rightarrow \mathsf{Originator}(y) \tag{17.22}$$
$$\mathsf{hasFirstOriginator}(x,y) \wedge \mathsf{Originator}(y) \rightarrow \mathsf{OriginatorList}(x) \tag{17.23}$$
$$\mathsf{hasFirstOriginator}(x,y) \wedge \mathsf{OriginatorList}(x) \rightarrow \mathsf{Originator}(y) \tag{17.24}$$
$$\mathsf{nextOriginator}(x,y) \wedge \mathsf{Originator}(y) \rightarrow \mathsf{Originator}(x) \tag{17.25}$$
$$\mathsf{nextOriginator}(x,y) \wedge \mathsf{Originator}(x) \rightarrow \mathsf{Originator}(y) \tag{17.26}$$
$$\mathsf{belongsTo}(x,y) \wedge \mathsf{Repository}(y) \rightarrow \mathsf{RepositoryObject}(x) \tag{17.27}$$
$$\mathsf{belongsTo}(x,y) \wedge \mathsf{RepositoryObject}(x) \rightarrow \mathsf{Repository}(y) \tag{17.28}$$
$$\mathsf{originatesFrom}(x,y) \wedge \mathsf{Cruise}(y) \rightarrow \mathsf{RepositoryObject}(x) \tag{17.29}$$
$$\mathsf{originatesFrom}(x,y) \wedge \mathsf{RepositoryObject}(x) \rightarrow \mathsf{Cruise}(y) \tag{17.30}$$
$$\mathsf{isCreatedOn}(x,y) \wedge \mathsf{RepositoryObject}(x) \rightarrow \mathsf{time{:}Instant}(y) \tag{17.31}$$
$$\mathsf{isIssuedOn}(x,y) \wedge \mathsf{RepositoryObject}(x) \rightarrow \mathsf{time{:}Instant}(y) \tag{17.32}$$
$$\mathsf{isReleasedOn}(x,y) \wedge \mathsf{RepositoryObject}(x) \rightarrow \mathsf{time{:}Instant}(y) \tag{17.33}$$
$$\mathsf{hasRepositoryObjectRelationTo}(x,y) \wedge \mathsf{RepositoryObject}(y) \rightarrow \mathsf{RepositoryObject}(x) \tag{17.34}$$
$$\mathsf{hasRepositoryObjectRelationTo}(x,y) \wedge \mathsf{RepositoryObject}(x) \rightarrow \mathsf{RepositoryObject}(y) \tag{17.35}$$

A repository object is an information object and has exactly one repository object ID which has exactly one ID scheme and one string value. A repository object provides `originator` agent role type and also has exactly one originator list and this list has exactly one first originator and possibly other originator. Each originator itself is an agent role in a repository object and its agent role type is `originator`. Since an originator is an agent role, it is performed by exactly one agent.

$$\mathsf{RepositoryObject} \sqsubseteq \mathsf{InformationObject} \tag{17.36}$$
$$\mathsf{RepositoryObject} \sqsubseteq (=1\ \mathsf{hasRepositoryObjectID}.\mathsf{RepositoryObjectID}) \tag{17.37}$$
$$\mathsf{RepositoryObjectID} \sqsubseteq (=1\mathsf{hasRepositoryObjectIDScheme}.\mathsf{RepositoryObjectIDScheme}) \tag{17.38}$$
$$\mathsf{RepositoryObjectID} \sqsubseteq (=1\mathsf{hasRepositoryObjectIDValue}.\mathsf{xsd{:}string}) \tag{17.39}$$
$$\mathsf{RepositoryObject} \sqsubseteq (=1\mathsf{hasOriginatorList}.\mathsf{OriginatorList}) \tag{17.40}$$
$$\mathsf{RepositoryObject} \sqsubseteq \exists\mathsf{providesAgentRoleType}.\{\#\mathtt{originator}\} \tag{17.41}$$
$$\mathsf{OriginatorList} \sqsubseteq \exists\mathsf{hasOriginator}.\mathsf{Originator} \tag{17.42}$$
$$\mathsf{OriginatorList} \sqsubseteq (=1\ \mathsf{hasFirstOriginator}.\mathsf{Originator}) \tag{17.43}$$
$$\mathsf{hasFirstOriginator} \sqsubseteq \mathsf{hasOriginator} \tag{17.44}$$
$$\mathsf{Originator} \sqsubseteq \mathsf{AgentRole} \tag{17.45}$$
$$\mathsf{Originator} \sqsubseteq \exists\mathsf{hasAgentRoleType}.\{\#\mathtt{originator}\} \tag{17.46}$$
$$\mathsf{Originator} \sqsubseteq \exists\mathsf{isAgentRoleIn}.\mathsf{RepositoryObject} \tag{17.47}$$

Since an agent role is an agent role in exactly one thing, the following asserts that the repository object in which an originator is an agent role is exactly the repository object that has the originator list to which the originator belongs to.

$$\mathsf{RepositoryObject}(x) \land \mathsf{hasOriginatorList}(x,y) \land \mathsf{hasOriginator}(y,z) \to \mathsf{isAgentRoleIn}(z,x) \tag{17.48}$$

The above axiom can be translated using rollification technique into:

$$\mathsf{RepositoryObject} \equiv \exists R_{\mathsf{RepositoryObject}}.\mathsf{Self}$$
$$\mathsf{hasOriginator}^{-} \circ \mathsf{hasOriginatorList}^{-} \circ R_{\mathsf{RepositoryObject}} \sqsubseteq \mathsf{isAgentRoleIn}$$

where $R_{\mathsf{RepositoryObject}}$ is a fresh property name.

Subproperties of hasRepositoryObjectRelationTo. The names follow DataCite scheme.

$$\mathsf{isCitedBy} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.49}$$
$$\mathsf{cites} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.50}$$
$$\mathsf{isSupplementTo} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.51}$$
$$\mathsf{isSupplementedBy} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.52}$$
$$\mathsf{isContinuedBy} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.53}$$
$$\mathsf{continues} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.54}$$
$$\mathsf{hasMetadata} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.55}$$
$$\mathsf{isMetadataFor} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.56}$$
$$\mathsf{isNewVersionOf} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.57}$$
$$\mathsf{isPreviousVersionOf} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.58}$$
$$\mathsf{isPartOf} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.59}$$
$$\mathsf{hasPart} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.60}$$
$$\mathsf{isReferencedBy} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.61}$$
$$\mathsf{references} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.62}$$
$$\mathsf{isDocumentedBy} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.63}$$
$$\mathsf{documents} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.64}$$
$$\mathsf{isCompiledBy} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.65}$$

$$\mathsf{compiles} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.66}$$
$$\mathsf{isVariantFormOf} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.67}$$
$$\mathsf{isOriginalFormOf} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.68}$$
$$\mathsf{isIdenticalTo} \sqsubseteq \mathsf{hasRepositoryObjectRelationTo} \tag{17.69}$$

## 17.3 Views

The $n$-th author:

# 18 ⟨DataFileSet⟩ Pattern

## 18.1 Description

DataFileSet is a subclass of RepositoryObject (hence also of InformationObject). In addition to the properties inherited from repository object class, a datafile set acts as a container of one or more datafiles (we assume that it is non-empty). A datafile set has 0 or more spatial extents and temporal extents. It is created by at most one process and at most by one instrument. Additionally, it provides observation for something and is downloadable (as a set) at some URI. Note that relation with other datafile sets can be modeled through repository object relation.

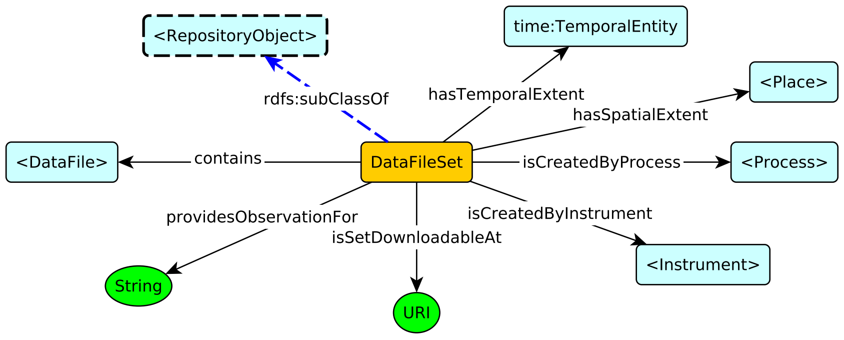

Figure 18.1: ⟨DataFileSet⟩ pattern

## 18.2 Axiomatization

**Axioms included from other patterns:** (2.1), (2.2), (2.3), (2.4), (2.6), (2.8), (2.9), (2.10), (2.11), (2.12), (2.13), (2.14), (2.15), (7.1), (7.2), (7.3), (7.4), (7.5), (7.6), (7.7), (7.8), (7.9), (7.10), (7.11), (7.12), (10.1), (11.1), (17.1), (17.2), (17.3), (17.4), (17.5), (17.6), (17.7), (17.8), (17.9), (17.10), (17.11), (17.12), (17.13), (17.14), (17.15), (17.16), (17.17), (17.18), (17.19), (17.20), (17.21), (17.22), (17.23), (17.24), (17.25), (17.26), (17.27), (17.28), (17.29), (17.30), (17.31), (17.32), (17.33), (17.34), (17.35), (17.36), (17.37), (17.38), (17.39), (17.40), (17.41), (17.42), (17.43), (17.44), (17.45), (17.46), (17.47), (17.48), (17.49), (17.50), (17.51), (17.52), (17.53), (17.54), (17.55), (17.56), (17.57), (17.58), (17.59), (17.60), (17.61), (17.62), (17.63), (17.64), (17.65), (17.66), (17.67), (17.68), (17.69).
Guarded domain and range restriction

$$\mathsf{hasTemporalExtent}(x,y) \land \mathsf{time{:}TemporalEntity}(y) \to \mathsf{DataFileSet}(x) \tag{18.1}$$

$$\mathsf{hasTemporalExtent}(x,y) \land \mathsf{DataFileSet}(x) \to \mathsf{time{:}TemporalEntity}(y) \tag{18.2}$$

$$\mathsf{hasSpatialExtent}(x,y) \land \mathsf{Place}(y) \to \mathsf{DataFileSet}(x) \tag{18.3}$$

$$\mathsf{hasSpatialExtent}(x,y) \land \mathsf{DataFileSet}(x) \to \mathsf{Place}(y) \tag{18.4}$$

$$\mathsf{isCreatedByProcess}(x,y) \land \mathsf{DataFileSet}(x) \to \mathsf{Process}(x) \tag{18.5}$$

$$\mathsf{isCreatedByInstrument}(x,y) \land \mathsf{DataFileSet}(y) \to \mathsf{Instrument}(y) \tag{18.6}$$

$$\mathsf{isSetDownloadableAt}(x,y) \land \mathsf{DataFileSet}(x) \to \mathsf{xsd{:}anyURI}(y) \tag{18.7}$$

$$\mathsf{providesObservationFor}(x,y) \land \mathsf{DataFileSet}(x) \to \mathsf{xsd{:}string}(y) \tag{18.8}$$

$$\mathsf{contains}(x,y) \land \mathsf{DataFile}(y) \to \mathsf{DataFileSet}(x) \tag{18.9}$$

$$\mathsf{contains}(x,y) \land \mathsf{DataFileSet}(x) \rightarrow \mathsf{DataFile}(x) \tag{18.10}$$

A datafile set is a repository object. It contains at least one datafile.

$$\mathsf{DataFileSet} \sqsubseteq \mathsf{RepositoryObject} \sqcap \exists \mathsf{contains}.\mathsf{DataFile} \tag{18.11}$$

# 19 ⟨File⟩ Pattern Stub

## 19.1 Description

⟨File⟩ is a pattern stub that describes essentially a physical file. However, this is not intended to provide a *complete* description of a physical file, but rather to focus only on particular attributes that are deemed important for the application. File is described by an information object and a number of file properties/attributes. Each of these attribute is of certain kind (e.g., file format, md5checksum, file size), has exactly a value (numeric or string), and has exactly a unit (e.g., bytes). These file attributes can be aligned to QUDT Ontologies.

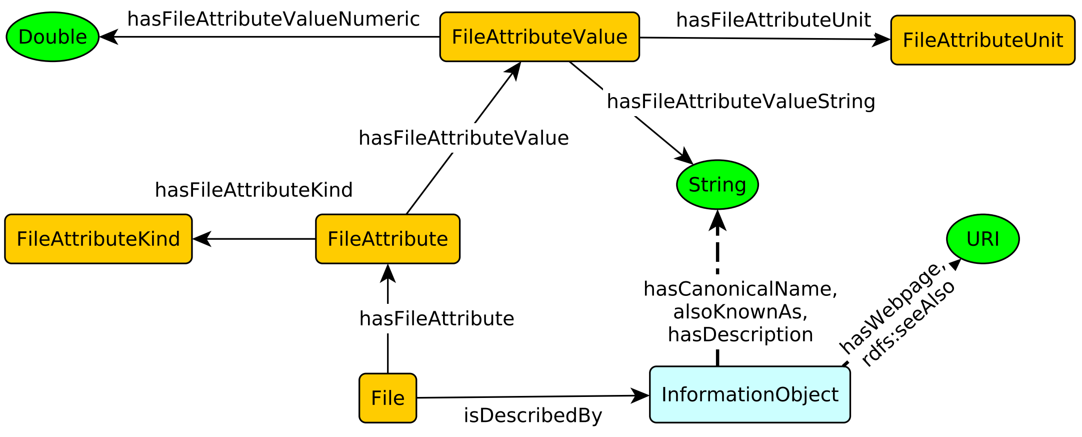

Figure 19.1: ⟨File⟩ pattern

## 19.2 Axiomatization

**Axioms imported from other patterns:** (7.1), (7.2), (7.3), (7.4), (7.5), (7.6), (7.7), (7.8), (7.9), (7.10), (7.11), (7.12)
Guarded domain and range restrictions.

$$\mathsf{hasFileAttribute}(x,y) \land \mathsf{FileAttribute}(y) \to \mathsf{File}(x) \tag{19.1}$$
$$\mathsf{hasFileAttribute}(x,y) \land \mathsf{File}(x) \to \mathsf{FileAttribute}(y) \tag{19.2}$$
$$\mathsf{hasFileAttributeKind}(x,y) \land \mathsf{FileAttributeKind}(y) \to \mathsf{FileAttribute}(x) \tag{19.3}$$
$$\mathsf{hasFileAttributeKind}(x,y) \land \mathsf{FileAttribute}(x) \to \mathsf{FileAttributeKind}(y) \tag{19.4}$$
$$\mathsf{hasFileAttributeValue}(x,y) \land \mathsf{FileAttributeValue}(y) \to \mathsf{FileAttribute}(x) \tag{19.5}$$
$$\mathsf{hasFileAttributeValue}(x,y) \land \mathsf{FileAttribute}(x) \to \mathsf{FileAttributeValue}(y) \tag{19.6}$$
$$\mathsf{hasFileAttributeUnit}(x,y) \land \mathsf{FileAttributeUnit}(y) \to \mathsf{FileAttributeValue}(x) \tag{19.7}$$
$$\mathsf{hasFileAttributeUnit}(x,y) \land \mathsf{FileAttributeValue}(x) \to \mathsf{FileAttributeUnit}(y) \tag{19.8}$$
$$\mathsf{hasFileAttributeValueString}(x,y) \land \mathsf{xsd{:}string}(y) \to \mathsf{FileAttributeValue}(x) \tag{19.9}$$
$$\mathsf{hasFileAttributeValueString}(x,y) \land \mathsf{FileAttributeValue}(x) \to \mathsf{xsd{:}string}(y) \tag{19.10}$$
$$\mathsf{hasFileAttributeValueNumeric}(x,y) \land \mathsf{xsd{:}double}(y) \to \mathsf{FileAttributeValue}(x) \tag{19.11}$$
$$\mathsf{hasFileAttributeValueNumeric}(x,y) \land \mathsf{FileAttributeValue}(x) \to \mathsf{xsd{:}double}(y) \tag{19.12}$$

A File is described by (exactly one) information object. A file may have some file attributes. Each file attribute has exactly one file attribute kind and one file attribute value. A file attribute value then has either numeric value or a numeric string and may have a unit.

$$\mathsf{File} \sqsubseteq \exists \mathsf{isDescribedBy}.\mathsf{InformationObject} \tag{19.13}$$

$$\mathsf{FileAttribute} \sqsubseteq (=1\ \mathsf{hasFileAttributeKind}.\mathsf{FileAttributeKind}) \tag{19.14}$$

$$\mathsf{FileAttribute} \sqsubseteq (=1\ \mathsf{hasFileAttributeValue}.\mathsf{FileAttributeValue}) \tag{19.15}$$

$$\begin{aligned}\mathsf{FileAttributeValue} \sqsubseteq\ & (=1\ \mathsf{hasFileAttributeValueNumeric}.\mathsf{xsd{:}double}) \\ & \sqcup (=1\ \mathsf{hasFileAttributeValueString}.\mathsf{xsd{:}string})\end{aligned} \tag{19.16}$$

$$\mathsf{FileAttributeValue} \sqsubseteq (\leqslant 1\ \mathsf{hasFileAttributeUnit}.\mathsf{FileAttributeUnit}) \tag{19.17}$$

# 20 ⟨DataFile⟩ Pattern

A DataFile is simply a kind of File that is contained by DataFileSet. For DataFile, we are interested in the following file attributes: file format, file size, and checksum. This pattern then reuses ⟨File⟩ pattern with the aforementioned file attributes as instances of FileAttributeKind

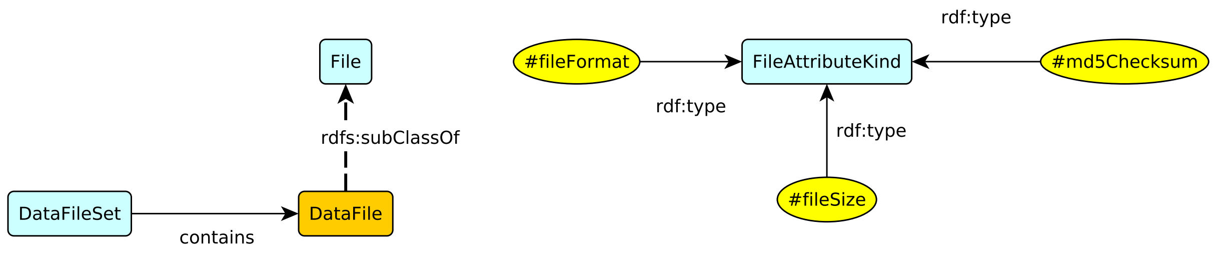

Figure 20.1: ⟨DataFile⟩ pattern

## 20.1 Axiomatization

**Axioms included from other patterns:** (7.1), (7.2), (7.3), (7.4), (7.5), (7.6), (7.7), (7.8), (7.9), (7.10), (7.11), (7.12), (19.1), (19.2), (19.3), (19.4), (19.5), (19.6), (19.7), (19.8), (19.9), (19.10), (19.11), (19.12), (19.13), (19.14), (19.15), (19.16), (19.17).

$$\mathsf{DataFile} \sqsubseteq \mathsf{File} \tag{20.1}$$

$$\mathsf{DataFile} \sqsubseteq \exists \mathsf{hasFileAttribute}.\exists \mathsf{hasFileAttributeKind}.\{\#\texttt{file\_size}\} \tag{20.2}$$

$$\mathsf{DataFile} \sqsubseteq \exists \mathsf{hasFileAttribute}.\exists \mathsf{hasFileAttributeKind}.\{\#\texttt{file\_format}\} \tag{20.3}$$

$$\begin{aligned}
\mathsf{FileAttributeKind}(\#\texttt{file\_size}), \mathsf{FileAttributeKind}(\#\texttt{file\_format}),\\
\mathsf{FileAttributeKind}(\#\texttt{adler32\_checksum}), \mathsf{FileAttributeKind}(\#\texttt{crc32\_checksum}),\\
\mathsf{FileAttributeKind}(\#\texttt{md2\_checksum}), \mathsf{FileAttributeKind}(\#\texttt{md4\_checksum}),\\
\mathsf{FileAttributeKind}(\#\texttt{md5\_checksum}), \mathsf{FileAttributeKind}(\#\texttt{sha1\_checksum}),\\
\mathsf{FileAttributeKind}(\#\texttt{sha2\_224\_checksum}), \mathsf{FileAttributeKind}(\#\texttt{sha2\_256\_checksum}),\\
\mathsf{FileAttributeKind}(\#\texttt{sha2\_384\_checksum}), \mathsf{FileAttributeKind}(\#\texttt{sha2\_512\_checksum}),\\
\mathsf{FileAttributeKind}(\#\texttt{sha2\_512\_224\_checksum}), \mathsf{FileAttributeKind}(\#\texttt{sha2\_512\_256\_checksum}),\\
\mathsf{FileAttributeKind}(\#\texttt{sha3\_224\_checksum}), \mathsf{FileAttributeKind}(\#\texttt{sha3\_256\_checksum}),\\
\mathsf{FileAttributeKind}(\#\texttt{sha3\_384\_checksum}), \mathsf{FileAttributeKind}(\#\texttt{sha3\_512\_checksum}),\\
\mathsf{FileAttributeKind}(\#\texttt{haval256\_5\_checksum}), \mathsf{FileAttributeKind}(\#\texttt{haval256\_4\_checksum}),\\
\mathsf{FileAttributeKind}(\#\texttt{haval256\_3\_checksum}), \mathsf{FileAttributeKind}(\#\texttt{ripemd128\_checksum}),\\
\mathsf{FileAttributeKind}(\#\texttt{ripemd160\_checksum}), \mathsf{FileAttributeKind}(\#\texttt{tiger\_checksum}),\\
\mathsf{FileAttributeKind}(\#\texttt{whirlpool0\_checksum}), \mathsf{FileAttributeKind}(\#\texttt{whirlpool1\_checksum})
\end{aligned} \tag{20.4}$$

## 21 ⟨Process⟩ Pattern Stub

*Modeling for this stub is omitted for now. For the moment, a process pattern only consists of a class Process and an attached information object.*

## 22 Instrument Pattern Stub

*No modeling is done here yet. For now, we simply assume an Instrument class and an attached information object.*

# 23 ⟨DocumentSet⟩ pattern

## 23.1 Description

A document set is a kind of repository object that contains at least one document. The document set is published by some agent and has a certain document type. The document types can be taken from MIME types or other types.

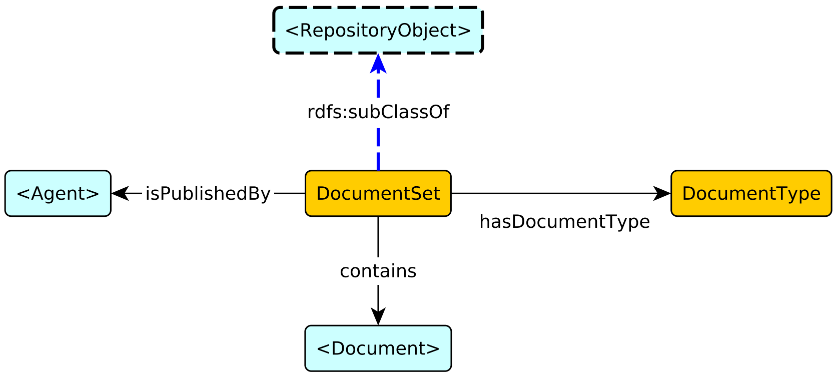

Figure 23.1: ⟨DocumentSet⟩ pattern

## 23.2 Axiomatization

**Axioms included from other patterns:** (2.1), (2.2), (2.3), (2.4), (2.6), (2.8), (2.9), (2.10), (2.11), (2.12), (2.13), (2.14), (2.15), (7.1), (7.2), (7.3), (7.4), (7.5), (7.6), (7.7), (7.8), (7.9), (7.10), (7.11), (7.12), (10.1), (11.1), (17.1), (17.2), (17.3), (17.4), (17.5), (17.6), (17.7), (17.8), (17.9), (17.10), (17.11), (17.12), (17.13), (17.14), (17.15), (17.16), (17.17), (17.18), (17.19), (17.20), (17.21), (17.22), (17.23), (17.24), (17.25), (17.26), (17.27), (17.28), (17.29), (17.30), (17.31), (17.32), (17.33), (17.34), (17.35), (17.36), (17.37), (17.38), (17.39), (17.40), (17.41), (17.42), (17.43), (17.44), (17.45), (17.46), (17.47), (17.48), (17.49), (17.50), (17.51), (17.52), (17.53), (17.54), (17.55), (17.56), (17.57), (17.58), (17.59), (17.60), (17.61), (17.62), (17.63), (17.64), (17.65), (17.66), (17.67), (17.68), (17.69).
Guarded domain and range restriction.

$$\mathsf{isPublishedBy}(x,y) \land \mathsf{Agent}(y) \to \mathsf{DocumentSet}(x) \tag{23.1}$$
$$\mathsf{isPublishedBy}(x,y) \land \mathsf{DocumentSet}(x) \to \mathsf{Agent}(y) \tag{23.2}$$
$$\mathsf{contains}(x,y) \land \mathsf{Document}(y) \to \mathsf{DocumentSet}(x) \tag{23.3}$$
$$\mathsf{contains}(x,y) \land \mathsf{DocumentSet}(x) \to \mathsf{Document}(y) \tag{23.4}$$
$$\mathsf{hasDocumentType}(x,y) \land \mathsf{DocumentType}(y) \to \mathsf{DocumentSet}(x) \tag{23.5}$$
$$\mathsf{hasDocumentType}(x,y) \land \mathsf{DocumentSet}(x) \to \mathsf{DocumentType}(y) \tag{23.6}$$

A document set is a repository object that contains at least one document, has exactly one document type, and is published by some agent.

$$\mathsf{DocumentSet} \sqsubseteq \mathsf{RepositoryObject} \tag{23.7}$$
$$\mathsf{DocumentSet} \sqsubseteq \exists \mathsf{contains}.\mathsf{Document} \tag{23.8}$$
$$\mathsf{DocumentSet} \sqsubseteq (=1\ \mathsf{hasDocumentType}.\mathsf{DocumentType}) \tag{23.9}$$
$$\mathsf{DocumentSet} \sqsubseteq \exists \mathsf{isPublishedBy}.\mathsf{Agent} \tag{23.10}$$

# 24 Document pattern

## 24.1 Description

A Document represents an actual document. It is modeled as an element of document set which is downloadable at some URI and has exactly one document file. A document file itself is a kind of file.

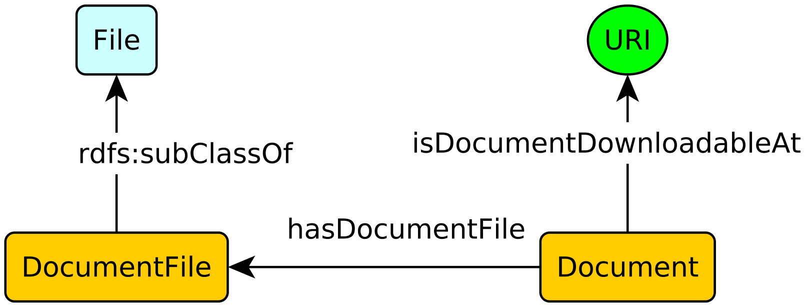

Figure 24.1: ⟨Document⟩ pattern

## 24.2 Axiomatization

**Axioms included from other patterns:** (7.1), (7.2), (7.3), (7.4), (7.5), (7.6), (7.7), (7.8), (7.9), (7.10), (7.11), (7.12), (19.1), (19.2), (19.3), (19.4), (19.5), (19.6), (19.7), (19.8), (19.9), (19.10), (19.11), (19.12), (19.13), (19.14), (19.15), (19.16), (19.17).
Guarded domain and range restrictions.

$$\mathsf{hasDocumentFile}(x,y) \land \mathsf{DocumentFile}(y) \to \mathsf{Document}(x) \tag{24.1}$$

$$\mathsf{hasDocumentFile}(x,y) \land \mathsf{Document}(x) \to \mathsf{DocumentFile}(x) \tag{24.2}$$

$$\mathsf{isDocumentDownloadableAt}(x,y) \land \mathsf{xsd{:}anyURI}(y) \to \mathsf{Document}(x) \tag{24.3}$$

$$\mathsf{isDocumentDownloadableAt}(x,y) \land \mathsf{Document}(x) \to \mathsf{xsd{:}anyURI}(y) \tag{24.4}$$

A document has exactly one document file. A document file is a file.

$$\mathsf{Document} \sqsubseteq (=1\ \mathsf{hasDocumentFile.DocumentFile}) \tag{24.5}$$

$$\mathsf{DocumentFile} \sqsubseteq \mathsf{File} \tag{24.6}$$

## 24.3 Views

# Bibliography

[Hu et al., 2013] Hu, Y., Janowicz, K., Carral, D., Scheider, S., Kuhn, W., Berg-Cross, G., Hitzler, P., Dean, M., and Kolas, D. (2013). A geo-ontology design pattern for semantic trajectories. In Tenbrink, T., Stell, J. G., Galton, A., and Wood, Z., editors, *Spatial Information Theory – 11th International Conference, COSIT 2013, Scarborough, UK, September 2-6, 2013. Proceedings*, volume 8116 of *Lecture Notes in Computer Science*, pages 438–456. Springer, Heidelberg.