MODL: A Modular Ontology Design Library

Version 1.0

Contributors:
COGAN SHIMIZU — Wright State University
PASCAL HITZLER — Wright State University
QUINN HIRT — Wright State University

Document Date: February 13, 2023

Acknowledgement. Cogan Shimizu acknowledges support by the Dayton Area Graduate Studies Institute (DAGSI). This work was partially supported by the Air Force Office of Scientific Research under award number FA9550-18-1-0386. Chapter **??** is adapted from [**?**].

Contents

List of Figures

1 Introduction

Motivation

The Information Age is an apt description for these modern times; between the World Wide Web and the Internet of Things an unfathomable amount of information is accessible to humans and machines, but the sheer volume and heterogeneity of the data have their drawbacks. Humans have difficulty drawing *meaning* from large amounts of data. Machines can parse the data, but do not *understand* it. Thus, in order to bridge this gap, data would need to be organized in such a way that some critical part of the human conceptualization is preserved. Ontologies are a natural fit for this role, as they may act as a vehicle for the sharing of *understanding* [**?**].

Unfortunately, published ontologies have infrequently lived up to such a promise, hence the recent emphasis on FAIR (Findable, Accessible, Interoperable, and Reusable) data practices [**?**]. More specifically, many ontologies are not interoperable or reusable. This is usually due to incompatible ontological commitments: strong—or very weak—ontological committments lead to an ontology that is really only useful for a specific use-case, or to an ambiguous model that is almost meaningless by itself.

To combat this, we have developed a methodology for developing so-called modular ontologies [**?**]. In particular, we are especially interested in pattern-based modules [**?**]. A modularized ontology is an ontology that individual users can easily adapt to their own use-cases, while still preserving relations with other versions of the ontology; that is, keeping it *interoperable* with other ontologies. Such ontologies may be so adapted due to their "plug-and-play" nature; that is, one module may be swapped out for another developed from the same pattern.

An ontology design pattern is, essentially, a small self-contained ontology that addresses a general problem that has been observed to be invariant over different domains or applications [**?**]. By tailoring a pattern to a more specific use-case, an ontology engineer has developed a *module*. This modelling paradigm moves much of the cost away from the formalization of a conceptualization (i.e. the logical axiomatization). Instead, pattern-based modular ontolody design (PBMOD) is predicated upon knowledge of available patterns, as well as being aware of the use-cases it addresses and its ontological commitments.

Thus, in order to address the findability and accessibility aspects of PBMOD, we have developed MODL: a modular ontology design library, which is herein described.

Overview

MODL is a curated collection of well-documented ontology design patterns. Some of the patterns are novel, but many more have been extracted from existing ontologies and streamlined for use in a general manner. MODL, as an artefact, is distributed online as a collection of annotated OWL files and this technical report.

There are two different ways to use MODL—for use in ontology modelling and for use in tools. In both cases, MODL is distributed as a ZIP archive of the patterns' OWL files and accompanying documentation. In the case of the Ontology Engineer, it is simply used as a resource while building an ontology, perhaps by using Modular Ontology Modelling or eXtreme Design methodologies. For the tool developer, we also supply an ontology consisting of exactly the OPLa annotations from each pattern that pertain to OntologicalCollection. As OPLa is fully specified in OWL, these annotations make up an ontology of patterns and their relations. One particular use-case that we foresee is a tool developer querying the ontology for which patterns are related to the current pattern, or looking for a pattern based on keywords or similarity to competency questions.

Organization

Namespaces

For MODL we currently use the namespace `https://archive.org/services/purl/purl/modular_ontology_design_library/<VERSION>/<PATTERN>`.

Current Patterns

1. Metapatterns
   a) Explicit Typing
   b) Property Reification
   c) Stubs
2. Organization of Data
   a) Aggregation, Bag, Collection
   b) Sequence, List
   c) Tree
3. Space, Time, and Movement
   a) Spatiotemporal Extent
   b) Spatial Extent
   c) Temporal Extent
   d) Trajectory
   e) Event
4. Agents and Roles
   a) AgentRole
   b) ParticipantRole
   c) Name Stub
5. Description and Details
   a) Quantities and Units
   b) Partonymy/Meronymy
   c) Provenance
   d) Identifier

Categories

Metapatterns This category contains patterns that can be considered to be "patterns for patterns." In other literature, notably [**?**], they may be called *structural ontology design patterns*, as they are independent of any specific context, i.e. they are content-independent. This is particularly true for the metapattern for property reification, which, while a modelling strategy, is also a workaround for the lack of n-ary relationships in OWL. The other metapatterns address structural design choices frequently encountered when working with domain experts. They present a best practice to non-ontologists for addressing language specific limitations.

Organization of Data This category contains patterns that pertain to how data might be organized. These patterns are necessarily highly abstract, as they are ontological reflections of common data structures in computer science. The pattern for aggregation, bag, or collection is a simple model for connecting many concepts to a single concept. Analogously, for the list and tree pattterns, which aim to capture ordinality and acyclicity, as well. More so than other patterns in this library, these patterns provide an axiomatization as a high-level framework that must be specialized (or modularized) to be truly useful.

Space, Time, and Movement This category contains patterns that model the movement of a thing through a space or spaces and a general event pattern. The semantic trajectory pattern is a more general pattern for modelling the discrete movements along some dimensions. The spatiotemporal extent pattern is a trajectory along the familiar dimensions of time and space. Both patterns are included for convenience.

Agents and Roles This category contains patterns that pertain to agents interacting with things. Here, we consider an agent to be anything that performs some action or role. This is important, as it decouples the role of an agent from the agent itself. For example, a Person may be Husband and Widower at some point, but should not be both simultaneously. These patterns enable the capture of this data. In fact, the agent role and participante role patterns are convenient specializations of property reification that have evolved into a modelling practice writ large. In this category, we also include the name stub, which is a convenient instantiation of the stub metapattern; it allows us to acknowledge that a name is a complicated thing, but sometimes we only really need the string representation.

Description and Details This category contains patterns that model the description of things. These patterns are relatively straightforward, models for capturing "how much?" and "what kind?" for a particular thing; patterns that are derived from Winston's part-whole taxonomy [**?**]; a pattern extracted from PROV-O [**?**], perhaps to be used to answer "where did this data come from?"; and a pattern for associating an identifier with something.

2 Preliminaries

We list the individual patterns contained in MODL, together with their axioms and explanations thereof. Schema diagrams are provided throughout, but the reader should keep in mind that while schema diagrams are very useful for understanding an ontology [**?**], 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 [**?**]. 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 [**?**]; 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.

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 **??**, we check all of the following axioms whether they should be included.[2] We list them in natural language, see Figure **??** for the formal versions in description logic notation, and Figure **??** 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.
5. The range of R is B.

[1]Preliminary results supporting this claim can be found in [**?**].

[2]The OWLAx Protégé plug-in [**?**] provides a convenient interface for adding these axioms.

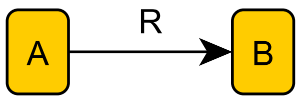

Figure 2.1: Generic node-edge-node schema diagram for explaining systematic axiomatization

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.

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 [?] 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
arrow with white head and no label: a subClass relationship

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 solid tip and label: a relationship (or property) other than a subClass relationship

3 Patterns

3.1 Explicit Typing

Figure 3.1: Schema Diagram for the Explicit Typing Pattern. The visual notation is explained in Chapter **??**.

3.1.1 Summary

The pattern for explicit typing is very straightforward. Indeed, it is merely a representation of what we consider to be a "best practice." This pattern is used when there is a finite, but mutable number of types of a thing. We find this easier to maintain than a series of subclass relationships.

3.1.2 Axiomatization

$$\top \sqsubseteq \forall \mathsf{hasType}.\mathsf{Type} \tag{1}$$

3.1.3 Explanations

1. Range: the range of $\mathsf{hasType}$ is Type.

3.1.4 Competency Questions

CQ1. What is the type of Event?
CQ2. Which type of apparatus is that?

3.2 Property Reification

Figure 3.2: Schema Diagram for Property Reification. The visual notation is explained in Chapter **??**. Additioanlly, we use the dotted line with solid arrow to indicate which property is being reified. This relation has no bearing on the below axioms.

3.2.1 Summary

In OWL, unfortunately, it is not possible to directly represent n-ary relationships. However, it is still possible to capture that information. This notion is called reification. The Property Reification pattern is essentially a metapattern; it could be better considered to be a modelling practice. Here, though, we present a set of axioms that will allow a developer to quickly reify a concept by specializing the framework.

Consider that we would like to relate two Things together via some propertyToBeReified given some Context information that also needs to be captured. To do so, we create a ReifiedProperty and attach the information to this concept. A more concrete example of this can be seen in the AgentRole and ParticipantRole patterns (Sections **??** and **??**).

The axioms below are minimalistic, because it is hard to make claims about the domain and range at the most general case. It should be safe to say that there is certain some connection between the first object of interest and the reified property itself. But, perhaps, the second reified property is reused from some other pattern or part of the ontology—we cannot make any statements about it at this level. Furthermore, concept of "context" is loose and open to interpretation by the developer. Could it be subclassed during specialization or use of this pattern, perhaps? Is it necessary, perhaps not. It does, however, suffice to show *how* reification with context works.

3.2.2 Axiomatization

$$\top \sqsubseteq \forall \mathsf{reifiedProperty1}.\mathsf{ReifiedProperty} \tag{1}$$
$$\top \sqsubseteq \forall \mathsf{hasContext}.\mathsf{Context} \tag{2}$$
$$\top \sqsubseteq \exists \mathsf{hasContext}.\mathsf{Context} \tag{3}$$
$$\tag{4}$$

3.2.3 Explanations

1. Range: the range of reifiedProperty1 is ReifiedProperty.
2. Range: the range of hasContext is Context.
3. Existential: a ReifiedProperty should have at least some contextual information, otherwise, it wouldn't need to be reified.

3.2.4 Competency Questions

CQ1. What was the street named during the Great Depression?
CQ2. From what years was Al Gore Vice President?
CQ3. What is the unit of measurement was used to weigh the elephant?

3.3 Stubs

Figure 3.3: Schema Diagram for Stubs. The visual notation is explained in Chapter **??**.

3.3.1 Summary

Stubs are a very minimal pattern that could also be described as a technique or best practice. Essentially, during modelling, there are frequently times when developers recognize that a concept is complex, but also out of the scope of an ontology or knowledge graph. However, the developer would like to keep the ontology extensible or allow others to build off of the ontology at that point. One example of this is Name or Title. In many cases, there is no reason to store more than a string for a name or title. However, names and titles are not necessarily inherent to a thing at all times. Yet, delving into those details may be unnecessary for a use-case. To account for this, a developer may want to use as stub. That is, acknowledge the complexity of a concept, but also include the information that is useful. This metapattern is described in more detail in [**?**].

3.3.2 Axiomatization

$$\top \sqsubseteq \forall \mathsf{hasValue}.\mathsf{xsd{:}AnyValue} // \mathsf{Stub} \qquad \sqsubseteq \exists \mathsf{hasValue}.\mathsf{xsd{:}AnyValue} \tag{1}$$

3.3.3 Explanations

1. Range: the range of hasValue is any xsd datatype. We use AnyValue in the above axiom to indicate that any datatype will suffice.
2. Existential: the Stub must have a value.

3.3.4 Competency Questions

CQ1. Which street is that?
CQ2. What is the title of Alfred Tennyson?

3.4 Aggregation, Bag, Collection

Figure 3.4: Schema Diagram for the Aggregation, Bag, Collection Pattern. The visual notation is explained in Chapter **??**.

3.4.1 Summary

The pattern for an Aggregation, Bag, or Collection is relatively simple. The Bag is a type of unordered collection. This pattern was included in this library for demonstrating a more approachable interface for the partonymy pattern, with respect to membership. For example, we may use this pattern to represent a committee. In this case, the committee member is theBagItem, the committee is the Bag, and the itemOf property is a sub-property to the po-member property found in the Partonymy/Meronymy pattern (Section **??**). This pattern was adapted from the Bag ontology design pattern and be found at `http://ontologydesignpatterns.org/wiki/Submissions:Bag`. Some language is borrowed from the description.

3.4.2 Axiomatization

$$
\begin{align}
\mathsf{Bag} &\sqsubseteq \mathsf{Collection} \tag{1}\\
\mathsf{itemOf} &\sqsubseteq \mathsf{po\text{-}member} \tag{2}\\
\exists \mathsf{itemOf}.\mathsf{BagItem} &\sqsubseteq \mathsf{Bag} \tag{3}\\
\mathsf{Bag} &\sqsubseteq \forall \mathsf{itemOf}.\mathsf{BagItem} \tag{4}\\
& \tag{5}
\end{align}
$$

3.4.3 Explanations

1. Subclass: a Bag is a Collection
2. Subproperty: itemOf is a subproperty to po-member from the Partonymy Pattern (Section **??**).
3. Scoped Domain: the scoped domain of itemOf, scoped by BagItem, is Bag.
4. Scoped Range: the scoped range of itemOf, scoped by Bag, is BagItem.

3.4.4 Competency Questions

CQ1. What bag is this item an element of?
CQ2. What resource does this item refer to?
CQ3. What are the items contained in this bag?

3.5 Sequence, List

Figure 3.5: Schema Diagram for the Sequence and List Pattern. The visual notation is explained in Chapter ??.

3.5.1 Summary

The Sequence Pattern is a way of imposing order upon items of interest; it follows the conceptualization of a Linked List from computer science. This pattern is a simplified view of the Tree Pattern (as found in Section ??) and is adapted from [?]. While this pattern seems very abstract, it is both easy to specialize and occurs very frequently. In this resource, the pattern occurs in the Trajectory Pattern (a sequence of Fixes), the SpatiotemporalExtent Pattern (a sequence of Place, Time pairs), and SpatialExtent (a sequence of PointsInSpace.

3.5.2 Axiomatization

$$\mathsf{FirstItem} \sqsubseteq \mathsf{ListItem} \quad (1)$$
$$\mathsf{LastItem} \sqsubseteq \mathsf{ListItem} \quad (2)$$
$$\mathsf{ListItem} \sqsubseteq \forall \mathsf{hasNext}.\mathsf{ListItem} \quad (3)$$
$$\mathsf{ListItem} \sqsubseteq \forall \mathsf{hasNext}^{-}.\mathsf{ListItem} \quad (4)$$
$$\mathsf{ListItem} \sqcap \neg \mathsf{LastItem} \equiv \mathsf{ListItem} \sqcap {=}1 \mathsf{hasNext}.\mathsf{ListItem} \quad (5)$$
$$\mathsf{ListItem} \sqcap \neg \mathsf{FirstItem} \equiv \mathsf{ListItem} \sqcap {=}1 \mathsf{hasNext}^{-}.\mathsf{ListItem} \quad (6)$$
$$\mathsf{FirstItem} \equiv \mathsf{ListItem} \sqcap \neg \exists \mathsf{hasNext}^{-}.\top \quad (7)$$
$$\mathsf{LastItem} \equiv \mathsf{ListItem} \sqcap \neg \exists \mathsf{hasNext}.\top \quad (8)$$
$$\mathsf{hasNext} \sqsubseteq \mathsf{hasSuccessor} \quad (9)$$
$$\mathsf{hasNext} \circ \mathsf{hasSuccessor} \sqsubseteq \mathsf{hasSuccessor} \quad (10)$$
$$\mathrm{Irreflexive}(\mathsf{hasSuccessor}) \quad (11)$$

3.5.3 Explanations

1. Subclass: the FirstItem is a ListItem.
2. Subclass: the LastItem is a ListItem.
3. Scoped Range: the range of hasNext, scoped by ListItem, is ListItem.
4. Scoped Range: the range of $\mathsf{hasNext}^{-}$, scoped by ListItem, is ListItem.
5. A ListItem that is not the LastItem has exactly one next ListItem.
6. A ListItem that is not the FirstItem has exactly one previous ListItem.
7. The FirstItem does not have have a predecessor.
8. The LastItem does not have a next ListItem.
9. Subproperty: hasNext is a subproperty to hasSuccessor.
10. Role Chain: the successor of a ListItem's next ListItem is its successor.
11. Irreflexivity.

3.5.4 Competency Questions

CQ1. What is the first element of the list?
CQ2. What is the last element of the list?
CQ3. Is x a predecessor of y?

3.6 Tree

Figure 3.6: Schema Diagram for the Tree Pattern. The visual notation is explained in Chapter **??**.

3.6.1 Summary

The Tree pattern allows a developer to organize data into a tree data structure. An ontological tree, however, is subtly different from those that occur in other parts of computer science; these trees should be viewed as static—something to be queried, not manipulated. For example, a motivating use case is the organization of organisms into a phylogenetic tree. Such examples and more information may be found in [**?**], from where this pattern is adapted.

3.6.2 Axiomatization

$$\mathsf{LeafNode} \sqsubseteq \mathsf{TreeNode} \tag{1}$$
$$\mathsf{RootNode} \sqsubseteq \mathsf{TreeNode} \tag{2}$$
$$\mathsf{TreeNode} \sqsubseteq \forall \mathsf{hasOutDegree.xsd{:}positiveInteger} \tag{3}$$
$$\mathsf{TreeNode} \sqsubseteq =1 \mathsf{hasOutDegree.xsd{:}positiveInteger} \tag{4}$$
$$\mathsf{LeafNode} \equiv \mathsf{TreeNode} \sqcap \forall \mathsf{hasOutDegree}.\{0\text{\^{}\^{}}\mathsf{xsd{:}positiveInteger}\} \tag{5}$$
$$\mathsf{TreeNode} \sqcap \neg\mathsf{LeafNode} \equiv \mathsf{TreeNode} \sqcap \forall \mathsf{hasOutDegree}.\{x\text{\^{}\^{}}\mathsf{xsd{:}positiveInteger} \mid 1 \leq x\} \tag{6}$$
$$\mathsf{hasChild} \equiv \mathsf{hasParent}^{-} \tag{7}$$
$$\mathsf{hasDescendant} \equiv \mathsf{hasAncestor}^{-} \tag{8}$$
$$\mathsf{hasChild} \sqsubseteq \mathsf{hasDescendant} \tag{9}$$
$$\mathsf{hasDescendant} \circ \mathsf{hasDescendant} \sqsubseteq \mathsf{hasDescendant} \tag{10}$$
$$\mathsf{TreeNode} \sqsubseteq \forall \mathsf{hasChild.TreeNode} \tag{11}$$
$$\mathsf{TreeNode} \sqcap \neg\mathsf{LeafNode} \equiv \mathsf{TreeNode} \sqcap \exists \mathsf{hasChild.TreeNode} \tag{12}$$
$$\mathsf{TreeNode} \sqsubseteq \forall \mathsf{hasDescendant.TreeNode} \tag{13}$$
$$\mathsf{TreeNode} \sqsubseteq \forall \mathsf{hasParent.TreeNode} \tag{14}$$
$$\mathsf{TreeNode} \sqsubseteq \forall \mathsf{hasSibling.TreeNode} \tag{15}$$
$$\mathsf{TreeNode} \sqcap \neg\mathsf{RootNode} \equiv \mathsf{TreeNode} \sqcap =1 \mathsf{hasParent}.\top \tag{16}$$
$$\mathsf{TreeNode} \sqsubseteq \forall \mathsf{hasAncestor.TreeNode} \tag{17}$$
$$\mathsf{RootNode} \equiv \mathsf{TreeNode} \sqcap \neg\exists \mathsf{hasParent}.\top \tag{18}$$
$$\mathsf{LeafNode} \equiv \mathsf{TreeNode} \sqcap \neg\exists \mathsf{hasChild}.\top \tag{19}$$
$$\mathrm{Irreflexive}(\mathsf{hasChild}) \tag{20}$$
$$\mathrm{Irreflexive}(\mathsf{hasParent}) \tag{21}$$
$$\mathrm{Irreflexive}(\mathsf{hasDescendant}) \tag{22}$$
$$\mathrm{Irreflexive}(\mathsf{hasAncestor}) \tag{23}$$

$$\mathsf{hasSibling} \equiv \mathsf{hasSibling}^{-} \tag{24}$$
$$\text{Irreflexive}(\mathsf{hasSibling}) \tag{25}$$

3.6.3 Explanations

1. Subclass: every LeafNode is a TreeNode.
2. Subclass: the RootNode is a TreeNode.
3. Scoped Range: the range of hasOutDegree, scoped by TreeNode, is xsd:positiveInteger.
4. Existential: a TreeNode has exactly one hasOutDegree.
5. A LeafNode is a TreeNode that has an out degree of 0.
6. A TreeNode that is not a LeafNode has at least out degree of 1.
7. Inverse Alias
8. Inverse Alias
9. Subproperty: hasChild is subproperty of hasDescendant.
10. Role Chain: hasDescendant is transitive.
11. Scoped Range: the range of hasChild, scoped by TreeNode, is TreeNode.
12. A TreeNode that is not a LeafNode has a child that is a TreeNode.
13. Scoped Range: the range of hasDescendant, scoped by TreeNode, is TreeNode.
14. Scoped Range: the range of hasParent, scoped by TreeNode, is TreeNode.
15. Scoped Range: the range of hasSibling, scoped by TreeNode, is TreeNode.
16. A TreeNode that is not the RootNode has a TreeNode that is its parent.
17. Scoped Range: the range of hasAncestor, scoped by TreeNode, is TreeNode.
18. RootNode does not have a TreeNode that is its parent.
19. LeafNodes do not have TreeNodes that are its children.
20. Irreflexivity
21. Irreflexivity
22. Irreflexivity
23. Irreflexivity
24. Inverse Alias
25. Irreflexivity

3.6.4 Competency Questions

We remark that these competency questions are as general as the pattern. See [**?**] for more information.
CQ1. Determine the root.
CQ2. Determine all ancestors of a given node.
CQ3. Determine all leaves.
CQ4. Determine all descendants of a given node.
CQ5. Determine all descendants of a given node which are leaves.
CQ6. Given two nodes, determine whether one is a descendant of the other.
CQ7. given two nodes, determine all common ancestors.
CQ8. Given two nodes, determine the latest common ancestor.
CQ9. Given two nodes x and y, determine the earliest ancestor of x which not an ancestor of y.

3.7 Spatiotemporal Extent

Figure 3.7: Schema Diagram for the Spatiotemporal Extent Pattern. The visual notation is explained in Chapter **??**.

3.7.1 Summary

The SpatiotemporalExtent pattern wraps the Trajectory Pattern (Section **??**). Essentially, it uses the Trajectory Pattern's ability to capture discrete snapshots of something moving along some dimension, but casts it into the familiar three physical dimensions, plus time. This is done by adding the atPlace and atTime properties that hang off of Fix. This pattern is more fully described in [**?**]. The SpatiotemporalExtent is primarily used when it is difficult to separate space and time when talking about a concept.

3.7.2 Axiomatization

$$\top \sqsubseteq \forall \mathsf{hasSpatiotemporalExtent}.\mathsf{SpatiotemporalExtent} \tag{1}$$
$$\top \sqsubseteq \forall \mathsf{hasTrajectory}.\mathsf{Trajectory} \tag{2}$$
$$\mathsf{SpatiotemporalExtent} \sqsubseteq \exists \mathsf{hasTrajectory}.\mathsf{Trajectory} \tag{3}$$
$$\top \sqsubseteq \forall \mathsf{atPlace}.\mathsf{Place} \tag{4}$$
$$\top \sqsubseteq \forall \mathsf{atTime}.\mathsf{Time} \tag{5}$$
$$\mathsf{Segment} \sqsubseteq =1 \mathsf{startsFrom}.\mathsf{Fix} \tag{6}$$
$$\mathsf{Segment} \sqsubseteq =1 \mathsf{endsAt}.\mathsf{Fix} \tag{7}$$
$$\mathsf{Segment} \sqsubseteq \exists \mathsf{hasSegment}^{-}.\mathsf{Trajectory} \tag{8}$$
$$\mathsf{startsFrom}^{-} \circ \mathsf{endsAt} \sqsubseteq \mathsf{hasNext} \tag{9}$$
$$\mathsf{hasNext} \sqsubseteq \mathsf{hasSuccessor} \tag{10}$$
$$\mathsf{hasSuccessor} \circ \mathsf{hasSucessor} \sqsubseteq \mathsf{hasSucessor} \tag{11}$$
$$\mathsf{hasNext}^{-} \equiv \mathsf{hasPrevious} \tag{12}$$
$$\mathsf{hasSuccessor}^{-} \equiv \mathsf{hasPredecessor} \tag{13}$$
$$\mathsf{Fix} \sqcap \neg \exists \mathsf{endsAt}^{-}.\mathsf{Segment} \sqsubseteq \mathsf{StartingFix} \tag{14}$$
$$\mathsf{Fix} \sqcap \neg \exists \mathsf{startsFrom}^{-}.\mathsf{Segment} \sqsubseteq \mathsf{EndingFix} \tag{15}$$
$$\mathsf{Trajectory} \sqsubseteq \exists \mathsf{hasSegment}.\mathsf{Segment} \tag{16}$$
$$\mathsf{hasSegment} \circ \mathsf{startsFrom} \sqsubseteq \mathsf{hasFix} \tag{17}$$
$$\mathsf{hasSegment} \circ \mathsf{endsAt} \sqsubseteq \mathsf{hasFix} \tag{18}$$
$$\exists \mathsf{hasSegment}.\mathsf{Segment} \sqsubseteq \mathsf{Trajectory} \tag{19}$$
$$\exists \mathsf{hasSegment}^{-}.\mathsf{Trajectory} \sqsubseteq \mathsf{Segment} \tag{20}$$
$$\exists \mathsf{hasFix}.\mathsf{Segment} \sqsubseteq \mathsf{Trajectory} \tag{21}$$
$$\exists \mathsf{hasFix}^{-}.\mathsf{Trajectory} \sqsubseteq \mathsf{Fix} \tag{22}$$

3.7.3 Explanations

1. Range: the range of hasSpatiotemporalExtent is SpatiotemporalExtent.
2. Range: the range of hasTrajectory is hasTrajectory.
3. Existential: a SpatiotemporalExtent has at least one Trajectory.
4. Range: the range of atPlace is Place.
5. Range: the range of atTime is Time.
6. Segment startFrom exactly one Fix.
7. Segment endsAt exactly one Fix.
8. Existential: A Segment belongs to at least one Trajectory.
9. Role Chain: the concatenation of startsFrom$^-$ and endsAt is hasNext.
10. Subproperty: hasNext is a subproperty to hasSuccessor.
11. Role Chain: hasSuccessor is transitive.
12. Inverse Alias.
13. Inverse Alias.
14. A Fix that is not where a segment ends is a StartingFix.
15. A Fix that is not where a segment starts is a EndingFix.
16. Existential: a Trajectory has at least one Segment.
17. Role Chain: the concatenation of hasSegment and startsFrom is hasFix.
18. Role Chain: the concatenation of hasSegment and endsAt is hasFix.
19. Scoped Domain: the domain of hasSegment, scoped by Segment, is Trajectory.
20. Scoped Domain: the domain of hasSegment$^-$, scoped by Trajectory, is Segment.
21. Scoped Domain: the domain of hasFix, scoped by Segment, is Trajectory.
22. Scoped Domain: the domain of hasFix$^-$, scoped by Trajectory, is Fix.

3.7.4 Competency Question

CQ1. Show which birds stop at x and y.
CQ2. Show the trajectories which cross national parks.
CQ3. Show the trajectories of birds which are less than one year old.

3.8 Spatial Extent

Figure 3.8: Schema Diagram for Spatial Extent. The visual notation is explained in Chapter **??**.

3.8.1 Summary

The SpatialExtent pattern is characterized by a set of Interiors, which are in turn characterized by a PointInSpace-Sequence. A PointInSpace-Sequence consists of PointInSpace-SequenceElements, which are constituted by PointInSpace. A PointInSpace is described by a value and a reference system. PIS-Sequence is a specialization of the Sequence Pattern (Section **??**). We also further choose to use the Explicit Typing Pattern for PointInSpace and ReferenceSystem.

3.8.2 Axiomatization

$$\begin{aligned}
\mathsf{SpatialExtent} &\sqsubseteq {=}n\mathsf{contains.Interior} && (1)\\
\mathsf{Interior} &\sqsubseteq {=}1\mathsf{isDefinedBy.PIS\text{-}Sequence} && (2)\\
\mathsf{PIS\text{-}Sequence} &\sqsubseteq {=}1\mathsf{hasFirst.PIS\text{-}SequenceElement} && (3)\\
\mathsf{PIS\text{-}Sequence} &\sqsubseteq {=}1\mathsf{hasLast.PIS\text{-}SequenceElement} && (4)\\
\mathsf{PIS\text{-}SequenceElement} &\sqsubseteq {=}1\mathsf{hasNext.PIS\text{-}SequenceElement} && (5)\\
\mathsf{PIS\text{-}SequenceElement} &\sqsubseteq {=}1\mathsf{constitutedBy.PointInSpace} && (6)\\
\mathsf{PointInSpace} &\sqsubseteq {=}1\mathsf{hasReferenceSystem.ReferenceSystem} && (7)\\
\mathsf{PointInSpace} &\sqsubseteq {=}1\mathsf{hasValue.Value} && (8)
\end{aligned}$$

3.8.3 Explanations

1. Numerical Restriction: a SpatialExtent contains exactly n Interiors. See the following section.
2. Numerical Restriction: a Interior isDefinedBy exactly 1 PIS-Sequence.
3. Numerical Restriction: a PIS-Sequence has exactly 1 first PIS-SequenceElement.
4. Numerical Restriction: a PIS-Sequence has exactly 1 last PIS-SequenceElement.
5. Numerical Restriction: a PIS-SequenceElement has exactly 1 next PIS-SequenceElement.
6. Numerical Restriction: a PIS-SequenceElement isConstitutedBy exactly 1 PointInSpace.
7. Numerical Restriction: a PointInSpace has exactly 1 ReferenceSystem.
8. Numerical Restriction: a PointInSpace has exactly 1 Value.

3.8.4 Remarks

We would also like the pattern to be able to express that a SpatialExtent consists of exactly some Interiors and no others. This is done by equipping the pattern with an axiom that must be tailored to the use-case and two rules for generating a set of assertions.

$$\mathsf{SpatialExtent} \sqsubseteq {=}n\mathsf{contains.Interior} \qquad (9)$$

where n is the number of expected Interiors. Next,

$$\textit{contains}(\textsf{spatialExtent, interior}_k) \text{ for } k = 1, ..., n$$

and

$$\textsf{interior}_i \neq \textsf{interior}_j \text{ for } i \neq j$$

This allows us to express a SpatialExtent as a set of Interiors.

3.8.5 Competency Questions

CQ1. Where was the Battle of Manassas?
CQ2. What path did the moose take to Canada?
CQ3. Where is the largest prairie in the United States?

3.9 Temporal Extent

Figure 3.9: Schema Diagram for Temporal Extent. The visual notation is explained in Chapter **??**.

3.9.1 Summary

A TemporalExtent is composed of a number of ComplexTimeIntervals, which may be intervals of non-zero length (i.e. TimeIntervals) or intervals of length 0 (i.e. PointsInSpace).

3.9.2 Axiomatization

$$\begin{aligned}
\mathsf{TemporalExtent} &\sqsubset {=}n\mathsf{contains.ComplexTemporalExtent} & (1)\\
\mathsf{TimeInterval} &\sqsubset \mathsf{ComplexTemporalExtent} & (2)\\
\mathsf{TimeInterval} &\sqsubset {=}1\mathsf{startsFrom.PointInTime} & (3)\\
\mathsf{TimeInterval} &\sqsubset {=}1\mathsf{endsAt.PointInTime} & (4)\\
\mathsf{PointInTime} &\sqsubset \mathsf{ComplexTemporalExtent} & (5)\\
\mathsf{PointInTime} &\sqsubset {=}1\mathsf{hasReferenceSystem.ReferenceSystem} & (6)\\
\mathsf{PointInTime} &\sqsubset {=}1\mathsf{hasValue.Value} & (7)
\end{aligned}$$

3.9.3 Explanations

1. Numerical Restriction: a TemporalExtent contains exactly n ComplexTemporalExtents. See below remarks.
2. Subclass: every TimeInterval is a ComplexTemporalExtent.
3. Numerical Restriction: a TimeInterval startsAt exactly 1 PointInTime.
4. Numerical Restriction: a TimeInterval endsAt exactly 1 PointInTime.
5. Subclass: every PointInTime is a ComplexTemporalExtent.
6. Numerical Restriction: a PointInTime has exactly 1 ReferenceSystem.
7. Numerical Restriction: a PointInTime has exactly 1 Value.

3.9.4 Remarks

We would also like the pattern to be able to express that a TemporalExtent consists of exactly some TimeIntervals or PointsInTime and no other things. This is done by equipping the pattern with an axiom that must be tailored to the use-case and two rules for generating a set of assertions.

$$\mathsf{TemporalExtent} \sqsubseteq {=}n\mathsf{contains.Interior} \qquad (8)$$

where n is the number of expected ComplextimeIntverals. Next,

$$\mathit{contains}(\mathsf{temporalExtent,\ complexTimeInterval}_k) \text{ for } k = 1, ..., n$$

and

$$\mathsf{complexTimeInterval}_i \neq \mathsf{complexTimeInterval}_j \text{ for } i \neq j$$

This allows us to express a TemporalExtent as a set of ComplexTimeIntervals.

3.9.5 Competency Questions

CQ1. Which dates did World War II span?
CQ2. What era was the ice age?

3.10 Trajectory

Figure 3.10: Schema Diagram for the Trajectory Pattern. The visual notation is explained in Chapter **??**.

3.10.1 Summary

The Trajectory Pattern allows a developer to track something moving through some space. This is, of course, very abstract and is intended to be a starting point for capturing any movement that occurs at discrete points in a space. Intuitively, there is the notion of moving through time and space and those captured discrete points in space may be GPS position recordings. This sort of data may be best captured with the SpatiotemporalExtent Pattern (Section **??**), which extends the Trajectory Pattern. This pattern may be also used as a starting point for modelling procedures (i.e. steps are discrete points in procedure space) or chemical reactions (we can really only be sure of what our sensors tell us, and they only tell us things at their polling rates). This pattern is an abstraction of the Semantic Trajectory pattern found in [**?**].

3.10.2 Axiomatization

$$\mathsf{Segment} \sqsubseteq {=}1\mathsf{startsFrom}.\mathsf{Fix} \tag{1}$$
$$\mathsf{Segment} \sqsubseteq {=}1\mathsf{endsAt}.\mathsf{Fix} \tag{2}$$
$$\mathsf{Segment} \sqsubseteq \exists\mathsf{hasSegment}^{-}.\mathsf{Trajectory} \tag{3}$$
$$\mathsf{startsFrom}^{-} \circ \mathsf{endsAt} \sqsubseteq \mathsf{hasNext} \tag{4}$$
$$\mathsf{hasNext} \sqsubseteq \mathsf{hasSuccessor} \tag{5}$$
$$\mathsf{hasSuccessor} \circ \mathsf{hasSucessor} \sqsubseteq \mathsf{hasSucessor} \tag{6}$$
$$\mathsf{hasNext}^{-} \equiv \mathsf{hasPrevious} \tag{7}$$
$$\mathsf{hasSuccessor}^{-} \equiv \mathsf{hasPredecessor} \tag{8}$$
$$\mathsf{Fix} \sqcap \neg\exists\mathsf{endsAt}^{-}.\mathsf{Segment} \sqsubseteq \mathsf{StartingFix} \tag{9}$$
$$\mathsf{Fix} \sqcap \neg\exists\mathsf{startsFrom}^{-}.\mathsf{Segment} \sqsubseteq \mathsf{EndingFix} \tag{10}$$
$$\mathsf{Trajectory} \sqsubseteq \exists\mathsf{hasSegment}.\mathsf{Segment} \tag{11}$$
$$\mathsf{hasSegment} \circ \mathsf{startsFrom} \sqsubseteq \mathsf{hasFix} \tag{12}$$
$$\mathsf{hasSegment} \circ \mathsf{endsAt} \sqsubseteq \mathsf{hasFix} \tag{13}$$
$$\exists\mathsf{hasSegment}.\mathsf{Segment} \sqsubseteq \mathsf{Trajectory} \tag{14}$$
$$\exists\mathsf{hasSegment}^{-}.\mathsf{Trajectory} \sqsubseteq \mathsf{Segment} \tag{15}$$
$$\exists\mathsf{hasFix}.\mathsf{Segment} \sqsubseteq \mathsf{Trajectory} \tag{16}$$
$$\exists\mathsf{hasFix}^{-}.\mathsf{Trajectory} \sqsubseteq \mathsf{Fix} \tag{17}$$

3.10.3 Explanations

1. Segment startFrom exactly one Fix.

2. Segment endsAt exactly one Fix.
3. Existential: A Segment belongs to at least one Trajectory.
4. Role Chain: the concatenation of startsFrom$^-$ nad endsAt is hasNext.
5. Subproperty: hasNext is a subproperty to hasSuccessor.
6. Role Chain: hasSuccessor is transitive.
7. Inverse Alias.
8. Inverse Alias.
9. A Fix that is not where a segment ends is a StartingFix.
10. A Fix that is not where a segment starts is a EndingFix.
11. Existential: a Trajectory has at least one Segment.
12. Role Chain: the concatenation of hasSegment and startsFrom is hasFix.
13. Role Chain: the concatenation of hasSegment and endsAt is hasFix.
14. Scoped Domain: the domain of hasSegment, scoped by Segment, is Trajectory.
15. Scoped Domain: the domain of hasSegment$^-$, scoped by Trajectory, is Segment.
16. Scoped Domain: the domain of hasFix, scoped by Segment, is Trajectory.
17. Scoped Domain: the domain of hasFix$^-$, scoped by Trajectory, is Fix.

3.10.4 Competency Questions

CQ1. What is the first step of the procedure?
CQ2. What was the cruise's final stop?

3.11 Event

Figure 3.11: Schema Diagram for the Event Pattern. The visual notation is explained in Chapter **??**.

3.11.1 Summary

The purpose of this pattern is to provide a minimalistic model of an event where it is not always possible to separate its spatial and the temporal aspects, thus can model events that move or possess discontinuous temporal extent. Events, according to this model, have at least one participant, attached via a ParticipantRole (Section **??**). A more thorough examination of the pattern and some additional (optional) axioms can be found in [**?**]. Some language is borrowed from `http://ontologydesignpatterns.org/wiki/Submissions:EventCore`.

3.11.2 Axiomatization

$$\mathsf{subEventOf} \circ \mathsf{subEventOf} \sqsubseteq \mathsf{subEventOf} \tag{1}$$

$$\mathsf{Event} \sqsubseteq {=}1\mathsf{hasSpatiotemporalExtent}.\mathsf{SpatiotemporalExtent} \tag{2}$$

$$\mathsf{Event} \sqsubseteq \exists\mathsf{providesParticipantRole}.\mathsf{ParticipantRole} \tag{3}$$

$$\top \sqsubseteq \forall\mathsf{hasSpatiotemporalExtent}.\mathsf{SpatiotemporalExtent} \tag{4}$$

$$\top \sqsubseteq \forall\mathsf{providesParticipantRole}.\mathsf{ParticipantRole} \tag{5}$$

$$\exists\mathsf{subEventOf}.\top \sqsubseteq \mathsf{Event} \tag{6}$$

$$\top \sqsubseteq \forall\mathsf{subEventOf}.\mathsf{Event} \tag{7}$$

3.11.3 Explanations

1. Role Chain: subEventOf is transitive.
2. Event has exactly one SpatiotemporalExtent.
3. Event provides at least one ParticipantRole.
4. Range: the range of hasSpatiotemporalExtent is SpatiotemporalExtent.
5. Range: the range of providesParticipantRole is ParticipantRole.
6. Domain: the domain of subEventOf is Event.
7. Range: the range of subEventOf is Event.

3.11.4 Remarks

It is also possible to equip the pattern with the following rule.

$$\mathsf{Event}(x) \land \mathsf{providesParticipantRole}(x,p) \land \mathsf{subEventOf}(x,y) \to \mathsf{providesParticipantRole}(y,p) \tag{8}$$

This rule can be converted into OWL DL through *rolification* [**?**].This results in the following axioms.

$$\mathsf{Event} \equiv \exists R_{\mathsf{Event}}.\mathsf{Self} \tag{9}$$

$$\mathsf{subEventOf}^{-} \circ R_{\mathsf{Event}} \circ \mathsf{providesParticipantRole} \sqsubseteq \mathsf{providesParticipantRole} \tag{10}$$

3.11.5 Competency Questions

CQ1. Where and when did the 1990 World Chess Championship Match take place?
CQ2. Who were involved in the 1990 World Chess Championship Match?

3.12 AgentRole

Figure 3.12: Schema Diagram for the AgentRole Pattern. The visual notation is explained in Chapter **??**.

3.12.1 Summary

The AgentRole pattern is essentially a reification of association with something. That is, it's very unlikely that an Agent will be associated with something for all time. Thus, the association relation is not binary, perhaps $\textit{associated}(x, y, t)$, agent x is associated with thing y at time t. Thus, the reification. The association becomes a concept in its own right and has a temporal extent, allowing an Agent to be associated to a Thing (e.g. Event, Section **??**) for some TemporalExtent.

3.12.2 Axiomatization

$$\begin{align}
\mathsf{AgentRole} &\sqsubseteq {=}1\mathsf{isPerformedBy.Agent} \tag{1}\\
\mathsf{AgentRole} &\sqsubseteq {=}1\mathsf{hasTemporalExtent.TemporalExtent} \tag{2}\\
\exists\mathsf{isPerformedBy.Agent} &\sqsubseteq \mathsf{AgentRole} \tag{3}\\
\mathsf{AgentRole} &\sqsubseteq \forall\mathsf{isPerformedBy.Agent} \tag{4}\\
\exists\mathsf{hasTemporalExtent.TemporalExtent} &\sqsubseteq \mathsf{AgentRole} \tag{5}\\
\top &\sqsubseteq \forall\mathsf{hasTemporalExtent.TemporalExtent} \tag{6}\\
\top &\sqsubseteq \forall\mathsf{providesAgentRole.AgentRole} \tag{7}\\
\mathsf{isPerformedBy} &\equiv \mathsf{performsAgentRole}^- \tag{8}\\
\mathsf{isProvidedBy} &\equiv \mathsf{providesAgentRole}^- \tag{9}
\end{align}$$

3.12.3 Explanations

1. Exactly one Agent performs an AgentRole.
2. An AgentRole has exactly one TemporalExtent.
3. Scoped Domain: the scoped domain of isPerformedBy, scoped by Agent, is AgentRole.
4. Scoped Range: the scoped range of isPerformedBy, scoped by AgentRole, is Agent.
5. Scoped Domain: the scoped domain of hasTemporalExtent, scoped by TemporalExtent, is AgentRole.
6. Range: the range of hasTemporalExtent is TemporalExtent.
7. Range: the range of providesAgentRole is AgentRole.
8. Inverse Alias.
9. Inverse Alias.

3.12.4 Competency Questions

CQ1. When was Cogan Shimizu a student at Wright State University?
CQ2. Who was the lead actor for the movie, Sharknado?
CQ3. Who was on the World Cup winning team in 2017?

3.13 ParticipantRole

Figure 3.13: Schema Diagram for the ParticipantRole Pattern. The visual notation is explained in Chapter **??**.

3.13.1 Summary

The ParticipantRole Pattern is a specialization of the AgentRole Pattern, which can be found in Section **??**; many axioms are inherited due to this. We include it for convenience as it occurs frequently in our modelling experiences. This pattern has additional synergies with the Event Pattern [**?**, **?**].

3.13.2 Axiomatization

$$\begin{aligned}
\mathsf{ParticipantRole} &\sqsubseteq \mathsf{AgentRole} && (1)\\
\mathsf{providesParticipantRole} &\sqsubseteq \mathsf{providesAgentRole} && (2)\\
\top &\sqsubseteq \forall \mathsf{providesParticipantRole}.\mathsf{ParticipantRole} && (3)\\
\mathsf{AgentRole} &\sqsubseteq {=}1\mathsf{isPerformedBy}.\mathsf{Agent} && (4)\\
\mathsf{AgentRole} &\sqsubseteq {=}1\mathsf{hasTemporalExtent}.\mathsf{TemporalExtent} && (5)\\
\exists \mathsf{isPerformedBy}.\mathsf{Agent} &\sqsubseteq \mathsf{AgentRole} && (6)\\
\mathsf{AgentRole} &\sqsubseteq \forall \mathsf{isPerformedBy}.\mathsf{Agent} && (7)\\
\exists \mathsf{hasTemporalExtent}.\mathsf{TemporalExtent} &\sqsubseteq \mathsf{AgentRole} && (8)\\
\top &\sqsubseteq \forall \mathsf{hasTemporalExtent}.\mathsf{TemporalExtent} && (9)\\
\top &\sqsubseteq \forall \mathsf{providesAgentRole}.\mathsf{AgentRole} && (10)\\
\mathsf{isPerformedBy} &\equiv \mathsf{performsAgentRole}^{-} && (11)\\
\mathsf{isProvidedBy} &\equiv \mathsf{providesAgentRole}^{-} && (12)
\end{aligned}$$

3.13.3 Explanations

1. Subclass: every ParticipantRole is an AgentRole.
2. Subproperty: providesParticipantRole is a subproperty of providesAgentRole.
3. Range: the range of providesParticipantRole is ParticipantRole.
4. Exactly one Agent performs an AgentRole.
5. An AgentRole has exactly one TemporalExtent.
6. Scoped Domain: the scoped domain of isPerformedBy, scoped by Agent, is AgentRole.
7. Scoped Range: the scoped range of isPerformedBy, scoped by AgentRole, is Agent.
8. Scoped Domain: the scoped domain of hasTemporalExtent, scoped by TemporalExtent, is AgentRole.
9. Range: the range of hasTemporalExtent is TemporalExtent.
10. Range: the range of providesAgentRole is AgentRole.
11. Inverse Alias.
12. Inverse Alias.

3.13.4 Competency Questions

CQ1. Who were the participants in this event?
CQ2. Which students attended the lecture?
CQ3. Who were the passengers on the cruise?

3.14 Name Stub

Figure 3.14: Schema Diagram for Name Stub. The visual notation is explained in Chapter **??**.

3.14.1 Summary

The NameStub Pattern is a specialization of the Stub Pattern found in Section **??**. It is included here for convenience as it is has been frequently encountered in our modelling experiences.

3.14.2 Axiomatization

$$\top \sqsubseteq \forall \mathsf{nameAsString}.\mathsf{xsd{:}string} \quad (1)$$

3.14.3 Explanations

1. Range: the range of nameAsString is xsd:string.

3.14.4 Competency Question

CQ1. What is the name of the lecturer?

3.15 Quantities and Units

Figure 3.15: Schema Diagram for Quantities and Units. The visual notation is explained in Chapter **??**.

3.15.1 Summary

This pattern is heavily adapted from QUDT[1] and [**?**]. This pattern allows a developer to express a quantity of some stuff. The nature of quantities is rather complex, due to the fact that there are a multitude of dimensions, unit types, and ways to measure quantities. The Quantity class is used to express the nature of the quantity via its QuantityKind. This is intended to be a controlled vocabulary. We direct the reader to QUDT's documentation for further exploration. A QuantityValue expresses the magnitude of the Quantity via an xsd:double and a Unit. Unit is also recommended to be a controlled vocabulary. Both hasQuantityKind and hasUnit are instances of the Explicit Typing Pattern (Section **??**).

3.15.2 Axiomatization

$$\top \sqsubseteq \forall \mathsf{hasQuantityKind}.\mathsf{QuantityKind} \tag{1}$$
$$\top \sqsubseteq \forall \mathsf{hasQuantityValue}.\mathsf{QuantityValue} \tag{2}$$
$$\top \sqsubseteq \forall \mathsf{hasUnit}.\mathsf{Unit} \tag{3}$$
$$\top \sqsubseteq \forall \mathsf{hasNumericalValue}.\mathsf{xsd{:}double} \tag{4}$$

3.15.3 Explanations

1. Range: the range of hasQuantityKind is QuantityKind.
2. Range: the range of hasQuantityValue is QuantityValue.
3. Range: the range of hasUnit is Unit.
4. Range: the range of hasNumericValue is xsd:double.

3.15.4 Competency Questions

CQ1. How much does an elephant weigh in kilograms?
CQ2. How long is Jupiter from the Sun, at its farthest, in furlongs?
CQ3. How long ago was the Mezazoic Era?

[1]`http://www.qudt.org/release2/qudt-catalog.html`

3.16 Partonymy/Meronymy

Figure 3.16: Schema Diagram for Partonymy.

3.16.1 Summary

Part-whole relations are of fundamental importance for how we organize concepts. This pattern follows an approach laid out by Winston in his 1987 landmark paper on "A Taxonomy of Part-Whole Relations" [**?**] which was based on linguistic considerations, but also provided for logical characterizations and axiomatics, and, as such, inform the pattern.

Essentially, we distinguish between different, interacting partonomies. For example, a component may be part of an engine, which is part of a plane, which belongs to a fleet. These are all part-hood relationships, but they are not transitive.

3.16.2 Axiomatization

$$\text{po-component} \circ \text{po-component} \sqsubseteq \text{po-component} \quad (1)$$
$$\text{po-member} \circ \text{po-member} \sqsubseteq \text{po-member} \quad (2)$$
$$\text{po-portion} \circ \text{po-portion} \sqsubseteq \text{po-portion} \quad (3)$$
$$\text{po-stuff} \circ \text{po-stuff} \sqsubseteq \text{po-stuff} \quad (4)$$
$$\text{po-feature} \circ \text{po-feature} \sqsubseteq \text{po-feature} \quad (5)$$
$$\text{po-place} \circ \text{po-place} \sqsubseteq \text{po-place} \quad (6)$$
$$\text{AsymmetricObjectProperty}(\text{po-component}) \quad (7)$$
$$\text{AsymmetricObjectProperty}(\text{po-member}) \quad (8)$$
$$\text{AsymmetricObjectProperty}(\text{po-portion}) \quad (9)$$
$$\text{AsymmetricObjectProperty}(\text{po-stuff}) \quad (10)$$
$$\text{AsymmetricObjectProperty}(\text{po-feature}) \quad (11)$$
$$\text{AsymmetricObjectProperty}(\text{po-place}) \quad (12)$$
$$\text{po-component} \sqsubseteq \text{part-of} \quad (13)$$
$$\text{po-member} \sqsubseteq \text{part-of} \quad (14)$$
$$\text{po-portion} \sqsubseteq \text{part-of} \quad (15)$$
$$\text{po-stuff} \sqsubseteq \text{part-of} \quad (16)$$
$$\text{po-feature} \sqsubseteq \text{part-of} \quad (17)$$
$$\text{po-place} \sqsubseteq \text{part-of} \quad (18)$$
$$\text{spatially-located-in} \circ \text{spatially-located-in} \sqsubseteq \text{spatially-located-in} \quad (19)$$
$$\text{ReflexiveObjectProperty}(\text{spatially-located-in}) \quad (20)$$
$$\text{po-component} \circ \text{spatially-located-in} \sqsubseteq \text{spatially-located-in} \quad (21)$$
$$\text{spatially-located-in} \circ \text{po-component} \sqsubseteq \text{spatially-located-in} \quad (22)$$

$$\text{po-member} \circ \text{spatially-located-in} \sqsubseteq \text{spatially-located-in} \tag{23}$$
$$\text{spatially-located-in} \circ \text{po-member} \sqsubseteq \text{spatially-located-in} \tag{24}$$
$$\text{po-portion} \circ \text{spatially-located-in} \sqsubseteq \text{spatially-located-in} \tag{25}$$
$$\text{spatially-located-in} \circ \text{po-portion} \sqsubseteq \text{spatially-located-in} \tag{26}$$
$$\text{po-stuff} \circ \text{spatially-located-in} \sqsubseteq \text{spatially-located-in} \tag{27}$$
$$\text{spatially-located-in} \circ \text{po-stuff} \sqsubseteq \text{spatially-located-in} \tag{28}$$
$$\text{po-feature} \circ \text{spatially-located-in} \sqsubseteq \text{spatially-located-in} \tag{29}$$
$$\text{spatially-located-in} \circ \text{po-feature} \sqsubseteq \text{spatially-located-in} \tag{30}$$
$$\text{po-place} \circ \text{spatially-located-in} \sqsubseteq \text{spatially-located-in} \tag{31}$$
$$\text{spatially-located-in} \circ \text{po-place} \sqsubseteq \text{spatially-located-in} \tag{32}$$
$$\text{Po-Component-Type} \sqsubseteq \text{RelationInstance} \tag{33}$$
$$\text{Po-Member-Type} \sqsubseteq \text{RelationInstance} \tag{34}$$
$$\text{Po-Portion-Type} \sqsubseteq \text{RelationInstance} \tag{35}$$
$$\text{Po-Stuff-Type} \sqsubseteq \text{RelationInstance} \tag{36}$$
$$\text{Po-Feature-Type} \sqsubseteq \text{RelationInstance} \tag{37}$$
$$\text{Po-Place-Type} \sqsubseteq \text{RelationInstance} \tag{38}$$
$$\text{Po-Part-Of-Type} \sqsubseteq \text{RelationInstance} \tag{39}$$
$$\text{Spatially-Located-In-Type} \sqsubseteq \text{RelationInstance} \tag{40}$$

3.16.3 Explanations

1. Transitivity.
2. Transitivity.
3. Transitivity.
4. Transitivity.
5. Transitivity.
6. Transitivity.
7. Asymmetric Object Property.
8. Asymmetric Object Property.
9. Asymmetric Object Property.
10. Asymmetric Object Property.
11. Asymmetric Object Property.
12. Asymmetric Object Property.
13. Subclass.
14. Subclass.
15. Subclass.
16. Subclass.
17. Subclass.
18. Subclass.
19. Transitivity.
20. Reflexive Object Property.
21. Role Chain: the concatenation of po-component and spatially-located-in is spatially-located-in.

22. Role Chain: the concatenation of spatially-located-in and po-component is spatially-located-in.
23. Role Chain: the concatenation of po-member and spatially-located-in is spatially-located-in.
24. Role Chain: the concatenation of spatially-located-in and po-member is spatially-located-in.
25. Role Chain: the concatenation of po-portion and spatially-located-in is spatially-located-in.
26. Role Chain: the concatenation of spatially-located-in and po-portion is spatially-located-in.
27. Role Chain: the concatenation of po-stuff and spatially-located-in is spatially-located-in.
28. Role Chain: the concatenation of spatially-located-in and po-stuff is spatially-located-in.
29. Role Chain: the concatenation of po-feature and spatially-located-in is spatially-located-in.
30. Role Chain: the concatenation of spatially-located-in and po-feature is spatially-located-in.
31. Role Chain: the concatenation of po-place and spatially-located-in is spatially-located-in.
32. Role Chain: the concatenation of spatially-located-in and po-place is spatially-located-in.
33. Subclass.
34. Subclass.
35. Subclass.
36. Subclass.
37. Subclass.
38. Subclass.
39. Subclass.
40. Subclass.

3.16.4 Competency Question

CQ1. Is the Everglades part of Florida?
CQ2. Is the plane in the Warehouse?
CQ3. What are all engine components?
CQ4. Is he part of the family?

3.17 Provenance

Figure 3.17: Schema Diagram for the Provenance Pattern. The visual notation is explained in Chapter **??**.

3.17.1 Summary

The EntityWithProvenance Pattern is extracted from the PROV-O ontology. At the pattern level, we do not want to make the ontological committment to a full-blown ontology. It suffices to align a sub-pattern to the core of PROV-O [**?**].

The EntityWithProvenance class is any item of interest to which a developer would like to attach provenance information. That is they are interested in capturing, who or what created that item, what was used to derive it, and what method was used to do so. The "who or what" is captured by using the Agent class. The property, wasDerivedFrom is eponymous—it denotes that some set of resources was used during the ProvenanceActivity to generate the EntityWithProvenance.

3.17.2 Axiomatization

$$\exists \mathsf{attributedTo}.\mathsf{Agent} \sqsubseteq \mathsf{EntityWithProvenance} \quad (1)$$
$$\mathsf{EntityWithProvenance} \sqsubseteq \forall \mathsf{attributedTo}.\mathsf{Agent} \quad (2)$$
$$\exists \mathsf{generatedBy}.\mathsf{ProvenanceActivity} \sqsubseteq \mathsf{EntityWithProvenance} \quad (3)$$
$$\mathsf{EntityWithProvenance} \sqsubseteq \forall \mathsf{generatedBy}.\mathsf{ProvenanceActivity} \quad (4)$$
$$\exists \mathsf{used}.\mathsf{EntityWithProvenance} \sqsubseteq \mathsf{ProvenanceActivity} \quad (5)$$
$$\mathsf{ProvenanceActivity} \sqsubseteq \forall \mathsf{used}.\mathsf{EntityWithProvenance} \quad (6)$$
$$\exists \mathsf{performedBy}.\mathsf{Agent} \sqsubseteq \mathsf{ProvenanceActivity} \quad (7)$$
$$\mathsf{ProvenanceActivity} \sqsubseteq \forall \mathsf{performedBy}.\mathsf{Agent} \quad (8)$$

3.17.3 Explanations

1. Scoped Domain:The scoped domain of attributedTo, scoped by Agent, is EntityWithProvenance.
2. Scoped Range: The scoped range of attributedTo, scoped by EntityWithProvenance, is Agent.
3. Scoped Domain:The scoped domain of generatedBy, scoped by ProvenanceActivity, is EntityWithProvenance.
4. Scoped Range: The scoped range of generatedBy, scoped by EntityWithProvenance, is ProvenanceActivity.
5. Scoped Domain:The scoped domain of used, scoped by EntityWithProvenance, is ProvenanceActivity
6. Scoped Range: The scoped range of used, scoped by ProvenananceActivity, is EntityWithProvenance.

7. Scoped Domain:The scoped domain of performedBy, scoped by Agent, is ProvenanceActivity.
8. Scoped Range: The scoped range of performedBy, scoped by ProvenanceActivity, is Agent.

3.17.4 Competency Questions

CQ1. Who are the contributors to this Wikidata page?
CQ2. From which database is this entry taken?
CQ3. Which method was used to generate this chart and from which spreadsheet did the data originate?
CQ4. Who provided this research result?

3.18 Identifier

Figure 3.18: Schema Diagram for the Identifier Pattern. The visual notation is explained in Chapter **??**.

3.18.1 Summary

This pattern is used for associating some sort of identifier and metadata with a thing. One could view this pattern as a reification of the ExplicitType Pattern as found in Section **??**. In this case, we wish to associate additional information aside from its type with a thing, e.g. an identifier may be a URL or a primary key value in a database. We believe that this pattern meshes well with the EntityWithProvenance Pattern which may be found in Section **??**.

3.18.2 Axiomatization

$$\top \sqsubseteq \forall \mathsf{hasIdentifier}.\mathsf{Identifier} \tag{1}$$
$$\exists \mathsf{hasIdentifierType}.\top \sqsubseteq \mathsf{Identifier} \tag{2}$$
$$\top \sqsubseteq \forall \mathsf{hasIdentifierType}.\mathsf{IDType} \tag{3}$$
$$\top \sqsubseteq \forall \mathsf{identifierAsText}.\mathsf{xsd{:}string} \tag{4}$$

3.18.3 Explanations

1. Range: the range of hasIdentifier is Identifier.
2. Domain: the domain of hasIdentifierType is Identifier.
3. Range: the range of hasIdentifierType is IDType.
4. Range: the range of identifierAsText is xsd:string.

3.18.4 Competency Questions

CQ1. The merchant is assigned what identifier in this historical databse?
CQ2. Where can this information be validated/obtained?