# The Enslaved Ontology 1.0: People of the Historic Slave Trade

*Contributors:*
COGAN SHIMIZU — Wright State University
PASCAL HITZLER — Wright State University and Kansas State University
QUINN HIRT — Wright State University
ALICIA SHEILL — Michigan State University
SEILA GONZALEZ — Michigan State University
CATHERINE FOLEY — Michigan State University
DEAN REHBERGER — Michigan State University
ETHAN WATRALL — Michigan State University
WALTER HAWTHORNE — Michigan State University
DUNCAN TARR — Michigan State University
RYAN CARTY — Michigan State University
JEFF MIXTER — OCLC

*Document Date:* April 8, 2019

This work was supported by The Mellon Foundation through the project *Enslaved: People of the Historic Slave Trade*.

# Contents

# List of Figures

# 1 Overview

We are presenting the ontology which drives the data gathering and integration done as part of the project *Enslaved: People of the Historic Slave Trade*,[1] funded by The Andrew W. Mellon Foundation through Michigan State University's Matrix: The Center for digital Humanities & Social Sciencs.

Development of the ontology was a collaborative effort and was carried out using the principles laid out in, e.g., [Krisnadhi et al., 2015a, Krisnadhi and Hitzler, 2016, Krisnadhi et al., 2016]. The modeling team included domain experts, data experts, software developers, and ontology engineers.

The ontology has, in particular, be developed as a *modular* ontology [Hitzler et al., 2017a] based on ontology design patterns [Hitzler et al., 2016]. This means, in a nutshell, that we first identified key terms relating to the data content and expert perspectives on the domain to be modeled, and then developed ontology modules for these terms. The resulting modules, which were informed by corresponding ontology design patterns, are listed and discussed in Chapter 2. The Enslaved Ontology, assembled from these modules, is then persented in Chapter 3.

For background regarding Semantic Web standards, in particular the Web Ontology Language OWL, including its relation to description logics, we refer the reader to [Hitzler et al., 2012, Hitzler et al., 2010].

[1] `http://enslaved.org/`

# 2 Modules

We list the individual modules of the ontology, together with their axioms and explanations thereof. Each axiom is listed only once (for now), i.e. some axioms pertaining to a module may be found in the axiom set listed for an earlier listed module. Schema diagrams are provided throughout, but the reader should keep in mind that while schema diagrams are very useful for understanding an ontology [Karima et al., 2017], they are also inherently ambiguous.

## Primer on Ontology Axioms

Logical axioms are presented (mostly) in description logic notation, which can be directly translated into the Web Ontology Language OWL [Hitzler et al., 2010]. We use description logic notation because it is, in the end, easier for humans to read than any of the other serializations.[1]

Logical axioms serve many purposes in ontology modeling and engineering [Hitzler and Krisnadhi, 2016]; in our context, the primary reason why we choose a strong axiomatization is to disambiguate the ontology.

Almost all axioms which are part of the Enslaved Ontology are of the straightforward and local types. We will now describe these types in more detail, as it will make it much easier to understand the axiomatization of the Enslaved Ontology.

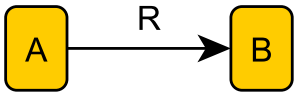


Figure 2.1: Generic node-edge-node schema diagram for explaining systematic axiomatization

There is a systematic way to look at each node-edge-node triple in a schema diagram in order to decide on some of the axioms which should be added: Given a node-edge-node triple with nodes $A$ and $B$ and edge $R$ from $A$ to $B$, as depicted in Figure 2.1, we check all of the following axioms whether they should be included.[2] We list them in natural language, see Figure 2.2 for the formal versions in description logic notation, and Figure 2.3 for the same in Manchester syntax, where we also list our names for these axioms.

1. $A$ is a subClass of $B$.
2. $A$ and $B$ are disjoint.
3. The domain of $R$ is $A$.
4. For every $B$ which has an inverse $R$-filler, this inverse $R$-filler is in $A$. In other words, the domain of $R$, scoped by $B$, is $A$.

[1] Preliminary results supporting this claim can be found in [Shimizu, 2017].

[2] The OWLAx Protégé plug-in [Sarker et al., 2016] provides a convenient interface for adding these axioms.

1. $A \sqsubseteq B$
2. $A \sqcap B \sqsubseteq \bot$
3. $\exists R.\top \sqsubseteq A$
4. $\exists R.B \sqsubseteq A$
5. $\top \sqsubseteq \forall R.B$
6. $A \sqsubseteq \forall R.B$
6. $A \sqsubseteq R.B$
7. $B \sqsubseteq R^{-}.A$
8. $\top \sqsubseteq \leq 1R.\top$
9. $\top \sqsubseteq \leq 1R.B$
10. $A \sqsubseteq \leq 1R.\top$
11. $A \sqsubseteq \leq 1R.B$
11. $\top \sqsubseteq \leq 1R^{-}.\top$
12. $\top \sqsubseteq \leq 1R^{-}.A$
13. $B \sqsubseteq \leq 1R^{-}.\top$
14. $B \sqsubseteq \leq 1R^{-}.A$
15. $A \sqsubseteq \geq 0R.B$

Figure 2.2: Most common axioms which could be produced from a single edge $R$ between nodes $A$ and $B$ in a schema diagram: description logic notation.

5. The range of $R$ is $B$.
6. For every $A$ which has an $R$-filler, this $R$-filler is in $B$. In other words, the range of $R$, scoped by $A$, is $B$.
7. For every $A$ there has to be an $R$-filler in $B$.
8. For every $B$ there has to be an inverse $R$-filler in $A$.
9. $R$ is functional.
10. $R$ has at most one filler in $B$.
11. For every $A$ there is at most one $R$-filler.
12. For every $A$ there is at most one $R$-filler in $B$.
13. $R$ is inverse functional.
14. $R$ has at most one inverse filler in $A$.
15. For every $B$ there is at most one inverse $R$-filler.
16. For every $B$ there is at most one inverse $R$-filler in $A$.
17. An $A$ may have an $R$-filler in $B$.

Domain and range axoims are items 2–5 in this list. Items 6 and 7 are extistential axioms. Items 8–15 are about variants of functionality and inverse functionality. All axiom types except disjointness and those utilizing inverses also apply to datatype properties.

Structural tautologies are, indeed, tautologies, i.e., they do not carry any formal logical content. However as argued in [Hitzler and Krisnadhi, 2016] they can help humans to understand the ontology, by indicating *possible* relationships, i.e., relationships intended by the modeler which, however, cannot be cast into non-tautological axioms.

## Explanations Regarding Schema Diagrams

We utilize schema diagrams to visualize the ontology. In our experience, simple diagrams work best for this purpose. The reader needs to bear in mind, though, that these diagrams are ambiguous and incomplete visualizations of the ontology (or module), as the actual ontology (or module) is constituted by the set of axioms provided.

We use the following visuals in our diagrams:

**rectangular box with solid frame and orange fill:** a class
**rectangual box with dashed frame and blue fill:** a module, which is described in more detail elsewhere in the document
**rectangular box with dashed frame and purple fill:** a set of URIs constituting a controlled vocabulary
**oval with solid frame and yellow fill:** a data type

1. $A$ SubClassOf $B$ (subClass)
2. $A$ DisjointWith $B$ (disjointness)
3. $R$ some `owl:Thing` SubClassOf $A$ (domain)
4. $R$ some $B$ SubClassOf $A$ (scoped domain)
5. `owl:Thing` SubClassOf $R$ only $B$ (range)
6. $A$ SubClassOf $R$ only $B$ (scoped range)
7. $A$ SubClassOf $R$ some $B$ (existential)
8. $B$ SubClassOf inverse $R$ some $A$ (inverse existential)
9. `owl:Thing` SubClassOf $R$ max $1$ `owl:Thing` (functionality)
10. `owl:Thing` SubClassOf $R$ max $1$ $B$ (qualified functionality)
11. $A$ SubClassOf $R$ max $1$ `owl:Thing` (scoped functionality)
12. $A$ SubClassOf $R$ max $1$ $B$ (qualified scoped functionality)
13. `owl:Thing` SubClassOf inverse $R$ max $1$ `owl:Thing` (inverse functionality)
14. `owl:Thing` SubClassOf inverse $R$ max $1$ $A$ (inverse qualified functionality)
15. $B$ SubClassOf inverse $R$ max $1$ `owl:Thing` (inverse scoped functionality)
16. $B$ SubClassOf inverse $R$ max $1$ $A$ (inverse qualified scoped functionality)
17. $A$ SubClassOf $R$ min $0$ $B$ (structural tautology)

Figure 2.3: Most common axioms which could be produced from a single edge $R$ between nodes $A$ and $B$ in a schema diagram: Manchester syntax.

**arrow with white head and no label:** a subClass relationship
**arrow with solid tip and label:** a relationship (or property) other than a subClass relationship

## 2.1 Module Overview

The following are the modules which together constitute the Enslaved Ontology. Each of them will be presented in detail further below, though in different sequence. The Enslaved Ontology focuses on historic persons (or agents), historic events, places and dates, records about events and historic persons, and provenance information. However it also has accomodation for contemporary persons (or agents), such as present-day researchers and research projects contributing to the data.

**Agent** The Enslaved Ontology V1.0 primarily captures information about agents, i.e., persons and organizations, involved in the historic slave trade or in research or reporting about it.

**AgentRecord** Each piece of information about an agent is organized as a record for this agent. I.e. with each agent we typically associate many agent records, the combination of which makes up what we know about this particular agent. Each agent record by itself usually contains one piece of information about the agent, plus provenance information about this piece of information. AgentRecord has a number of sub-modules which we list further below in Section 2.1.1

**Event** Historic events are usually where a record about an agent originates.

**EntityWithProvenance** Our module for recording provenance information about agent records.

**Place** Places associated with events.

**TemporalExtent** For the recording of dates and time.

**ExternalReference** Is used to point to an external reference for an entity, e.g., a particular website about a person or event.

**Description** For textual descriptions of entities as obtained from referenced sources.
**ResearchProject** For information about research projects contributing to the data.

### 2.1.1 Submodules of AgentRecord

The following are the submodules for AgentRecord. Some of these apply only to persons, and not to other types of agents, as indicated in the explanatory text.

**OriginRecord** For recording ethnolinguistic origins and place or origin for a person.
**RaceRecord** For recording race information of a person.
**AgeRecord** For recording numeric age or age cateogry of a person.
**SexRecord** For recording the sex of a person.
**OccupationRecord** For recording information about occupation or skills of a person.
**PersonStatusRecord** For recording a person's freedom status, such as being enslaved or freed.
**NameRecord** For recording names of agents.
**InterAgentRelationshipRecord** For recording relationships between agents, such as family relations or ownership relations.
**ParticipantRoleRecord** For recording participation of agents in events.

## 2.2 Agent and Person

Agents are either persons or organizations. We use an Agent module, with Person as a subclass of Agent. If certain records apply to persons only, and not to agents in general, we declare this using appropriate axioms; see Axiom (7) below.

Agents may have descriptions and external references as described in the respective module descriptions. Central for the Enslaved Ontology are the agent records associated with agents, this module is also described in detail further below.

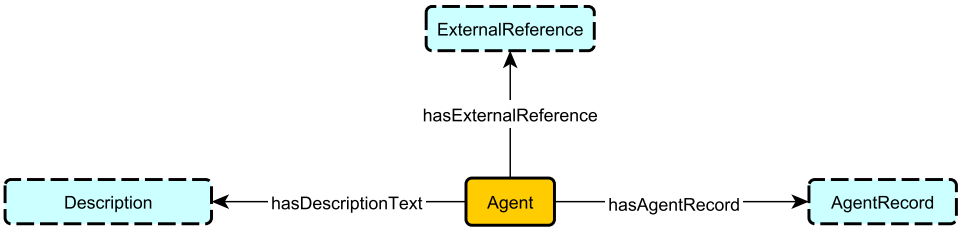


Figure 2.4: Schema Diagram for the Agent module

**Axioms:**

$$\text{Person} \sqsubseteq \text{Agent} \tag{1}$$
$$\text{Organization} \sqsubseteq \text{Agent} \tag{2}$$
$$\forall \text{hasAgentRecord}.\top \sqsubseteq \text{Agent} \tag{3}$$
$$\text{Agent} \sqsubseteq \geq 0 \text{hasExternalReference}.\text{ExternalReference} \tag{4}$$
$$\text{Agent} \sqsubseteq \geq 0 \text{hasDescriptionText}.\text{Description} \tag{5}$$
$$\text{Agent} \sqsubseteq \geq 0 \text{hasAgentRecord}.\text{AgentRecord} \tag{6}$$
$$\neg\text{Person} \sqsubseteq \neg\forall \text{hasAgentRecord}.(\text{OriginRecord} \sqcup \text{RaceRecord} \sqcup \text{AgeRecord} \sqcup \text{SexRecord} \sqcup \text{OccupationRecord} \sqcup \text{PersonStatusRecord}) \tag{7}$$

**Explanation of axioms above:**

1. SubClass: Every person is an agent.
2. SubClass: Every organization is an agent.
3. Domain: The domain of hasAgentRecord is Agent.
4. Structural tautology: An agent may have an external reference.
5. Structural tautology: An agent may have a description.
6. Structural tautology: An agent may have an agent record.
7. Non-Persons cannot have OriginRecords, RaceRecords, AgeRecords, SexRecords, OccupationRecords, or PersonStatusRecords.

## 2.3 ExternalReference

Is used to point to an external reference for an entity, e.g., a particular website about a person or event.

This module (and several others) are hardly more than stubs in the sense of [Krisnadhi and Hitzler, 2017b], as this seems sufficient for the moment.

An external referent can, e.g., be a website or a database, and is given in the form of a URI. The external ID is an identifier given as a string which further describes how the entity can be found in that database or website, and is referent specific.

Note that we are not recording provenance information for this.

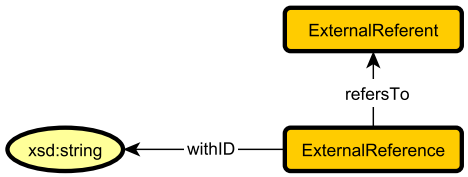


Figure 2.5: Schema Diagram for the ExternalReference module

**Axioms:**

$$\top \sqsubseteq \forall \text{hasExternalReference.ExternalReference} \quad (1)$$
$$\text{ExternalReference} \sqsubseteq \exists \text{refersTo.ExternalReferent} \sqcup \exists \text{withID.xsd:string} \quad (2)$$
$$\text{ExternalReference} \sqsubseteq \leq 1 \text{withID.xsd:string} \sqcap \leq 1 \text{refersTo.ExternalReference} \quad (3)$$
$$\text{ExternalReference} \sqsubseteq \forall \text{refersTo.ExternalReferent} \quad (4)$$
$$\text{ExternalReference} \sqsubseteq \forall \text{withID.xsd:string} \quad (5)$$
$$\top \sqsubseteq \leq 1 \text{hasExternalReference}^{-}.\top \quad (6)$$

**Explanation of axioms above:**

1. Range: The range of hasExternalReference is ExternalReference.
2. An ExternalReference needs to have at least one ExternalReferent or ExternalID.
3. Functionality: An ExternalReference has at most one ExternalID and at most one ExternalReference.
4. Scoped Range: The scoped range of refersTo, scoped by ExternalReference, is ExternalReferent.
5. Scoped Range: The scoped range of withID, scoped by ExternalReference, is xsd:string.
6. Inverse Functionality: The property hasExternalReference is inverse functional, i.e., two different agents cannot share an external reference.

## 2.4 Description

Gives a textual description about an Agent, Event, or Place for additional information about the subject. Each Description is furthermore an EntityWithProvenance, which indicates that we can record provenance information about the description; see the EntityWithProvenance module description for further details.

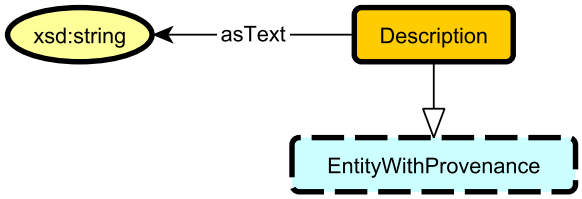


Figure 2.6: Schema Diagram for the Description module

**Axioms:**

$$\top \sqsubseteq \forall \text{hasDescriptionText.Description} \quad (1)$$
$$\text{Description} \sqsubseteq \text{EntityWithProvenance} \quad (2)$$
$$\text{Description} \sqsubseteq = 1 \text{asText.xsd:string} \quad (3)$$
$$\top \sqsubseteq \leq 1 \text{hasDescriptionText}^{-}.\top \quad (4)$$

**Explanation of axioms above:**

1. Range: The range of hasDescriptionText is Description.
2. SubClass: Every description is an entity with provenance.
3. Existential and Funcationality: Every description has exactly one text associated with it.
4. Inverse Functionality: The property hasDescriptionText is inverse functional, i.e., two different agents cannot share a Description.

## 2.5 AgentRecord

Each piece of information about an agent is organized as a record for this agent. I.e. with each agent we typically associate many agent records, the combination of which makes up what we know about this particular agent. Each agent record by itself usually contains one piece of information about the agent, plus provenance information about this piece of information. AgentRecord has a number of sub-modules which are described separately; the intended usage is that every agent record is of one of the types given by these sub-modules.[3]

PersonRecord is a sub-module of AgentRecord, for records that pertain to persons only, and not to agents in general, such as racial information.

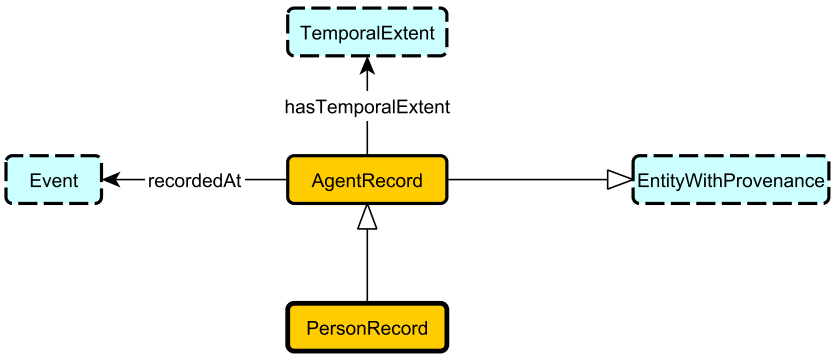


Figure 2.7: Schema Diagram for the AgentRecord module

**Axioms:**

$$\text{AgentRecord} \sqsubseteq \text{EntityWithProvenance} \tag{1}$$
$$\text{PersonRecord} \sqsubseteq \text{AgentRecord} \tag{2}$$
$$\text{hasPersonRecord} \sqsubseteq \text{hasAgentRecord} \tag{3}$$
$$\text{Person} \sqsubseteq \geq 0 \text{hasPersonRecord.PersonRecord} \tag{4}$$
$$\text{AgentRecord} \sqsubseteq = 1 \text{hasAgentRecord}^{-}.\text{Agent} \tag{5}$$
$$\text{PersonRecord} \sqsubseteq = 1 \text{hasPersonRecord}^{-}.\text{Person} \tag{6}$$
$$\text{AgentRecord} \sqsubseteq = 1 \text{hasTemporalExtent.TemporalExtent} \tag{7}$$

[3]Of course, further modules may be added in future versionf of the ontology.

$$\text{AgentRecord} \sqsubseteq \leq 1 \text{recordedAt.Event} \tag{8}$$
$$\text{AgentRecord} \sqsubseteq \leq 1 \text{isDirectlyBasedOn.EntityWithProvenance} \tag{9}$$

**Explanation of axioms above:**

1. SubClass: Every agent record is an entity with provenance.
2. SubClass: Every person record is an agent record.
3. hasPersonRecord is a subProperty of hasAgentRecord.
4. Structural tautology: A person may have a person record.
5. Inverse Existential and Inverse Functionality: Every agent record is record of exactly one agent.
6. Inverse Existential and Inverse Functionality: Every person record is record of exactly one person.
7. Existential and Functionality: Every agent record has exactly one temporal extent.
8. Functionality: Every agent record is recorded at at most one event.
9. Functionality: Every agent record is directly based on at most one source.

**Remarks:**

- RecordTypes are the subClasses of AgentRecord defined in the ontology.

## 2.6 OriginRecord

Describes the ethnolinguistic origin or the place of orgin for an Agent. This in fact applies to persons only, and not to other types of agents.

The origin can be given as either an actual place, as described under the Place module, or can be given in terms of a controlled vaculary, the Enslaved Controlled Vocabulary for Origins (ECVO). Note that any origin record – like all other subClasses of AgentRecord – is also an entity with provenence, i.e., can carry provenance information as detailed in the description of the EntityWithProvenance module.

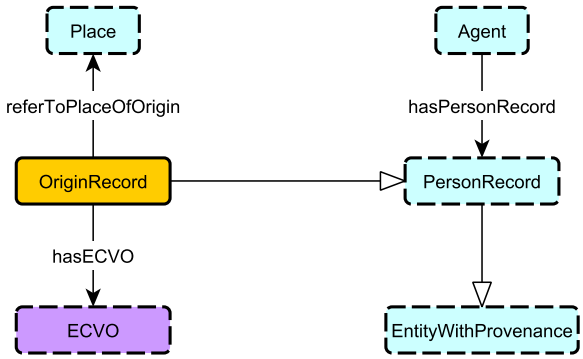


Figure 2.8: Schema Diagram for the OriginRecord module

**Axioms:**

$$\text{OriginRecord} \sqsubseteq \text{AgentRecord} \tag{1}$$
$$\text{OriginRecord} \sqsubseteq \leq 1 \text{hasECVO.ECVO} \tag{2}$$
$$\text{OriginRecord} \sqsubseteq \exists \text{refersToPlaceOfOrigin.Place} \sqcup \exists \text{hasECVO.ECVO} \tag{3}$$
$$\text{OriginRecord} \sqsubseteq \forall \text{refersToPlaceOfOrigin.Place} \tag{4}$$
$$\text{OriginRecord} \sqsubseteq \forall \text{hasECVO.ECVO} \tag{5}$$
$$\text{hasOriginRecord} \sqsubseteq \text{hasPersonRecord} \tag{6}$$

**Explanation of axioms above:**

1. SubClass: Every origin record is an agent record.
2. Functionality: Every origin record has at most one ECVO.
3. Every origin record has at least one ECVO or place associated.
4. Scoped Range: The scoped range of refersToPlaceOfOrigin, scoped by OriginRecord, is Place.
5. Scoped Range: The scoped range of hasECVO, scoped by OriginRecord, is ECVO.
6. hasOriginRecord is a subProperty of hasPersonRecord.

**Remarks:**

- Allowing several places per OriginRecord is deliberate. This depends on the source data.
- Requires controlled vocabulary to be defined.
- Certain ECVO values may only co-occur with certain Places. Corresponding constraints may need to be added when we have the ECVO.

## 2.7 RaceRecord

Describes the race of an Agent. This applies to persons only, not to agents in general. The race is given as a string, as based directly on a source which is listed as the record's provenance.

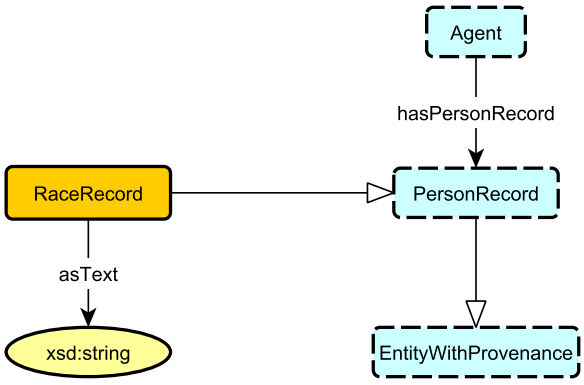


Figure 2.9: Schema Diagram for the RaceRecord module

**Axioms:**

$$\text{RaceRecord} \sqsubseteq \text{AgentRecord} \tag{1}$$
$$\text{RaceRecord} \sqsubseteq \leq 1 \text{asText.xsd:string} \tag{2}$$
$$\text{hasRaceRecord} \sqsubseteq \text{hasPersonRecord} \tag{3}$$

**Explanation of axioms above:**

1. SubClass: Every race record is an agent record.
2. Functionality: Every race record has exactly one text describing race.
3. hasRaceRecord is a subProperty of hasPersonRecord.

## 2.8 AgeRecord

An Age Record records the numerical age and/or the age category of a person; this does not apply to agents in more general. The numeric age is given as a number. Age categories are defined in a controlled vocabulary.

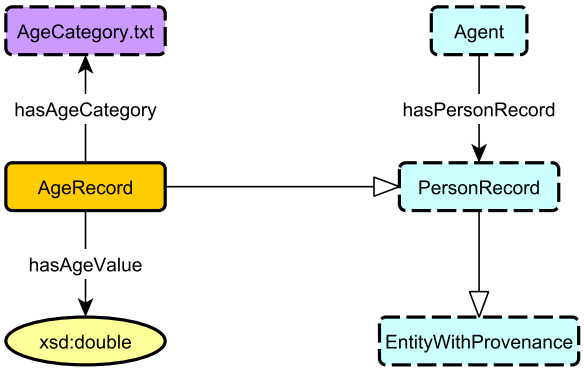


Figure 2.10: Schema Diagram for the AgeRecord module

**Axioms:**

$$\text{AgeRecord} \sqsubseteq \text{AgentRecord} \tag{1}$$
$$\text{AgeRecord} \sqsubseteq \leq 1 \text{hasValue.AgeCategory} \tag{2}$$
$$\text{AgeRecord} \sqsubseteq \leq 1 \text{hasAgeValue.xsd:double} \tag{3}$$
$$\text{AgeRecord} \sqsubseteq \exists \text{hasValue.AgeCategory} \sqcup \exists \text{hasAgeValue.xsd:double} \tag{4}$$
$$\text{hasAgeRecord} \sqsubseteq \text{hasPersonRecord} \tag{5}$$

**Explanation of axioms above:**

1. SubClass: Every age record is an agent record.

2. Functionality: Every age record has at most one age category.
3. Functionality: Every age record has at most one age value.
4. Every age record at least one of category or value.
5. hasAgeRecord is a subProperty of hasPersonRecord.

**Remarks:**

- Requires controlled vocabulary to be defined.
- Age and age category must be compatible if both are present. Will need to revisit this and add constraints once age categories have been determined.

## 2.9 SexRecord

Describes the sex of a person, as given by a source. This does not apply to agents in more general. The sex is given in form of a controlled vocabulary.

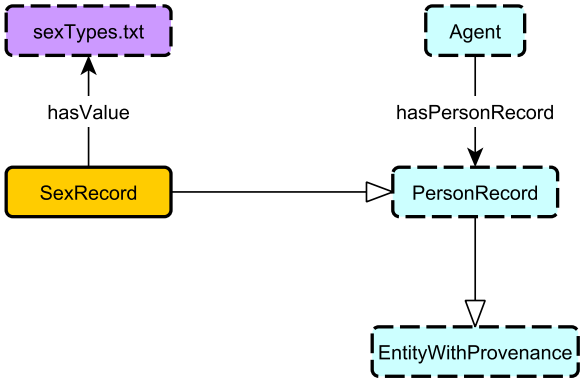


Figure 2.11: Schema Diagram for the SexRecord module

**Axioms:**

$$\text{SexRecord} \sqsubseteq \text{AgentRecord} \tag{1}$$
$$\text{SexRecord} \sqsubseteq =1\text{hasValue.SexTypes} \tag{2}$$
$$\text{hasSexRecord} \sqsubseteq \text{hasPersonRecord} \tag{3}$$

**Explanation of axioms above:**

1. SubClass: Every sex record is an agent record.
2. Existential and Functionality: Every sex record records exactly one sex type.
3. hasSexRecord is a subProperty of hasPersonRecord.

**Remarks:**

- Requires controlled vocabulary to be defined.

## 2.10 OccupationRecord

Describes occupation or skills that a person has, as given in a source. This does not apply to agents in more general. The occupation or skill is given in terms of a controlled vocabulary.

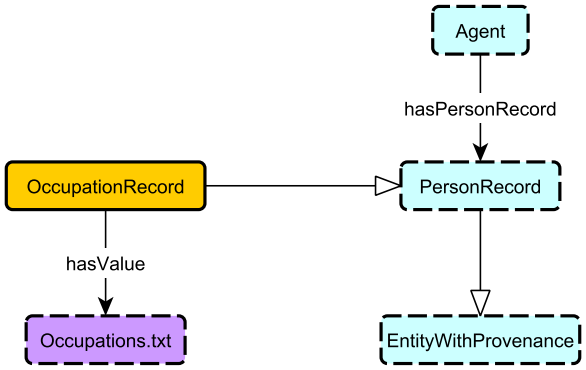


Figure 2.12: Schema Diagram for the OccupationRecord module

**Axioms:**

$$\text{OccupationRecord} \sqsubseteq \text{AgentRecord} \tag{1}$$

$$\text{OccupationRecord} \sqsubseteq =1\text{hasValue.Occupations} \tag{2}$$

$$\text{hasOccupationRecord} \sqsubseteq \text{hasPersonRecord} \tag{3}$$

**Explanation of axioms above:**

1. SubClass: Every occupation record is an agent record.
2. Existential and Functionality: Every occupation record records exactly one occupation.
3. hasOccupationRecord is a subProperty of hasPersonRecord.

**Remarks:**

- Requires controlled vocabulary to be defined.

## 2.11 PersonStatusRecord

Describes the status of a person's freedom, such as being enslaved or freed. This does not apply to agents in more general. The person status is given in terms of a controlled vocabulary.

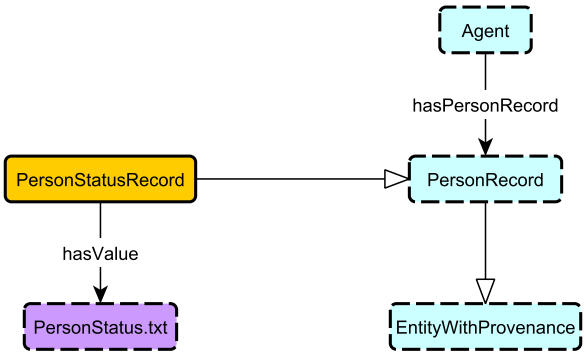


Figure 2.13: Schema Diagram for the PersonStatusRecord module

**Axioms:**

$$\text{PersonStatusRecord} \sqsubseteq \text{AgentRecord} \tag{1}$$
$$\text{PersonStatusRecord} \sqsubseteq =1\text{hasValue.PSCategories} \tag{2}$$
$$\text{hasPersonStatusRecord} \sqsubseteq \text{hasPersonRecord} \tag{3}$$
$$\text{PersonStatusRecord} \sqsubseteq \forall\text{hasStatusGeneratedEvent.Event} \tag{4}$$
$$\exists\text{hasStatusGeneratedEvent.Event} \sqsubseteq \text{PersonStatusRecord} \tag{5}$$
$$\text{hasStatusGeneratedEvent} \sqsubseteq \text{refersToEvent} \tag{6}$$

**Explanation of axioms above:**

1. SubClass: Every person status record is an agent record.
2. Existential and Functionality: Every person status record records exactly one person status category.
3. hasPersonStatusRecord is a subProperty of hasPersonRecord.
4. Scoped Range: The scoped range of hasStatusGeneratedEvent, scoped by PersonStatusRecord, is Event.
5. Scoped Domain: The scoped domain of hasStatusGeneratedEvent, scoped by Event, is PersonStatusRecord.
6. hasStatusGeneratedEvent is a subProperty of refersToEvent. Please see the Event module for refersToEvent.

**Remarks:**

- Requires controlled vocabulary to be defined.

## 2.12 NameRecord

Descrbes the name(s) of an agent. While names are complex entities, ontologically speaking, we opted for an extended stub in this case, which captures what is required for Enslaved. A name record can carry at most one preferred name variant, and additional name variants. Each name

variant in turn must have a full name given as string, and may optionally also have at most one each of first name and surname, given as strings.

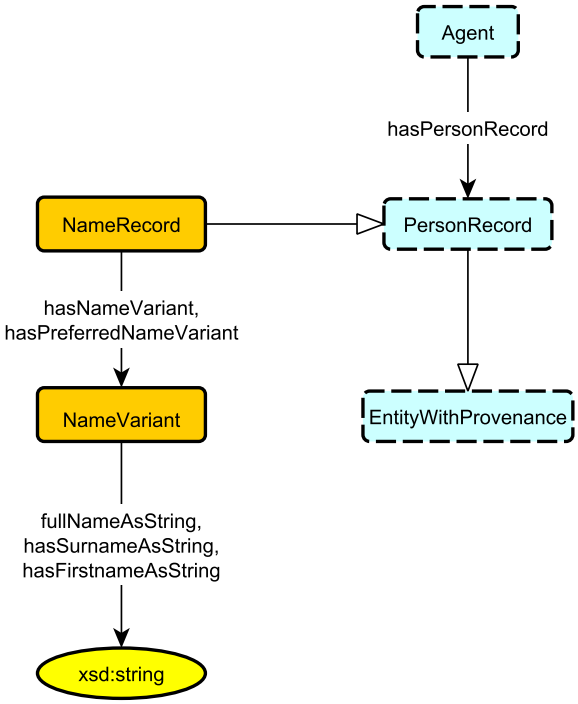


Figure 2.14: Schema Diagram for the NameRecord module

**Axioms:**

$$\text{NameRecord} \sqsubseteq \text{AgentRecord} \tag{1}$$
$$\top \sqsubseteq \forall \text{hasNameVariant.NameVariant} \tag{2}$$
$$\text{hasPreferredNameVariant} \sqsubseteq \text{hasNameVariant} \tag{3}$$
$$\text{NameRecord} \sqsubseteq \leq 1 \text{hasPreferredNameVariant.NameVariant} \tag{4}$$
$$\text{NameRecord} \sqsubseteq \exists \text{hasNameVariant.NameVariant} \tag{5}$$
$$\text{NameVariant} \sqsubseteq = 1 \text{fullNameAsString.xsd:string} \tag{6}$$
$$\text{NameVariant} \sqsubseteq \leq 1 \text{hasFirstnameAsString.xsd:string} \tag{7}$$
$$\text{NameVariant} \sqsubseteq \leq 1 \text{hasSurnameAsString.xsd:string} \tag{8}$$
$$\top \sqsubseteq \forall \text{fullNameAsString.xsd:string} \tag{9}$$
$$\top \sqsubseteq \forall \text{hasFirstNameAsString.xsd:string} \tag{10}$$
$$\top \sqsubseteq \forall \text{hasSurnameAsString.xsd:string} \tag{11}$$
$$\text{hasNameRecord} \sqsubseteq \text{hasAgentRecord} \tag{12}$$

**Explanation of axioms above:**

1. SubClass: Every name record is an agent record.
2. Range: The range of hasNameVariant is NameVariant.
3. hasPreferredNameVariant is a subProperty of hasNameVariant.
4. Functionality: Every name record has at most one preferred name variant.
5. Existential: Every name record has at least one name variant.
6. Existential and Functionality: Every name variant has exactly one full name as string.
7. Functionality: Every name variant has at most one first name as string.
8. Functionality: Every name variant has at most one surname as string.
9. Range: The range of fullNameAsString is xsd:string.
10. Range: The range of hasFirstNameAsString is xsd:string.
11. Range: The range of hasSurnameAsString is xsd:string.
12. hasNameRecord is a subProperty of hasAgentRecord.

## 2.13 InterAgentRelationshipRecord

Describes relationships between two agents as a 'from-to' pair. These relationships are distinct from participant roles which are discussed further below. Examples include family relations or ownership relations. Relationships are typed by a controled vocabulary, see Section 2.13.1 below.

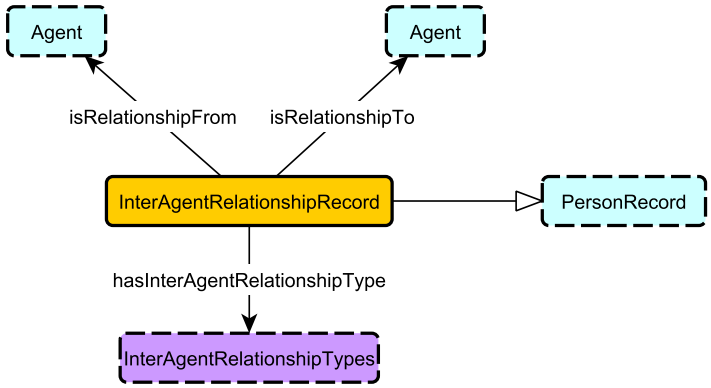


Figure 2.15: Schema Diagram for the InterAgentRelationshipRecord module

**Axioms:**

$$\text{InterAgentRelationshipRecord} \sqsubseteq \text{AgentRecord} \tag{1}$$
$$\text{InterAgentRelationshipRecord} \sqsubseteq =1\text{isRelationshipFrom.Agent} \tag{2}$$
$$\text{InterAgentRelationshipRecord} \sqsubseteq =1\text{isRelationshipTo.Agent} \tag{3}$$
$$\text{InterAgentRelationshipRecord} \sqsubseteq \forall\text{isRelationshipFrom.Agent} \tag{4}$$
$$\text{InterAgentRelationshipRecord} \sqsubseteq \forall\text{isRelationshipTo.Agent} \tag{5}$$
$$\exists\text{isRelationshipFrom.Agent} \sqsubseteq \text{InterAgentRelationshipRecord} \tag{6}$$

$$\exists \text{isRelationshipto.Agent} \sqsubseteq \text{InterAgentRelationshipRecord} \quad (7)$$

$$\text{InterAgentRelationshipRecord} \sqsubseteq \leq 1 \text{hasInterAgentRelationshipType.InterAgentRelationshipTypes} \quad (8)$$

$$\text{InterAgentRelationshipRecord} \sqsubseteq \forall \text{hasInterAgentRelationshipType.InterAgentRelationshipTypes} \quad (9)$$

$$\exists \text{hasInterAgentRelationshipType.InterAgentRelationshipTypes} \sqsubseteq \text{InterAgentRelationshipRecord} \quad (10)$$

$$\text{hasInterAgentRelationshipRecordRecord} \sqsubseteq \text{hasAgentRecord} \quad (11)$$

**Explanation of axioms above:**

1. SubClass: Every inter agent relationship record is an agent record.
2. Existential and Functionality: Every inter agent relationship record records a relationship from exactly one agent.
3. Existential and Functionality: Every inter agent relationship record records a relationship to exactly one agent.
4. Scoped Range: The scoped range of isRelationshipFrom, scoped by InterAgentRelationshipRecord, is Agent.
5. Scoped Range: he scoped range of isRelationshipTo, scoped by InterAgentRelationshipRecord, is Agent.
6. Scoped Domain: The scoped domain of isRelationshipFrom, scoped by Agent, is InterAgentRelationshipRecord.
7. Scoped Domain: The scoped domain of isRelationshipTo, scoped by Agent, is InterAgentRelationshipRecord.
8. Functionality: Every inter agent relationship record is of at most one inter-agent relationship type.
9. Scoped Range: The scoped range of hasInterAgentRelationshipType, scoped by InterAgentRelationshipRecord, is InterAgentRelationshipType.
10. Scoped Domain: The scoped domain of hasInterAgentRelationshipType, scoped by InterAgentRelationshipTypes, is InterAgentRelationshipRecord.
11. hasInterAgentRelationshipRecord is a subProperty of hasAgentRecord.

**Remarks:**

- If necessary later, the InterAgentRelationshipTypes can easily be typecast into subclasses of InterAgentRelationship [Krisnadhi et al., 2015b].
- We would like to further impose that no agent can be in an InterAgentRelationship with itself. However it seems that this cannot be expressed in OWL DL, since non-simple OWL properties cannot be irreflexive [Motik et al., 2012, Hitzler et al., 2010].

### 2.13.1 InterAgentRelationshipTypes Controlled Vocabulary

The following InterAgentRelationshipTypes are provided by means of the controled vocabulary mentioned above. The list is not exhaustive; the rationale for using a controled vocabulary is that it makes it easy to add further vocabulary terms as needed: Parent, Child, Grandparent, GrandChild, Sibling, Spouse, MotherInLaw, FatherInLaw, DaughterInLaw, SonInLaw, Cousin, Aunt, Uncle, Godmother, Godfather, Owner.

**Remarks:**

- We would like to impose axiomatic relationships between the InterAgentRelationshipTypes terms, such as some inter agent relationships being symmetric, functional or inverse functional, or even more complex relationships which would e.g., express how being an uncle can be composed of a sibling and a parent relationships. However, none of these axioms can be expressed in OWL DL. Some of them would necessitate the use of property characteristics for non-simple properties, which is not allowed according to the specification, see [Motik et al., 2012, Hitzler et al., 2010]. Some would necessitate the use of right-hand-side property chains, which are also not expressible in OWL DL – see [Krisnadhi et al., 2015b] for a discussion of related matters.

## 2.14 ParticipantRoleRecord

Describes the role in which an agent participates in event. The role is specific to an event, and participant role types are described in a controlled vocabulary.

Part of this module is derived from the pattern described in [Krisnadhi, 2016].

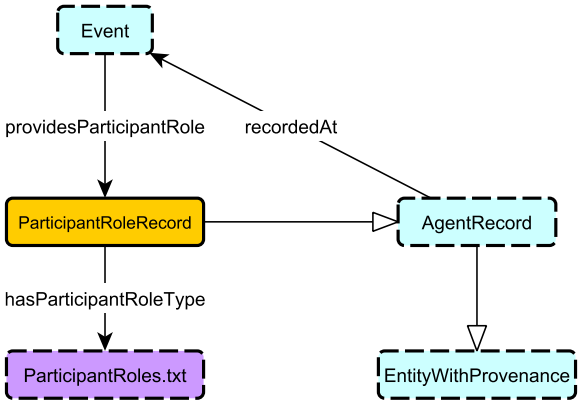


Figure 2.16: Schema Diagram for the ParticipantRoleRecord module

**Axioms:**

$$\text{ParticipantRoleRecord} \sqsubseteq \text{AgentRecord} \tag{1}$$

$$\text{ParticipantRoleRecord} \sqsubseteq =1\text{providesParticipantRole}^{-}.\text{Event} \tag{2}$$

$$\top \sqsubseteq \forall\text{providesParticipantRole}.\text{ParticipantRoleRecord} \tag{3}$$

$$\text{ParticipantRoleRecord} \sqsubseteq \leq 1\text{hasParticipantRoleType}.\text{ParticipantRoleTypes} \tag{4}$$

$$\exists\text{hasParticipantRoleType}.\text{ParticipantRoleType} \sqsubseteq \text{ParticipantRoleRecord} \tag{5}$$

$$\text{ParticipantRoleRecord} \sqsubseteq \forall\text{hasParticipantRoleType}.\text{ParticipantRoleType} \tag{6}$$

$$\text{hasParticipantRoleRecord} \sqsubseteq \text{hasAgentRecord} \tag{7}$$

$$\text{roleProvidedBy} \sqsubseteq \text{hasParticipantRoleRecord}^{-} \tag{8}$$

**Explanation of axioms above:**

1. SubClass: Every participant role record is an agent record.
2. Inverse Existential and Inverse Functionality: Every participant role record pertains to exactly one event.
3. Range: The range of providesParticipantRole is ParticipantRoleRecord.
4. Functionality: Every participant role record has at most one participante role type.
5. Scoped Domain: The scoped domain of hasParticipantRoleType, scoped by ParticipantRoleType, is ParticipantRoleRecord.
6. Scoped Range: The scoped range of hasParticipantRoleType, scoped by ParticipantRoleRecord, is ParticipantRoleType.
7. hasParticipantRoleRecord is a subProperty of hasAgentRecord.
8. roleProvidedBy is defined as the inverse of hasParticipantRoleRecord. This is added for convenience.

**Remarks:**

- Need to add specific participant role types as controlled vocabulary, as soon as controlled vocabulary is available.
- If necessary later, the ParticipantRoles can easily be typecast into subclasses of ParticipantRoleRecord [Krisnadhi et al., 2015b].

## 2.15 Event

Describes an event where agents are reported on and given roles in. Agent records are often generated by documents that describe an event. So, similar to agent records, events play a central role in the fabric of the Enslaved Ontology, as they tie together records of agents and the time and places where recording events happened.

Events are typed; the types are governed by a controlled vocabulary.

Parts of this module are borrowed from [Krisnadhi and Hitzler, 2017a], simplified and extended.

**Axioms:**

$$\text{Event} \sqsubseteq \geq 1 \text{hasName.xsd:string} \tag{1}$$
$$\text{Event} \sqsubseteq \geq 0 \text{hasDescriptionText.Description} \tag{2}$$
$$\text{Event} \sqsubseteq \geq 1 \text{hasExternalReference.ExternalReference} \tag{3}$$
$$\text{Event} \sqsubseteq \exists \text{atPlace.Place} \tag{4}$$
$$\text{Event} \sqsubseteq \geq 1 \text{providesParticipantRole.ParticipantRoleRecord} \tag{5}$$
$$\text{Event} \sqsubseteq = 1 \text{hasTemporalExtent.TemporalExtent} \tag{6}$$
$$\top \sqsubseteq \forall \text{atPlace.Place} \tag{7}$$
$$\top \sqsubseteq \forall \text{hasName.xsd:string} \tag{8}$$
$$\exists \text{subEventOf.Event} \sqsubseteq \text{Event} \tag{9}$$
$$\top \sqsubseteq \forall \text{subEventOf.Event} \tag{10}$$
$$\text{Event} \sqsubseteq = 1 \text{hasEventType.EventTypes} \tag{11}$$

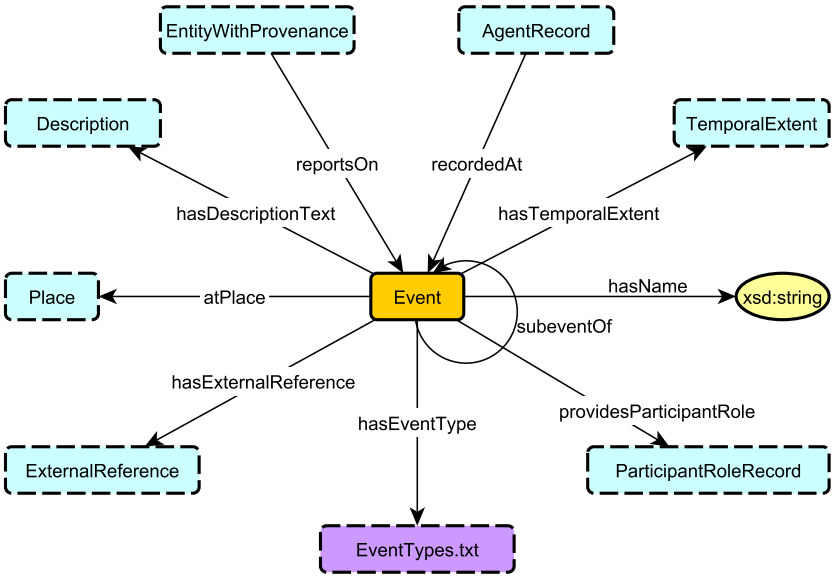


Figure 2.17: Schema Diagram for the Event module

$$\exists \text{hasEventType.EventTypes} \sqsubseteq \text{Event} \tag{12}$$

$$\text{Event} \sqsubseteq \forall \text{hasEventType.EventTypes} \tag{13}$$

$$\text{AgentRecord} \sqsubseteq \leq 1 \text{reportsOn.Event} \tag{14}$$

$$\text{AgentRecord}(x) \wedge \text{recordedAt}(x, y) \wedge \text{Event}(y) \rightarrow \text{reportsOn(x,y)} \tag{15}$$

$$\text{EntityWithProvenance} \sqsubseteq \geq 0 \text{reportsOn.Event} \tag{16}$$

$$\top \sqsubseteq \forall \text{refersToEvent.Event} \tag{17}$$

**Explanation of axioms above:**

1. Existential: Every event has at least one name.
2. Structural tautology: An event may have a description.
3. Existential: Every event has at least one external reference.
4. Existential: An event always has a place.
5. Existential: Every event provides at least one participant role.
6. Existential and Functionality: Every event has exactly one temporal extent.
7. Range: The range of atPlace is Place.
8. Range: The range hasName is xsd:string.
9. Domain: The domain of subEventOf is Event.
10. Range: The range of subEventOf is Event.
11. Existential and Functionality: Each event has exacty one event type.
12. Scoped Domain: The scoped domain of hasEventType, scoped by EventTypes, is Event.
13. Scoped Range: The scoped range of hasEventType, scoped by Event, is EventTypes.
14. Functionality: An agent record reports on at most one event.
15. If an agent record is recorded at an event, then that agent record also reports on this event.
16. Provenance entities may report on events.
17. Range: The range of refersToEvent is Event.

**Remarks:**

- Need to add specific event types as controlled vocabulary, as soon as controlled vocabulary is available.
- If necessary later, the EventTypes can easily be typecast into subclasses of Event [Krisnadhi et al., 2015b].
- Note that axiom (4) does *not* mean that the place has to be known or stored in the graph or database due to the open world assumption [Hitzler et al., 2010].
- Note that an event can be at several places, but has only one temporal extent.
- Functionality of recordedAt has been declared earlier already, in Section 2.5.
- Axiom 15 has been stated using a rule, becaus we believe this is more readable. It can be expressed in description logic syntax using three axioms, by making use of *rolification* [Krisnadhi et al., 2011], as follows.

$$\begin{aligned} \text{AgentRole} &\sqsubseteq \exists \text{agentRoleSelfProperty}.\text{Self} \\ \text{Event} &\sqsubseteq \exists \text{eventSelfProperty}.\text{Self} \\ \text{agentRoleSelfProperty} \circ \text{recordedAt} \circ \text{eventSelfProperty} &\sqsubseteq \text{reportsOn} \end{aligned}$$

- The property refersToEvent is used if something refers to an event, in a not further specified (i.e.) informal) way.

**Questions:**

- If an event happens at some temporal extent, does that give rise to constraints on temporal extents of participant roles provided by the event? We need to check on this later when we know about concrete participant roles.

## 2.16 EntityWithProvenance

This module provides the ability to talk about the source document that an agent record is generated from. It also allows for provenance chains that describe the transformation of the data by different actvities.

The intended use, which is mirrored by the axiomatization, is that any agent record is at the same time an entity with provenance, which is directly based on *exactly one*[4] other entity with provenance. In addition, entities with provenance may carry the document type of the original source, and indeed at most one, which is governed by a controlled vocabulary: this document type refers to the type of document of the historic source document from which the current information has originally been retrieved . Furthermore, an entity with provenance may carry information from where the document may be available, e.g., what library it is hosted at or what database it can be found in, as well as concrete reference URIs which point at a concrete online resource. Entities furthermore carry license information, taken from a controlled vocabulary.

Parts of this module are heavily borrowed from PROV-O [Lebo et al., 2013].

[4]This functionality axiom was given in the AgentRecord module description

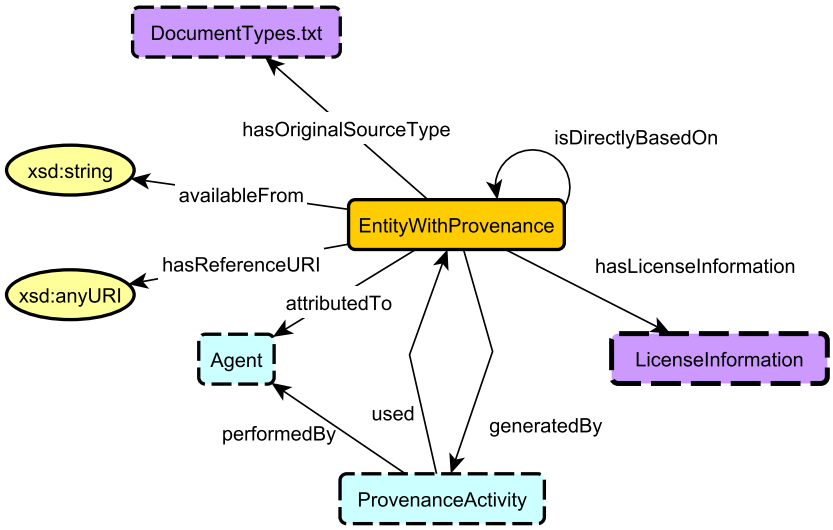


Figure 2.18: Schema Diagram for the EntityWithProvenance module

**Axioms:**

$$\text{EntityWithProvenance} \sqsubseteq \exists \text{availableFrom.xsd:string} \quad (1)$$
$$\text{EntityWithProvenance} \sqsubseteq \geq 0 \text{hasReferenceURI.xsd:anyURI} \quad (2)$$
$$\text{EntityWithProvenance} \sqsubseteq \exists \text{hasLicenseInformation.LicenseInformation} \quad (3)$$
$$\text{EntityWithProvenance} \sqsubseteq \leq 1 \text{hasOriginalSourceType.DocumentTypes} \quad (4)$$
$$\text{EntityWithProvenance} \sqsubseteq \forall \text{availableFrom.xsd:string} \quad (5)$$
$$\text{EntityWithProvenance} \sqsubseteq \forall \text{hasLicenseInformation.LicenseInformation} \quad (6)$$
$$\text{EntityWithProvenance} \sqsubseteq \forall \text{hasOriginalSourceType.DocumentTypes} \quad (7)$$
$$\exists \text{hasOriginalSourceType.DocumentTypes} \sqsubseteq \text{EntityWithProvenance} \quad (8)$$
$$\text{EntityWithProvenance} \sqsubseteq \forall \text{isDirectlyBasedOn.EntityWithProvenance} \quad (9)$$
$$\exists \text{attributedTo.Agent} \sqsubseteq \text{EntityWithProvenance} \quad (10)$$
$$\text{EntityWithProvenance} \sqsubseteq \forall \text{attributedTo.Agent} \quad (11)$$
$$\exists \text{generatedBy.ProvenanceActivity} \sqsubseteq \text{EntityWithProvenance} \quad (12)$$
$$\text{EntityWithProvenance} \sqsubseteq \forall \text{generatedBy.ProvenanceActivity} \quad (13)$$
$$\exists \text{used.EntityWithProvenance} \sqsubseteq \text{ProvenanceActivity} \quad (14)$$
$$\text{ProvenanceActivity} \sqsubseteq \forall \text{used.EntityWithProvenance} \quad (15)$$
$$\exists \text{performedBy.Agent} \sqsubseteq \text{ProvenanceActivity} \quad (16)$$
$$\text{ProvenanceActivity} \sqsubseteq \forall \text{performedBy.Agent} \quad (17)$$
$$\text{isDirectlyBasedOn} \circ \text{hasOriginalSourceType} \sqsubseteq \text{hasOriginalSourceType} \quad (18)$$

$$\text{isDirectlyBasedOn}^{-} \circ \text{hasOriginalSourceType} \sqsubseteq \text{hasOriginalSourceType} \quad (19)$$

**Explanation of axioms above:**

1. Existential: Every entity with provenance has at least one thing it is available from.
2. EntityWithProvenance may have a reference URI
3. Existential: Every entity with provenance has at least on license information.
4. Existential: Every entity with provenance has at most one originaly source document type.
5. Scoped Range: The scoped range of availableFrom, scoped by EntityWithProvenance, is xsd:string.
6. Scoped Range: The scoped range of hasLicenseInformation, scoped by EntityWithProvenance, is LicenseInformation.
7. Scoped Range: The scoped range of hasOriginalSourceType, scoped by EntityWithProvenance, is DocumentTypes.
8. Scoped Domain: The scoped domain of hasOriginalSourceType, scoped by DocumentTypes, is EntityWithProvenance.
9. Scoped Range: The scoped range of isDirectlyBasedOn, scoped by EntityWithProvenance, is EntityWithProvenance.
10. Scoped Domain:The scoped domain of attributedTo, scoped by Agent, is EntityWithProvenance.
11. Scoped Range: The scoped range of attributedTo, scoped by EntityWithProvenance, is Agent.
12. Scoped Domain:The scoped domain of generatedBy, scoped by ProvenanceActivity, is EntitWithProvenance.
13. Scoped Range: The scoped range of generatedBy, scoped by EntityWithProvenance, is ProvenanceActivity.
14. Scoped Domain:The scoped domain of used, scoped by EntityWithProvenance, is ProvenanceActivity
15. Scoped Range: The scoped range of used, scoped by ProvenanananceActivity, is EntityWithProvenance.
16. Scoped Domain:The scoped domain of performedBy, scoped by Agent, is ProvenanceActivity.
17. Scoped Range: The scoped range of performedBy, scoped by ProvenanceActivity, is Agent.
18. The original source type of an entity with provenance is the same as that of the entity with provenance it is directly based on.
19. The original source type of an entity with provenance is the same as that of any entity with provenance directly based on it.

**Remarks:**

- Need to add specific document types as subclasses from controlled vocabulary, as soon as controlled vocabulary is available.
- If necessary later, the EventTypes can easily be typecast into subclasses of Event [Krisnadhi et al., 2015b].
- While a provenance activity can in general be any activity, in this version of the Enslaved data we are restricting these to research projects, as specified in Section 2.17 below.

## 2.17 ResearchProject

With this module we cater for information about research projects which contribute data to Enslaved. A research project is considered to be an event, and as such inherits what has been described above for the Event module; however at this stage we have only limited requirements regarding the information to be stored about research projects. We describe all of these in the following (including, in the schema diagram, those which are redundant because of inheritance from Event), constituting the ResearchProject module. Within the fabric of the Enslaved Ontology, research projects will in particular play the role of provenance activities, as described above in Section 2.16.

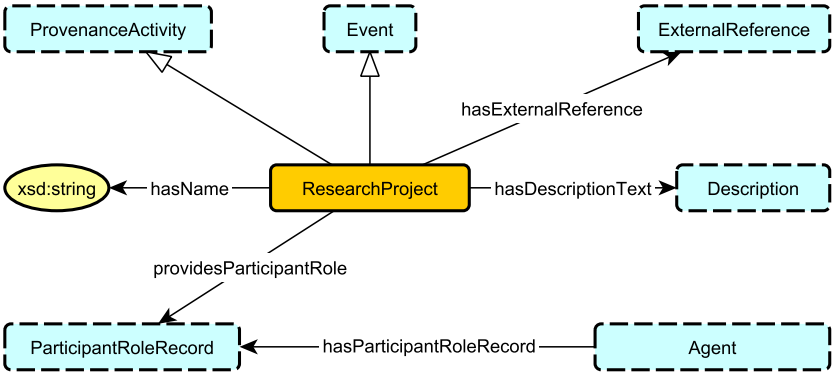


Figure 2.19: Schema Diagram for the ResearchProject module

**Axioms:**

We do not repeat axioms which are inherited from Event or ProvenanceActivity. In fact, the subClass statements are all we need, except for the provision of a number of participant roles. For the moment, we restrict this to ResearchProjectPIRole and ResearchProjectContributorRole.

$$\text{ResearchProject} \sqsubseteq \text{Event} \tag{1}$$

$$\text{ResearchProject} \sqsubseteq \text{ProvenanceActivity} \tag{2}$$

$$\text{ResearchProjectPIRole} \sqsubseteq \forall \text{providesParticipantRole}^{-}.\text{ResearchProject} \tag{3}$$

$$\text{ResearchProjectContributorRole} \sqsubseteq \forall \text{providesParticipantRole}^{-}.\text{ResearchProject} \tag{4}$$

$$\text{ResearchProjectPIRole} \sqsubseteq \text{Researcher} \tag{5}$$

$$\text{ResearchProjectContributorRole} \sqsubseteq \text{Researcher} \tag{6}$$

**Explanation of axioms above:**

1. ResearchProject is a subClass of Event.
2. ResearchProject is a subClass of ProvenanceActivity.
3. The research project PI role can only be provided by an event which is a research project.

4. The research project contributor role can only be provided by an event which is a research project.
5. ResearchProjectPIRole is a subClass of Researcher.
6. ResearchProjectContributorRole is a subClass of Researcher.

**Remarks:**

- In terms of data, the name for a research project will always be unique. However it would seem odd to ontologically require this.

## 2.18 Place

This module provides the ability represent information about the spatial location of a place and provides information about the type of place. This place module is intended as an incomplete but funcational place holder for a more complete and comprehensive module in form of a historical gazeteer which may become available in the future. It is acknowledged that our module constitutes a rather crude model.

Places can be given in two different ways, either as a latitude/longitude pair, or in terms of a piece of controlled vocabulary. Places furthermore may be typed by a controlled vocabulary. Every place must have a name, and may have alternate names.

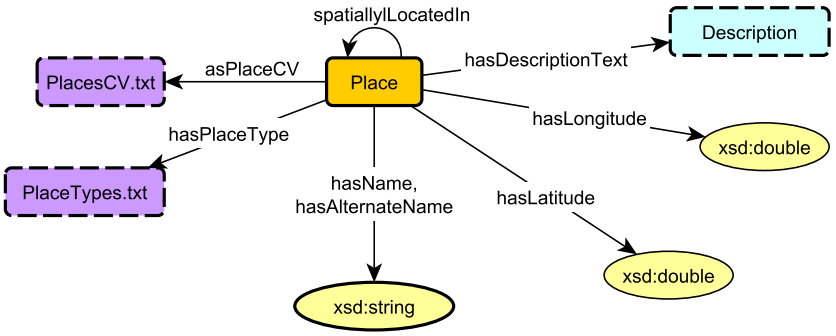


Figure 2.20: Schema Diagram for the Place module

**Axioms:**

$$\text{Place} \sqsubseteq \geq 0 \text{hasDescriptionText.Description} \quad (1)$$
$$\text{Place} \sqsubseteq = 1 \text{hasName.xsd:string} \quad (2)$$
$$\text{Place} \sqsubseteq \geq 0 \text{hasAlternateName.xsd:string} \quad (3)$$
$$\top \sqsubseteq \forall \text{hasAlternateName.xsd:string} \quad (4)$$
$$\top \sqsubseteq \forall \text{hasLatitude.xsd:double} \quad (5)$$
$$\top \sqsubseteq \forall \text{hasLongitude.xsd:double} \quad (6)$$
$$\text{Place} \sqsubseteq \geq 0 \text{spatiallyLocatedIn.Place} \quad (7)$$

$$\text{Place} \sqsubseteq \geq 0 \text{hasPlaceType.PlaceTypes} \quad (8)$$

$$\exists \text{asPlaceCV.PlacesCV} \sqsubseteq \text{Place} \quad (9)$$

$$\text{Place} \sqsubseteq \forall \text{asPlaceCV.PlacesCV} \quad (10)$$

$$\exists \text{hasPlaceType.PlaceTypes} \sqsubseteq \text{Place} \quad (11)$$

$$\text{Place} \sqsubseteq \forall \text{hasPlaceType.PlaceTypes} \quad (12)$$

$$\text{Place} \sqsubseteq \exists \text{asPlaceCV.PlacesCV} \sqcup \exists \text{hasDescriptionText.Description} \sqcup \exists \text{hasName.xsd:string} \sqcup \exists \text{hasCoordinates.Coordinates} \quad (13)$$

$$\text{Place} \sqsubseteq \text{EntityWithProvenance} \quad (14)$$

$$\exists \text{hasCoordinates.Coordinates} \sqsubseteq \text{Place} \quad (15)$$

$$\top \sqsubseteq \forall \text{hasCoordinates.Coordinates} \quad (16)$$

$$\text{Coordinates} \sqsubseteq \exists \text{hasLatitude.xsd:double} \quad (17)$$

$$\text{Coordinates} \sqsubseteq \exists \text{hasLongitude.xsd:double} \quad (18)$$

$$\exists \text{hasLatitude.xsd:double} \sqsubseteq \text{Coordinates} \quad (19)$$

$$\exists \text{hasLongitude.xsd:double} \sqsubseteq \text{Coordinates} \quad (20)$$

**Explanation of axioms above:**

1. Structural tautology: A place may have a description.
2. Existential and Functionality: A place has exactly one name.
3. Structural tautology: A place may have an alternate name.
4. Range: The range of hasAlternateName is xsd:string.
5. Range: The range of hasLatitude is xsd:double.
6. Range: The range of hasLongitude is xsd:double.
7. Structural tautology: A place may be spatially located in a place.
8. Structural tautology: A place may have a place type.
9. Scoped Domain: The scoped domain of asPlaceCV, scoped by PlacesCV, is Place.
10. Scoped Range: The scoped range of asPlaceCV, scoped by Place, is PlacesCV.
11. Scoped Domain: The scoped domain of hasPlaceType, scoped by PlaceTypes, is Place.
12. Scoped Range: The scoped range of hasPlaceType, scoped by Place, is PlaceTypes.
13. Several alternative Existentials: Each place is a controlled vocabulary or has a description or has a name or has coordinates.
14. subClass: Every Place is an EntityWithProvenance.
15. Scoped Domain: The scoped domain of hasCoordinates, scoped by Coordinates, is Place.
16. Range: The range of hasCoordinates is Coordinates.
17. Existential: Coordinates always has a Latitude.
18. Existential: Coordinates always has a Latitude.
19. Domain: The domain of hasLatitude is Coordinates.
20. Domain: The domain of hasLongitude is Coordinates.

**Remarks:**

- Need to add specific places and place types as controlled vocabularies, as soon as controlled vocabularies are available.
- If necessary later, the PlaceTypes can easily be typecast into subclasses of Place [Krisnadhi et al., 2015b].

- We abstain on the question whether every place necessarily has a place type, i.e. we avoid the ontological committment of casting this into an axiom.
- For export purposes, PlacesCVs can have provenance information attached to them; this is not reflected in the current documentation.

## 2.19 TemporalExtent

This module provides the capability to describe specific points in time and the relative time of different points in time. It is not intended as a complete model of temporal expressions, and may be replaced in later versions with more expressive models.

A temporal extent may start and end at a point in time, or may be circumscribed by occuring before or after a point in time, or by containing a point in time, or by falling within a timespan with given start and end points in time.

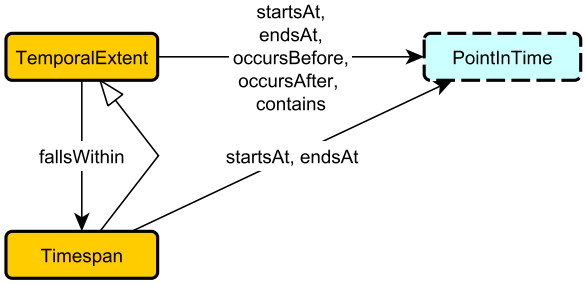


Figure 2.21: Schema Diagram for the TemporalExtent module, without PointInTime specified

For this version of the Enslaved Ontology we consider as points in time only dates. However, it is to be expected that this may have to be refined in a later version. Thus, for now, Figure 2.22 shows the schema diagram for the TemporalExtent module as we use it in this version of the ontology.

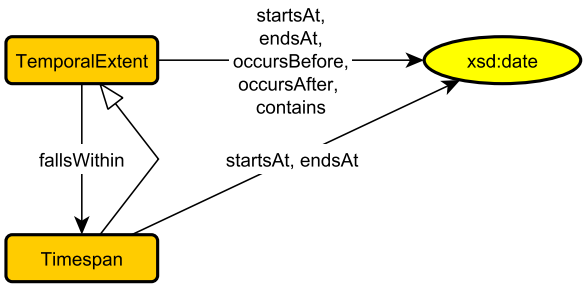


Figure 2.22: Schema Diagram for the TemporalExtent module, with PointInTime given as xsd:data

**Axioms:**

$$\top \sqsubseteq \forall \text{hasTemporalExtent.TemporalExtent} \quad (1)$$
$$\exists \text{startsAt.xsd:date} \sqsubseteq \text{TemporalExtent} \quad (2)$$
$$\text{TemporalExtent} \sqsubseteq \forall \text{startsAt.xsd:date} \quad (3)$$
$$\exists \text{endsAT.xsd:date} \sqsubseteq \text{TemporalExtent} \quad (4)$$
$$\text{TemporalExtent} \sqsubseteq \forall \text{endsAt.xsd:date} \quad (5)$$
$$\exists \text{occursBefore.xsd:date} \sqsubseteq \text{TemporalExtent} \quad (6)$$
$$\text{TemporalExtent} \sqsubseteq \forall \text{occursBefore.xsd:date} \quad (7)$$
$$\exists \text{occursAfter.xsd:date} \sqsubseteq \text{TemporalExtent} \quad (8)$$
$$\text{TemporalExtent} \sqsubseteq \forall \text{occursAfter.xsd:date} \quad (9)$$
$$\exists \text{contains.xsd:date} \sqsubseteq \text{TemporalExtent} \quad (10)$$
$$\text{TemporalExtent} \sqsubseteq \forall \text{contains.xsd:date} \quad (11)$$
$$\exists \text{fallsWithin.Timespan} \sqsubseteq \text{TemporalExtent} \quad (12)$$
$$\text{TemporalExtent} \sqsubseteq \forall \text{fallsWithin.Timespan} \quad (13)$$
$$\text{Timespan} \sqsubseteq \exists \text{startsAt.xsd:date} \quad (14)$$
$$\text{Timespan} \sqsubseteq \exists \text{EndsAt.xsd:date} \quad (15)$$

**Explanation of axioms above:**

1. Range: The range of hasTemporalExtent is TemporalExtent.
2. Scoped Domain: The scoped domain of startsAt, scoped by xsd:date, is TemporalExtent.
3. Scoped Range: The scoped range of startsAt, scoped by TemporalExtent, is PointInTime.
4. Scoped Domain: The scoped domain of endsAt, scoped by xsd:date, is TemporalExtent.
5. Scoped Range: The scoped range of endsAt, scoped by TemporalExtent, is xsd:date.
6. Scoped Domain: The scoped domain of occursBefore, scoped by xsd:date, is TemporalExtent.
7. Scoped Range: The scoped range of occursBefore, scoped by TemporalExtent, is xsd:date.
8. Scoped Domain: The scoped domain of occursAfter, scoped by xsd:date, is TemporalExtent.
9. Scoped Range: The scoped range of occursAfter, scoped by TemporalExtent, is xsd:date.
10. Scoped Domain: The scoped domain of contains, scoped by xsd:date, is TemporalExtent.
11. Scoped Range: The scoped range of contains, scoped by TemporalExtent, is xsd:date.
12. Scoped Domain: The scoped domain of fallsWithin, scoped by TimeSpan, is TemporalExtent.
13. Scoped Range: The scoped range of fallsWithin, scoped by TemporalExtent, is TimeSpan.
14. Existential: Every time span has a starting point in time.
15. Existential: Every time span has an ending point in time.

**Questions:**

- In OWL, can a property chain have a data property as last property in the chain? If not, we have to remove the corresponding axioms above and discuss this in the text.
- In OWL, can we do scoped domains and ranges for data properties? If not, we have to remove the corresponding axioms above and discuss this in the text.

# 3 Putting Things Together

The Enslaved Ontology (version 1.0) is constituted by the union of the modules described previously, plus a few global axioms not yet listed with the modules, as well as some meta-level annotations using the Ontology Design Pattern Represntation Language OPLa [Hitzler et al., 2017b, Shimizu et al., 2018].

We consider the controlled vocabularies to be separate from the actual ontology. One advantage of using controlled vocabularies as indicated in this document is, that they provide a seamless capability for expansion of the ontology, by adding further vocabulary items. Sometimes, however, it is the case that there are specific interactions between items in the controlled vocabulary and axioms. For example, certain EventTypes might only provide certain types of ParticipantRoles. These axioms will be provided separately from the main ontology; there will be documentation indicating which versions of the ontology and controlled vocabularies they are meant to complement.

Figure 3.1 shows a schema diagram for the combined ontology. Marked with a red dashed arrow in Figure 3.1 are some suggested *shortcuts* which we discuss further below. An alternative schema diagram for the ontology is given in Figure 3.2. The main difference to Figure 3.1 is that we removed most of the subclass relationships. Please recall that all our schema diagrams are necessarily ambiguous and incomplete – while they help to understand and use the ontology, it is the set of logical axioms which actually constitutes the ontology.

## 3.1 Axioms

All axioms belonging to the separate modules are part of the overall ontology. In addition, all pairs of classes which are not declared or inferred to be in a subclass relationship, are declared to be disjoint.

## 3.2 Shortcuts

When working with a complex modular ontology, it is often helpful to provide so-called *shortcuts* [Krisnadhi et al., 2016] which simplify population of the ontology or querying of the underlying knowledge graph in cases where the full complexity of the ontology is not required or needed.

Version 1.0 of the Enslaved Ontology carries a few such shortcuts which are convenient for population purposes. They are indicated as dashed red arrows in Figure 3.1.

### 3.2.1 Shortcut for TemporalExtent

One shortcut defines a property temporalExtentContains, which is a concatenation of the hasTemporalContent and contains properties. In terms of formal logic, the shortcut is defined by the following two axioms.

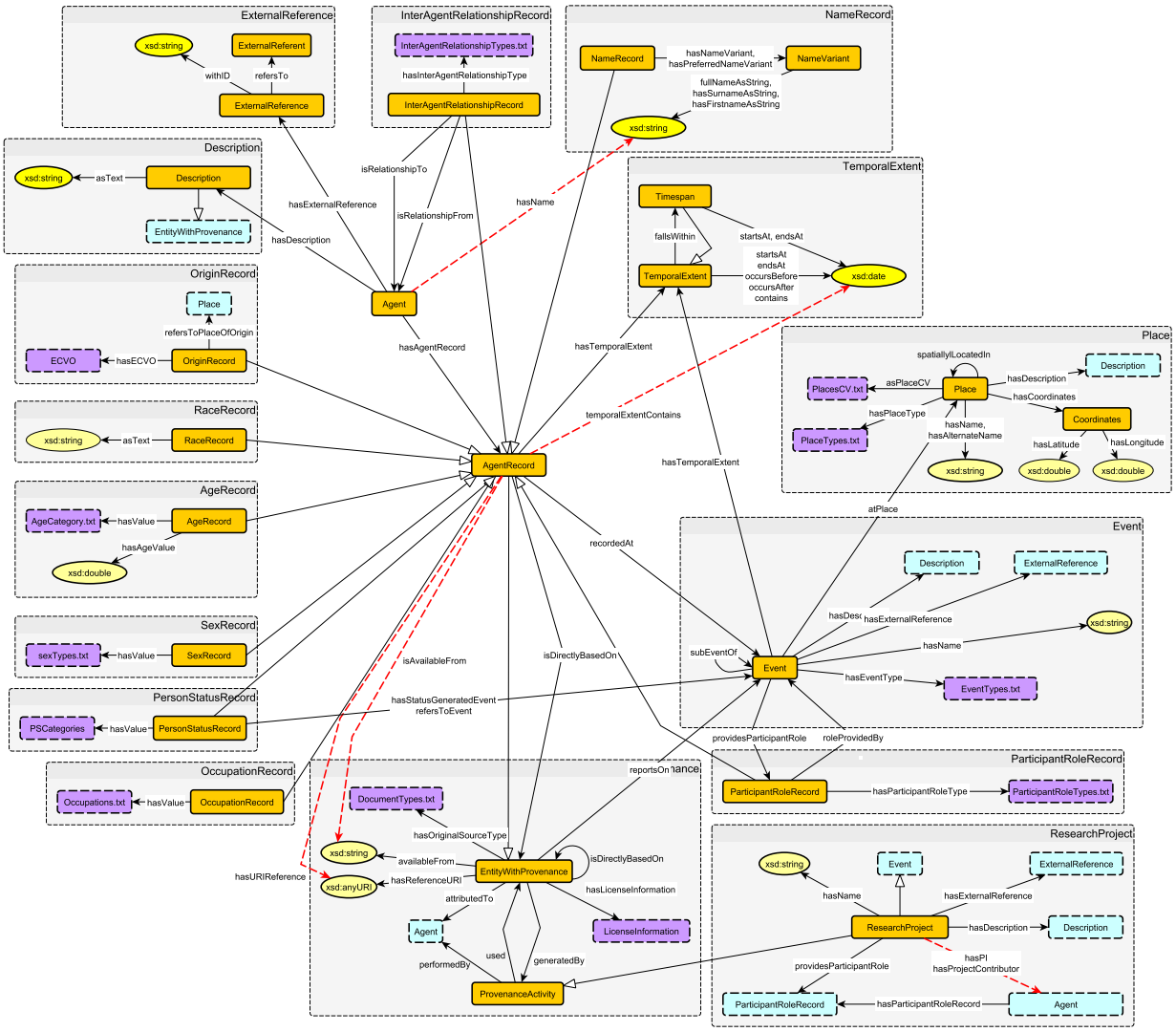


Figure 3.1: Schema Diagram for the Enslaved Ontology.

$$\begin{aligned}\mathsf{AgentRecord}(x) \wedge \mathsf{hasTemporalExtent}(x,y) \wedge \mathsf{TemporalExtent}(y) \wedge \mathsf{contains}(y,z)\\ \wedge\, \mathsf{PointInTime}(z) \rightarrow \mathsf{temporalExtentContains}(x,z)\end{aligned}$$

$$\begin{aligned}\mathsf{temporalExtentContains}(x,z) \rightarrow \exists y(\mathsf{AgentRecord}(x) \wedge \mathsf{hasTemporalExtent}(x,y)\\ \wedge\, \mathsf{TemporalExtent}(y) \wedge \mathsf{contains}(y,z) \wedge \mathsf{PointInTime}(z))\end{aligned}$$

Regretfully, neither of these axioms can be expressed in OWL DL. For the first axiom, the reason is that property chains in OWL DL can only involve object properties. If it were allowed that the last property of a chain could be a datatype property, then the axioms in Figure 3.3 would express it. It seems to us (though we did not do a formal investigation into it), that allowing property chains which carry a datatype property as the last element of the chain should cause no

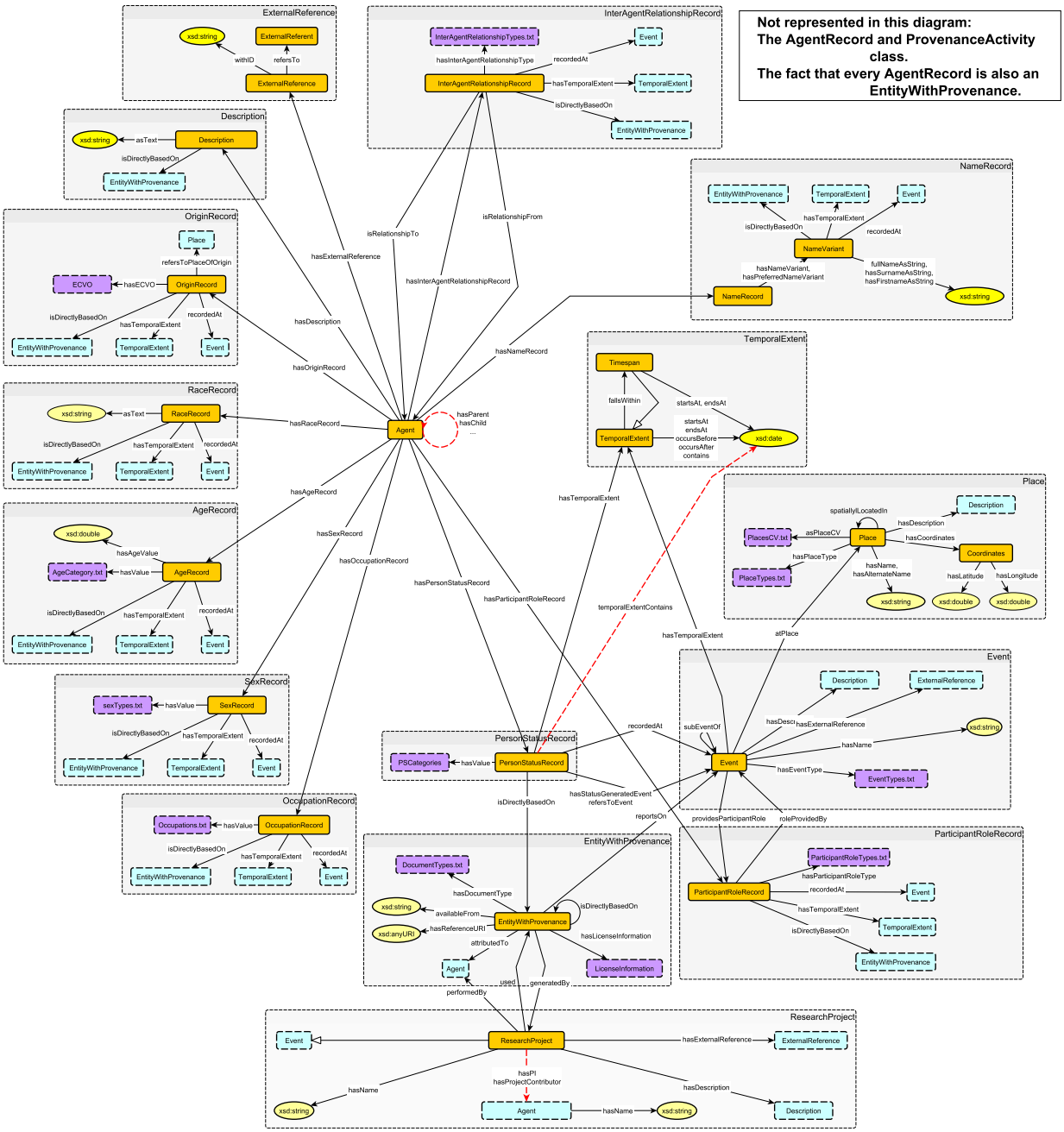


Figure 3.2: Alternative, Partial, Schema Diagram for the Enslaved Ontology, including indication of some shortcuts.

problems whatsoever with decidabilty, complexity, or even reasoning algorithms; alas the W3C standard does not allow it.

The second axiom cannot be fully expressed in OWL DL even if we allow for such property chains. Please see [Krisnadhi et al., 2015b] for a detailed discussion of these matters.

$$\begin{aligned}
\mathsf{AgentRecord} &\equiv \exists \mathsf{agentRecord.Self} \\
\mathsf{TemporalExtent} &\equiv \exists \mathsf{temporalExtent.Self} \\
\mathsf{PointInTime} &\equiv \exists \mathsf{pointInTime.Self} \\
\mathsf{agentRecord} \circ \mathsf{hasTemporalExtent} \circ \mathsf{temporalExtent} \circ \mathsf{contains} & \\
\circ \mathsf{pointInTime} &\sqsubseteq \mathsf{temporalExtentContains}
\end{aligned}$$

Figure 3.3: Partial axioms for the temporalExtentContains shortcut.

### 3.2.2 Shortcut for NameRecord

Another shortcut defines a property hasName, which is a concatenation of the hasNameRecord, hasNameVariant and fullNameAsString properties. In terms of formal logic, the shortcut is defined by the following two axioms.

$$\begin{aligned}
\mathsf{hasNameRecord}(x,y) \wedge \mathsf{hasNameVariant}(y,z) & \\
\wedge \mathsf{fullNameAsString}(z,w) &\rightarrow \mathsf{hasName}(x,w) \\
\mathsf{hasName}(x,w) \rightarrow \exists y \exists z (\mathsf{hasNameRecord}(x,y) &\wedge \mathsf{hasNameVariant}(y,z) \\
\wedge \mathsf{fullNameAsString}(z,w)) &
\end{aligned}$$

As before the first of these is a rule which could be expressed in OWL DL if OWL DL would allow property chains with a datatype property as last element in the chain:

$$\mathsf{hasNameRecord} \circ \mathsf{hasNameVariant} \circ \mathsf{fullNameAsString} \sqsubseteq \mathsf{hasName}.$$

The second cannot be fully expressed in OWL DL even under this accomodation. Please see [Krisnadhi et al., 2015b] for a detailed discussion of these matters.

The property hasName also occurs elsewhere in the ontology, and all these cases can be considered shortcuts. We used them in cases where for the application of the current version of the ontology the shortcut completely suffices.

### 3.2.3 Shortcuts for ResearchProject

Two further shortcuts define the properties hasPI and hasProjectContributor as concatenations of providesParticipantRole and hasParticipantRoleRecord$^-$, where the corresponding ParticipantRoleRecord is ResearchProjectPIRole, respectively, ResearchProjectContributorRole.

In terms of formal logic, these shortcuts are defined by the following two axioms.

$$\begin{aligned}
\mathsf{ResearchProject}(x) \wedge \mathsf{providesParticipantRole}(x,y) \wedge \mathsf{ResearchProjectPIRole}(y) & \\
\wedge \mathsf{hasParticipantRoleRecord}(z,y) \wedge \mathsf{Agent}(z) &\rightarrow \mathsf{hasPI}(x,z) \\
\mathsf{ResearchProject}(x) \wedge \mathsf{providesParticipantRole}(x,y) \wedge \mathsf{ResearchProjectContributorRole}(y) & \\
\wedge \mathsf{hasParticipantRoleRecord}(z,y) \wedge \mathsf{Agent}(z) & \\
\rightarrow \mathsf{hasProjectContributor}(x,z) &
\end{aligned}$$

$$\begin{aligned}
\mathsf{ResearchProject} &\equiv \exists \mathsf{researchProject.Self} \\
\mathsf{ResearchProjectPIRole} &\equiv \exists \mathsf{researchProjectPIRole.Self} \\
\mathsf{ResearchProjectContributorRole} &\equiv \exists \mathsf{researchProjectContributorRole.Self} \\
\mathsf{Agent} &\equiv \exists \mathsf{agent.Self} \\
\mathsf{researchProject} \circ \mathsf{providesParticipantRole} &\circ \mathsf{researchProjectPIRole} \\
&\circ \mathsf{hasParticipantRoleRecord} \circ \mathsf{agent} \sqsubseteq \mathsf{hasPI} \\
\mathsf{Agent} &\equiv \exists \mathsf{agent.Self} \\
\mathsf{researchProject} \circ \mathsf{providesParticipantRole} &\circ \mathsf{researchProjectContributorRole} \\
&\circ \mathsf{hasParticipantRoleRecord} \circ \mathsf{agent} \sqsubseteq \mathsf{hasResearchContributor}
\end{aligned}$$

Figure 3.4: Partial axioms for the research project shortcuts.

$$\begin{aligned}
\mathsf{hasPI}(x,z) \rightarrow \exists y(&\mathsf{ResearchProject}(x) \wedge \mathsf{providesParticipantRole}(x,y) \\
&\wedge \mathsf{ResearchProjectPIRole}(y) \wedge \mathsf{hasParticipantRoleRecord}(z,y) \\
&\wedge \mathsf{Agent}(z)) \\
\mathsf{hasProjectContributor}(x,z) \rightarrow \exists y(&\mathsf{ResearchProject}(x) \wedge \mathsf{providesParticipantRole}(x,y) \\
&\wedge \mathsf{ResearchProjectContributorRole}(y) \wedge \mathsf{hasParticipantRoleRecord}(z,y) \\
&\wedge \mathsf{Agent}(z))
\end{aligned}$$

The first two of these are rules can be expressed in OWL DL using the axioms given in Figure 3.4. The latter two cannot be fully expressed in OWL DL. Please see [Krisnadhi et al., 2015b] for a detailed discussion of these matters. In the ontology, we include only the axioms given in Figure 3.4.

### 3.2.4 ShortCuts for InterAgentRelationshipRecord

We also add shortcuts for the InterAgentRelationshipTypes defined in 2.13.1. Concretely, for each InterAgentRelationshipType $R$ we define the shortcut has$R$ using the following two axioms.

$$\begin{aligned}
\mathsf{InterAgentRelationshipRecord}(y) \wedge \mathsf{hasInterAgentRelationshipType}(y,R) &\\
\wedge \mathsf{IsRelationshipFrom}(y,x) \wedge \mathsf{IsRelationshipTo}(y,z) &\rightarrow \mathsf{has}R(x,z) \\
R(x,z) \rightarrow \exists y(\mathsf{InterAgentRelationshipRecord}(y) \wedge \mathsf{hasInterAgentRelationshipType}(y,R) &\\
\wedge \mathsf{IsRelationshipFrom}(y,x) \wedge \mathsf{IsRelationshipTo}(y,z)) &
\end{aligned}$$

However, similar to the case discussed above, only the first of these is expressible in OWL DL, using the axioms provided in Figure 3.5.

I.e., we have now defined the properties hasParent, hasChild, hasGrandparent, hasGrandChild, hasSibling, hasSpouse, hasMotherInLaw, hasFatherInLaw, hasDaughterInLaw, hasSonInLaw, hasCousin, hasAunt, hasUncle, hasGodmother, hasGodfather, hasOwner.

### 3.2.5 ShortCuts for EntityWithProvenance

We also add shortcuts for EntityWithProvenance, as follows.

$$\begin{aligned}\text{InterAgentRelationshipRecord} \sqcap \exists \text{hasInterAgentRelationshipType}.\{R\} &\\ \sqsubseteq \exists \text{interAgentRelationshipRecord}R.\text{Self} &\\ \text{IsRelationshipFrom}^{-} \circ \text{interAgentRelationshipRecord}R\text{IsRelationshipTo} &\sqsubseteq \text{has}R\end{aligned}$$

Figure 3.5: Partial axioms for the inter agent relationships shortcuts.

$$\begin{aligned}\text{AgentRecord} &\equiv \exists \text{agentRecord}.\text{Self}\\ \text{EntityWithProvenance} &\equiv \exists \text{entityWithProvenance}.\text{Self}\\ \text{agentRecord} \circ \text{isDirectlyBasedOn} \circ \text{availableFrom} &\sqsubseteq \text{isAvailableFrom}\\ \text{agentRecord} \circ \text{isDirectlyBasedOn} \circ \text{hasReferenceURI} &\sqsubseteq \text{hasURIReference}\end{aligned}$$

Figure 3.6: Partial axiomatization of the shortcuts for EntityWithProvenance

$$\begin{aligned}\text{AgentRecord}(x) \wedge \text{isDirectlyBasedOn}(x,y) \wedge \text{EntityWithProvenance}(y)&\\ \wedge \text{availableFrom}(y,z) \rightarrow \text{isAvailableFrom}(x,z)&\\ \text{isAvailableFrom}(x,z) \rightarrow \exists y(\text{AgentRecord}(x) \wedge \text{isDirectlyBasedOn}(x,y)&\\ \wedge \text{EntityWithProvenance}(y) \wedge \text{availableFrom}(y,z))&\\ \text{AgentRecord}(x) \wedge \text{isDirectlyBasedOn}(x,y) \wedge \text{EntityWithProvenance}(y)&\\ \wedge \text{hasReferenceURI}(y,z) \rightarrow \text{hasURIReference}(x,z)&\\ \text{hasURIReference}(x,z) \rightarrow \exists y(\text{AgentRecord}(x) \wedge \text{isDirectlyBasedOn}(x,y)&\\ \wedge \text{EntityWithProvenance}(y) \wedge \text{hasReferenceURI}(y,z))&\end{aligned}$$

For the same reasons as before, we can only translate the first and third of these into OWL DL under the provision that property chains ending in a datatype property were allowed. The corresponding axiomatization can be found in Figure 3.6.

## 3.3 Serialization, including OPLa

We use http://enslaved.org/ontology/1.0/ as the namespace for the Enslaved Ontology, and the ontology will be made available on `http://enslaved.org`. Contolled vocabularies will also be made available on the same site. The structure of the Enslave ontology is as follows.

1. Main ontology file: contains all the axioms constituting the ontology, sans shortcut definitions, as well as OPLa annotations that describe the modular structure of the ontology.
2. Controlled Vocabularies
3. Axioms for CV per version: contains axioms per version of ontology and controlled vocabularies for indicating specific interactions between elements in the vocabularies and classes (e.g. EventTypes might only provide specific ParticipantRoles).

4. Shortcuts: contains axioms (some of which are not specifiable in OWL) that facilitate population tasks.

# Bibliography


[Hitzler et al., 2016] Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., and Presutti, V., editors (2016). *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, volume 25 of *Studies on the Semantic Web*. IOS Press.

[Hitzler et al., 2017a] Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A. A., and Presutti, V. (2017a). Towards a simple but useful ontology design pattern representation language. In Blomqvist, E., Corcho, Ó., Horridge, M., Carral, D., and Hoekstra, R., editors, *Proceedings of the 8th Workshop on Ontology Design and Patterns (WOP 2017) co-located with the 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21, 2017.*, volume 2043 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Hitzler et al., 2017b] Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A. A., and Presutti, V. (2017b). Towards a simple but useful ontology design pattern representation language. In Blomqvist, E., Corcho, Ó., Horridge, M., Carral, D., and Hoekstra, R., editors, *Proceedings of the 8th Workshop on Ontology Design and Patterns (WOP 2017) co-located with the 16th International Semantic Web Conference (ISWC 2017), Vienna, Austria, October 21, 2017.*, volume 2043 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Hitzler and Krisnadhi, 2016] Hitzler, P. and Krisnadhi, A. (2016). On the roles of logical axiomatizations for ontologies. In Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., and Presutti, V., editors, *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, volume 25 of *Studies on the Semantic Web*, pages 73–80. IOS Press.

[Hitzler et al., 2012] Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P., and Rudolph, S., editors (11 December 2012). *OWL 2 Web Ontology Language: Primer (Second Edition)*. W3C Recommendation. Available at http://www.w3.org/TR/owl2-primer/.

[Hitzler et al., 2010] Hitzler, P., Krötzsch, M., and Rudolph, S. (2010). *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC.

[Karima et al., 2017] Karima, N., Hammar, K., and Hitzler, P. (2017). How to document ontology design patterns. In Hammar, K., Hitlzer, P., Lawrynowicz, A., Krisnadhi, A., Nuzzolese, A., and Solanki, M., editors, *Advances in Ontology Design and Patterns*, volume 32 of *Studies on the Semantic Web*, pages 15–28. IOS Press / AKA Verlag.

[Krisnadhi, 2016] Krisnadhi, A. (2016). The role patterns. In Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., and Presutti, V., editors, *Ontology Engineering with Ontology Design Patterns – Foundations and Applications*, volume 25 of *Studies on the Semantic Web*, pages 313–319. IOS Press.

[Krisnadhi and Hitzler, 2016] Krisnadhi, A. and Hitzler, P. (2016). Modeling with ontology design patterns: Chess games as a worked example. In Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., and Presutti, V., editors, *Ontology Engineering with Ontology Design Patterns – Foundations and Applications*, volume 25 of *Studies on the Semantic Web*, pages 3–21. IOS Press.

[Krisnadhi and Hitzler, 2017a] Krisnadhi, A. and Hitzler, P. (2017a). A core pattern for events. In Hammar, K., Hitlzer, P., Lawrynowicz, A., Krisnadhi, A., Nuzzolese, A., and Solanki, M., editors, *Advances in Ontology Design and Patterns*, volume 32 of *Studies on the Semantic Web*, pages 29–38. IOS Press / AKA Verlag.

[Krisnadhi and Hitzler, 2017b] Krisnadhi, A. and Hitzler, P. (2017b). The stub metapattern. In Hammar, K., Hitlzer, P., Lawrynowicz, A., Krisnadhi, A., Nuzzolese, A., and Solanki, M., editors, *Advances in Ontology Design and Patterns*, volume 32 of *Studies on the Semantic Web*, pages 39–64. IOS Press / AKA Verlag.

[Krisnadhi et al., 2015a] Krisnadhi, A., Hu, Y., Janowicz, K., Hitzler, P., Arko, R. A., Carbotte, S., Chandler, C., Cheatham, M., Fils, D., Finin, T. W., Ji, P., Jones, M. B., Karima, N., Lehnert, K. A., Mickle, A., Narock, T. W., O'Brien, M., Raymond, L., Shepherd, A., Schildhauer, M., and Wiebe, P. (2015a). The geolink modular oceanography ontology. In Arenas, M., Corcho, Ó., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P. T., Dumontier, M., Heflin, J., Thirunarayan, K., and Staab, S., editors, *The Semantic Web – ISWC 2015 – 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II*, volume 9367 of *Lecture Notes in Computer Science*, pages 301–309. Springer.

[Krisnadhi et al., 2016] Krisnadhi, A., Karima, N., Hitzler, P., Amini, R., Rodríguez-Doncel, V., and Janowicz, K. (2016). Ontology design patterns for linked data publishing. In Hitzler, P., Gangemi, A., Janowicz, K., Krisnadhi, A., and Presutti, V., editors, *Ontology Engineering with Ontology Design Patterns – Foundations and Applications*, volume 25 of *Studies on the Semantic Web*, pages 201–232. IOS Press.

[Krisnadhi et al., 2011] Krisnadhi, A., Maier, F., and Hitzler, P. (2011). OWL and Rules. In Polleres, A., d'Amato, C., Arenas, M., Handschuh, S., Kroner, P., Ossowski, S., and Patel-Schneider, P. F., editors, *Reasoning Web. Semantic Technologies for the Web of Data – 7th International Summer School 2011, Galway, Ireland, August 23-27, 2011, Tutorial Lectures*, volume 6848 of *Lecture Notes in Computer Science*, pages 382–415. Springer, Heidelberg.

[Krisnadhi et al., 2015b] Krisnadhi, A. A., Hitzler, P., and Janowicz, K. (2015b). On the capabilities and limitations of OWL regarding typecasting and ontology design pattern views. In Tamma, V. A. M., Dragoni, M., Gonçalves, R. S., and Lawrynowicz, A., editors, *Ontology Engineering – 12th International Experiences and Directions Workshop on OWL, OWLED 2015, co-located with ISWC 2015, Bethlehem, PA, USA, October 9-10, 2015, Revised Selected Papers*, volume 9557 of *Lecture Notes in Computer Science*, pages 105–116. Springer.

[Lebo et al., 2013] Lebo, T., Sahoo, S., and McGuinness, D., editors (30 April 2013). *PROV-O: The PROV Ontology*. W3C Recommendation. Available at http://www.w3.org/TR/prov-o/.

[Motik et al., 2012] Motik, B., Patel-Schneider, P., and Parsia, B., editors (11 December 2012). *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (Second Edition)*. W3C Recommendation. Available at http://www.w3.org/TR/owl2-syntax/.

[Sarker et al., 2016] Sarker, M. K., Krisnadhi, A. A., and Hitzler, P. (2016). OWLAx: A Protégé plugin to support ontology axiomatization through diagramming. In Kawamura, T. and Paulheim, H., editors, *Proceedings of the ISWC 2016 Posters & Demonstrations Track co-located with 15th International Semantic Web Conference (ISWC 2016), Kobe, Japan, October 19, 2016.*, volume 1690 of *CEUR Workshop Proceedings*. CEUR-WS.org.

[Shimizu, 2017] Shimizu, C. (2017). Rendering OWL in LaTeX for improved readability: Extensions to the OWLAPI. Master's thesis, Department of Computer Science and Engineering, Wright State University, Dayton, Ohio.

[Shimizu et al., 2018] Shimizu, C., Hirt, Q., and Hitzler, P. (2018). A protégé plug-in for annotating OWL ontologies with opla. In Gangemi, A., Gentile, A. L., Nuzzolese, A. G., Rudolph, S., Maleshkova, M., Paulheim, H., Pan, J. Z., and Alam, M., editors, *The Semantic Web: ESWC*

*2018 Satellite Events – ESWC 2018 Satellite Events, Heraklion, Crete, Greece, June 3-7, 2018, Revised Selected Papers*, volume 11155 of *Lecture Notes in Computer Science*, pages 23–27. Springer.