A First-Order Logic Formalization of the Unified Foundational Ontology

Daniele Porelo, Claudenir M. Fonseca, João Paulo A. Almeida,
Giancarlo Guizzardi, Tiago Prince Sales

December 3, 2021

Abstract

This document presents a formalization of the Unified Foundational Ontology (UFO) in first-order logic. This formalization is documented by means of three complementary representations: (i) a representation in standard Common Logic using the CLIF syntax; (ii) a representation in natural language; and, when applicable, (iii) a UML-based diagrammatic representation. The presented formalization is supported by consistency and satisfiability checks performed through automated proofing tools.

1 Introduction

This document presents a formalization of the Unified Foundational Ontology (UFO) in first-order logic. This formalization is documented by means of three complementary representations: (i) a representation in standard Common Logic using the CLIF syntax; (ii) a representation in natural language; and, when applicable, (iii) a UML-based diagrammatic representation. The presented formalization is supported by consistency and satisfiability checks performed through automated proofing tools.

The remainder of this document is organized as a single formalization section (Section 2), which contains subsections for each submodule of the ontology.

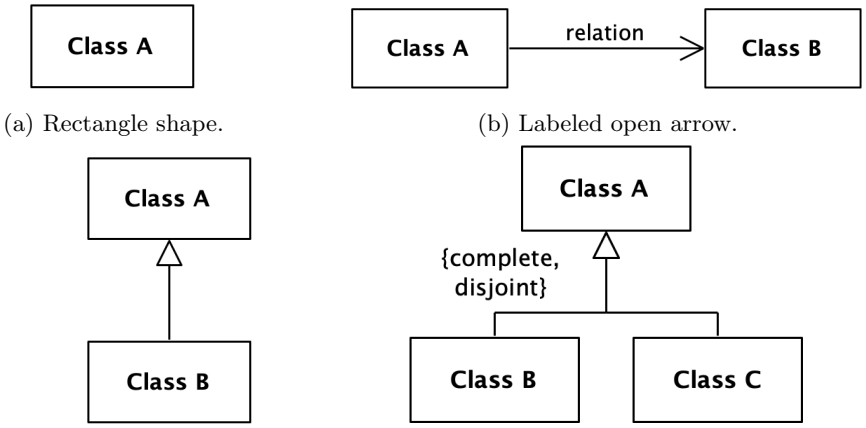
2 Formalization

This section contains the formalization of the Unified Foundational Ontology (UFO) in first-order logics. This formalization is organized in several subsections where each presents the formalization of a portion of the whole ontology. The formalization is presented through different equivalent representations, designed to support the understanding of its contents: (i) a representation in standard Common Logic using the CLIF syntax; (ii) a representation in natural language; and, when applicable, (iii) a UML-based diagrammatic representation.

The UML-based diagrammatic representation serves as a visual representation certain predicates and axioms, being each element in Figure 1 being translated as follows:

- Rectangle shape (Figure 1a): visual representation of unary predicates associated to types in the ontology; the associated predicate is shown in lower camel case with no spaces.

  $classA(x)$



(a) Rectangle shape.

(b) Labeled open arrow.

(c) Closed arrow.

(d) Labeled closed arrows.

Figure 1: UML-based representation of first-order logic axioms.

- Open arrow (Figure 1b): visual representation of binary predicates; the predicate associated to the arrows' label is shown in lower camel case with no spaces; the predicate can only be true for any x and y if it is also true predicates associated to the types of each end (keeping the order of the arrow in the binary predicate's positions); this representation may also be associated to ternary predicates ifif its third position represents a time-index.

  $\forall x, y(relation(x, y) \rightarrow (classA(x) \wedge classB(y)))$

  $\forall x, y, w(relation(x, y, w) \rightarrow (classA(x) \wedge classB(y) \wedge world(w)))$

- Closed arrow (Figure 1c): visual representation of specializations between ontology's types, where the type in the tail of the arrow is a subtype of the type in the head of the arrow.

  $\forall x(classB(x) \rightarrow classA(x))$

- Labeled closed arrow (Figure 1d): visual representation of disjoint and/or complete constraints over sets specializations between ontology's types.

  $\forall x(classB(x) \rightarrow classA(x))$
  $\forall x(classC(x) \rightarrow classA(x))$
  $\forall x(classA(x) \rightarrow (classB(x) \vee classC(x)))$ {complete}
  $\neg\exists x(classB(x) \wedge classC(x))$ {disjoint}

2.1 Partial Taxonomy of UFO: Thing

This subsection presents most general types of UFO's taxonomy specializing the type Thing (Figure 2).

a1 For every x, x is a Thing ifif x is either a Type or an Individual.

$\forall x(\mathsf{type\_}(x) \vee \mathsf{individual}(x) \leftrightarrow \mathsf{thing}(x))$

```
(cl-text  ax_thing_taxonomy
(forall (x)
  (iff (or (type_ x) (individual x))
```

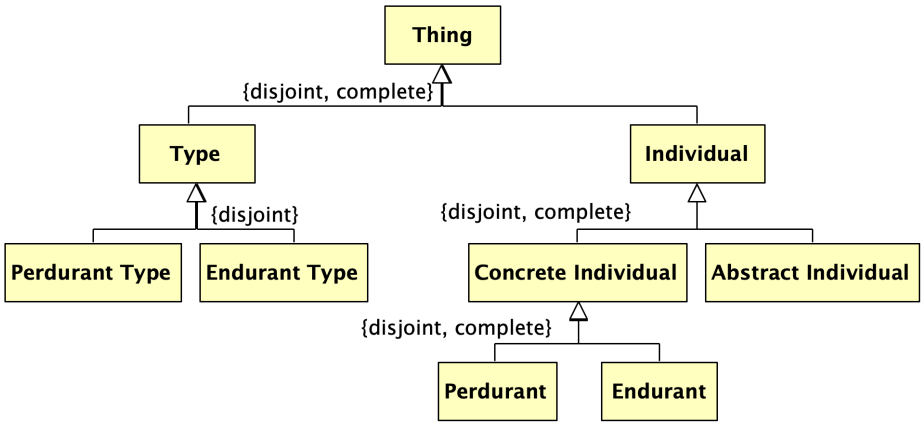


Figure 2: Visual representation of UFO's taxonomy of Thing.

```
    (thing x))
  )
)
```

a2 There is no x such that it is a Type and an Individual.
$\neg\exists x(\mathsf{type\_}(x) \wedge \mathsf{individual}(x))$

```
(cl-text  ax_thing_partition
(not (exists (x)
  (and (type_ x) (individual x)))
)
)
```

a3 For every x, x is an Individual iff x is either a Concrete Individual or an Abstract Individual.
$\forall x(\mathsf{concreteIndividual}(x) \vee \mathsf{abstractIndividual}(x) \leftrightarrow \mathsf{individual}(x))$

```
(cl-text  ax_individual_taxonomy
(forall (x)
  (iff (or (concreteIndividual x) (abstractIndividual x))
    (individual x))
  )
)
```

a4 There is no x such that it is a Concrete Individual and an Abstract Individual.
$\neg\exists x(\mathsf{concreteIndividual}(x) \wedge \mathsf{abstractIndividual}(x))$

```
(cl-text  ax_individual_partition
(not (exists (x)
  (and (concreteIndividual x) (abstractIndividual x)))
)
)
```

a5 For every x, x is a Concrete Individual iff x is either a Perdurant or an Endurant.
$\forall x(\mathsf{endurant}(x) \vee \mathsf{perdurant}(x) \leftrightarrow \mathsf{concreteIndividual}(x))$

```
(cl-text  ax_concreteIndividual_taxonomy
(forall (x)
  (iff (or (endurant x) (perdurant x))
    (concreteIndividual x))
  )
)
```

a6 There is no x such that it is a Perdurant and an Endurant.

$\neg\exists x(\mathsf{endurant}(x) \wedge \mathsf{perdurant}(x))$

```
(cl-text  ax_concreteIndividual_partition
(not (exists (x)
  (and (endurant x) (perdurant x)))
)
)
```

a7 For every x, x is a Concrete Individual iff x is either a Perdurant or an Endurant.

$\forall x(\mathsf{endurantType}(x) \vee \mathsf{perdurantType}(x) \rightarrow \mathsf{type\_}(x))$

```
(cl-text  ax_type_taxonomy
(forall (x)
  (if (or (endurantType x) (perdurantType x))
    (type_ x))
  )
)
```

a8 There is no x such that it is a Perdurant Type and an Endurant Type.

$\neg\exists x(\mathsf{endurantType}(x) \wedge \mathsf{perdurantType}(x))$

```
(cl-text  ax_type_partition
(not (exists (x)
  (and (endurantType x) (perdurantType x)))
)
)
```

2.2 Partial Taxonomy of UFO: Abstract Individual

This subsection presents a portion of UFO's taxonomy specializing the type Abstract Individual (Figure 3).

a9 Every x that is a Quale is also an Abstract Individual.

$\forall x(\mathsf{quale}(x) \rightarrow \mathsf{abstractIndividual}(x))$

```
(cl-text  ax_abstractIndividual_taxonomy_quale
(forall (x)
  (if (quale x)
    (abstractIndividual x))
  )
)
```

a10 Every x that is a Set is also an Abstract Individual.

$\forall x(\mathsf{set\_}(x) \rightarrow \mathsf{abstractIndividual}(x))$

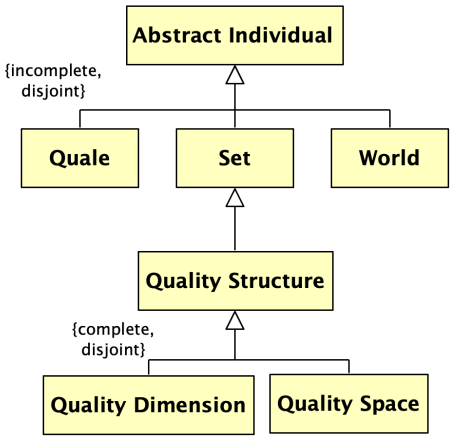


Figure 3: Visual representation of UFO's taxonomy of Abstract Individual.

```
(cl-text  ax_abstractIndividual_taxonomy_set
(forall (x)
  (if (set_ x)
    (abstractIndividual x))
  )
)
```

a11 Every x that is a World is also an Abstract Individual.

$\forall x(\mathsf{world}(x) \rightarrow \mathsf{abstractIndividual}(x))$

```
(cl-text  ax_abstractIndividual_taxonomy_world
(forall (x)
  (if (world x)
    (abstractIndividual x))
  )
)
```

a12 There is no x such that it is a Quale, a Set, and a World (pairwise disjoint).

$\neg\exists x((\mathsf{quale}(x) \wedge \mathsf{set\_}(x)) \vee (\mathsf{quale}(x) \wedge \mathsf{world}(x)) \vee (\mathsf{set\_}(x) \wedge \mathsf{world}(x)))$

```
(cl-text  ax_abstractIndividual_pairwiseDisjoint
(not (exists (x)
  (or (and (quale x) (set_ x)) (and (quale x) (world x)) (and (set_ x) (world x)
    )))
)
)
```

a13 Every x that is a Quality Structure is also a Set.

$\forall x(\mathsf{qualityStructure}(x) \rightarrow \mathsf{set\_}(x))$

```
(cl-text  ax_set_taxonomy_qualityStructure
(forall (x)
  (if (qualityStructure x)
    (set_ x))
  )
)
```

a14 For every x, x is a Quality Structure iff x is either a Quality Dimension or a Quality Space.
$\forall x(\mathsf{qualityDimension}(x) \vee \mathsf{qualitySpace}(x) \leftrightarrow \mathsf{qualityStructure}(x))$

```
(cl-text  ax_qualityStructure_taxonomy
(forall (x)
  (iff (or (qualityDimension x) (qualitySpace x))
    (qualityStructure x))
  )
)
```

a15 There is no x such that it is a Quality Dimension and a Quality Space.
$\neg\exists x(\mathsf{qualityDimension}(x) \wedge \mathsf{qualitySpace}(x))$

```
(cl-text  ax_qualityStructure_partition
(not (exists (x)
  (and (qualityDimension x) (qualitySpace x)))
)
)
```

2.3 Partial Taxonomy of UFO: Endurant

This subsection presents a portion of UFO's taxonomy specializing the type Endurant (Figure 4).

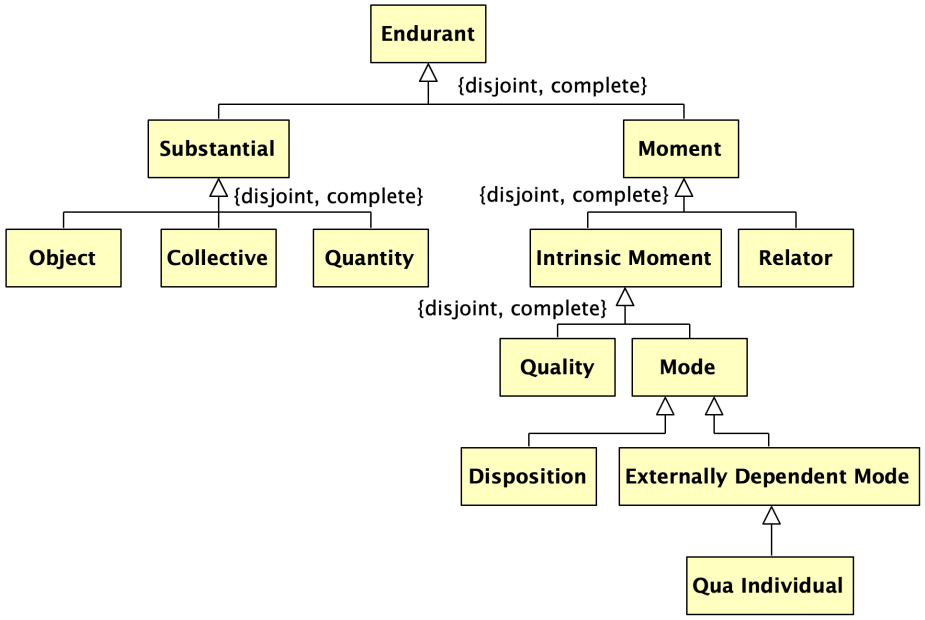


Figure 4: Visual representation of UFO's taxonomy of Endurant.

a16 For every x, x is an Endurant iff x is either a Substantial or a Moment.
$\forall x(\mathsf{substantial}(x) \vee \mathsf{moment}(x) \leftrightarrow \mathsf{endurant}(x))$

```
(cl-text  ax_endurant_taxonomy
(forall (x)
  (iff (or (substantial x) (moment x))
    (endurant x))
  )
)
```

a17 There is no x such that it is a Substantial and a Moment.

$\neg\exists x(\mathsf{substantial}(x) \wedge \mathsf{moment}(x))$

```
(cl-text  ax_endurant_partition
(not (exists (x)
  (and (substantial x) (moment x)))
)
)
```

a18 For every x, x is a Substantial ifif x is either an Object, a Collective, or a Quantity.

$\forall x(\mathsf{object}(x) \vee \mathsf{collective}(x) \vee \mathsf{quantity}(x) \leftrightarrow \mathsf{substantial}(x))$

```
(cl-text  ax_substantial_taxonomy
(forall (x)
  (iff (or (object x) (collective x) (quantity x))
    (substantial x))
  )
)
```

a19 There is no x such that it is an Object, a Collective, and a Quantity (pairwise disjoint).

$\neg\exists x((\mathsf{object}(x) \wedge \mathsf{collective}(x)) \vee (\mathsf{object}(x) \wedge \mathsf{quantity}(x)) \vee (\mathsf{collective}(x) \wedge \mathsf{quantity}(x)))$

```
(cl-text  ax_substantial_partition
(not (exists (x)
  (or (and (object x) (collective x)) (and (object x) (quantity x)) (and (
    collective x) (quantity x))))
)
)
```

a20 For every x, x is a Moment ifif x is either an Intrinsic Moment or a Relator.

$\forall x(\mathsf{intrinsicMoment}(x) \vee \mathsf{relator}(x) \leftrightarrow \mathsf{moment}(x))$

```
(cl-text  ax_moment_taxonomy
(forall (x)
  (iff (or (intrinsicMoment x) (relator x))
    (moment x))
  )
)
```

a21 There is no x such that it is an Intrinsic Moment and a Relator.

$\neg\exists x(\mathsf{intrinsicMoment}(x) \wedge \mathsf{relator}(x))$

```
(cl-text  ax_moment_partition
(not (exists (x)
  (and (intrinsicMoment x) (relator x)))
)
)
```

a22 For every x, x is an Intrinsic Moment ifif x is either a Quality or a Mode.

$\forall x(\mathsf{quality}(x) \vee \mathsf{mode}(x) \leftrightarrow \mathsf{intrinsicMoment}(x))$

```
(cl-text  ax_intrinsicMoment_taxonomy
(forall (x)
  (iff (or (quality x) (mode x))
    (intrinsicMoment x))
  )
)
```

a23 There is no x such that it is an Intrinsic Moment and a Relator.

$\neg\exists x(\mathsf{quality}(x) \wedge \mathsf{mode}(x))$

```
(cl-text  ax_intrinsicMoment_partition
(not (exists (x)
  (and (quality x) (mode x)))
)
)
```

a24 Every x that is a Disposition is also a Mode.

$\forall x(\mathsf{disposition}(x) \rightarrow \mathsf{mode}(x))$

```
(cl-text  ax_mode_taxonomy_disposition
(forall (x)
  (if (disposition x)
    (mode x))
  )
)
```

a25 Every x that is an Externally Dependent Mode is also a Mode.

$\forall x(\mathsf{externallyDependentMode}(x) \rightarrow \mathsf{mode}(x))$

```
(cl-text  ax_mode_taxonomy_externallyDependentMode
(forall (x)
  (if (externallyDependentMode x)
    (mode x))
  )
)
```

a26 Every x that is an Qua Individual is also an Externally Dependent Mode.

$\forall x(\mathsf{quaIndividual}(x) \rightarrow \mathsf{externallyDependentMode}(x))$

```
(cl-text  ax_externallyDependentMode_taxonomy_quaIndividual
(forall (x)
  (if (quaIndividual x)
    (externallyDependentMode x))
  )
)
```

2.4 Partial Taxonomy of UFO: **Endurant Type** by Ontological Natures

This subsection presents a portion of UFO's taxonomy specializing the type Endurant Type classified by ontological natures (Figure 5).

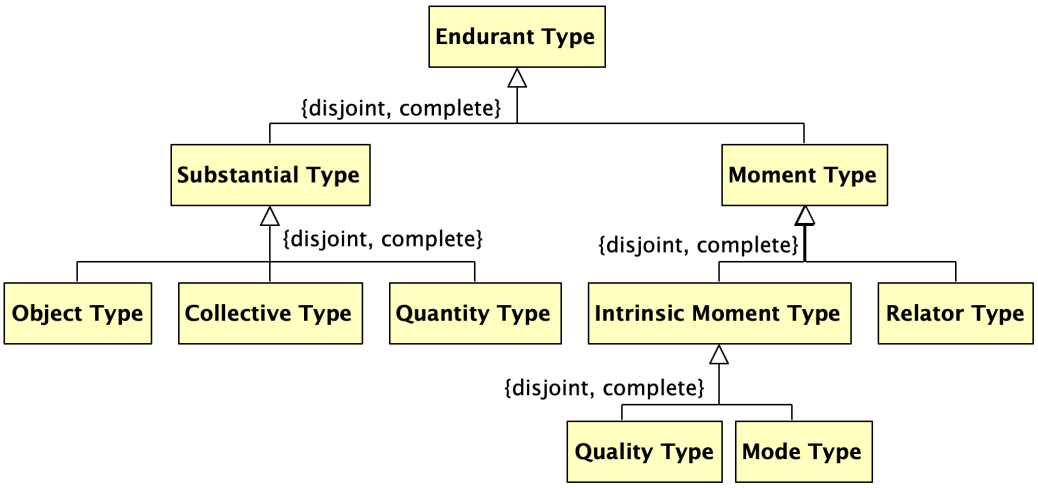


Figure 5: Visual representation of UFO's taxonomy of Endurant Type classified by ontological natures.

a27 For every x, x is an Endurant Type ifif x is either a Substantial Type or a Moment Type.

$\forall x(\mathsf{substantialType}(x) \vee \mathsf{momentType}(x) \leftrightarrow \mathsf{endurantType}(x))$

```
(cl-text  ax_endurantType_taxonomy_nature
(forall (x)
  (iff (or (substantialType x) (momentType x))
    (endurantType x))
  )
)
```

a28 There is no x such that it is a Substantial Type and a Moment Type.

$\neg\exists x(\mathsf{substantialType}(x) \wedge \mathsf{momentType}(x))$

```
(cl-text  ax_endurantType_partition_nature
(not (exists (x)
  (and (substantialType x) (momentType x)))
)
)
```

a29 For every x, x is a Substantial Type ifif x is either an Object Type, a Collective Type, or a Quantity Type.

$\forall x(\mathsf{objectType}(x) \vee \mathsf{collectiveType}(x) \vee \mathsf{quantityType}(x) \leftrightarrow \mathsf{substantialType}(x))$

```
(cl-text  ax_substantialType_taxonomy
(forall (x)
  (iff (or (objectType x) (collectiveType x) (quantityType x))
    (substantialType x))
  )
)
```

a30 There is no x such that it is an Object Type, a Collective Type, and a Quantity Type (pairwise disjoint).

$\neg\exists x((\mathsf{objectType}(x) \wedge \mathsf{collectiveType}(x)) \vee (\mathsf{objectType}(x) \wedge \mathsf{quantityType}(x)) \vee (\mathsf{collectiveType}(x) \wedge \mathsf{quantityType}(x)))$

```
(cl-text  ax_substantialType_partition
(not (exists (x)
  (or (and (objectType x) (collectiveType x)) (and (objectType x) (quantityType
    x)) (and (collectiveType x) (quantityType x))))
)
)
```

a31 For every x, x is a Moment Type iff x is either an Intrinsic Moment Type or a Relator Type.

$\forall x(\mathsf{intrinsicMomentType}(x) \vee \mathsf{relatorType}(x) \leftrightarrow \mathsf{momentType}(x))$

```
(cl-text  ax_momentType_taxonomy
(forall (x)
  (iff (or (intrinsicMomentType x) (relatorType x))
    (momentType x))
  )
)
```

a32 There is no x such that it is an Intrinsic Moment Type and a Relator Type.

$\neg\exists x(\mathsf{intrinsicMomentType}(x) \wedge \mathsf{relatorType}(x))$

```
(cl-text  ax_momentType_partition
(not (exists (x)
  (and (intrinsicMomentType x) (relatorType x)))
)
)
```

a33 For every x, x is an Intrinsic Moment Type iff x is either a Quality Type or a Mode Type.

$\forall x(\mathsf{qualityType}(x) \vee \mathsf{modeType}(x) \leftrightarrow \mathsf{intrinsicMomentType}(x))$

```
(cl-text  ax_intrinsicMomentType_taxonomy
(forall (x)
  (iff (or (qualityType x) (modeType x))
    (intrinsicMomentType x))
  )
)
```

a34 There is no x such that it is an Intrinsic Moment Type and a Relator Type.

$\neg\exists x(\mathsf{qualityType}(x) \wedge \mathsf{modeType}(x))$

```
(cl-text  ax_intrinsicMomentType_partition
(not (exists (x)
  (and (qualityType x) (modeType x)))
)
)
```

2.5 Partial Taxonomy of UFO: **Endurant Type** by Modal Properties of Types

This subsection presents a portion of UFO's taxonomy specializing the type Endurant Type classified by the modal properties of types (Figure 6).

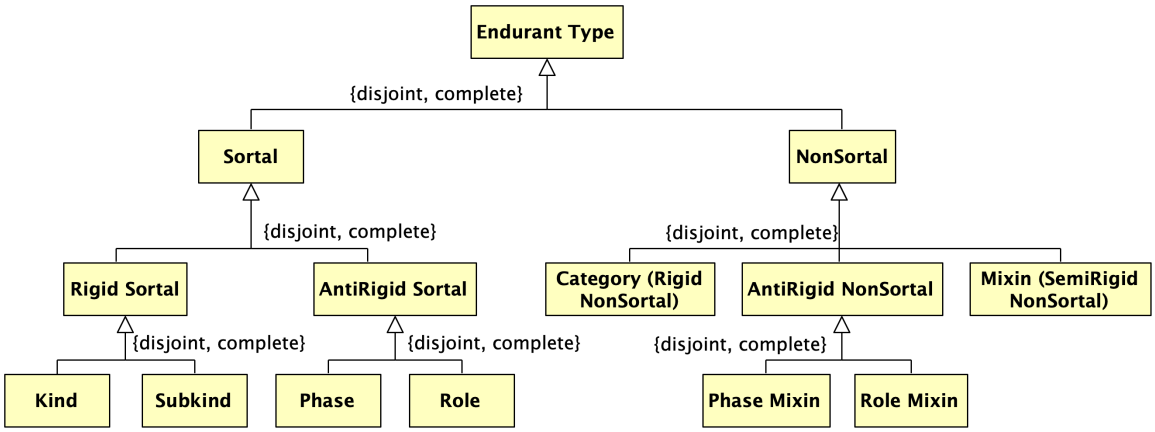


Figure 6: Visual representation of UFO's taxonomy of Endurant Type classified by the modal properties of types.

a35 For every x, x is an Endurant Type ifif x is either a Sortal or a NonSortal.

$\forall x(\mathsf{sortal}(x) \vee \mathsf{nonSortal}(x) \leftrightarrow \mathsf{endurantType}(x))$

```
(cl-text  ax_endurantType_taxonomy_properties
(forall (x)
  (iff (or (sortal x) (nonSortal x))
    (endurantType x))
  )
)
```

a36 There is no x such that it is a Sortal and a NonSortal.

$\neg\exists x(\mathsf{sortal}(x) \wedge \mathsf{nonSortal}(x))$

```
(cl-text  ax_endurantType_partition_properties
(not (exists (x)
  (and (sortal x) (nonSortal x)))
)
)
```

a37 For every x, x is a Sortal ifif x is either a Rigid Sortal or an AntiRigid Sortal.

$\forall x(\mathsf{rigidSortal}(x) \vee \mathsf{antiRigidSortal}(x) \leftrightarrow \mathsf{sortal}(x))$

```
(cl-text  ax_sortal_taxonomy
(forall (x)
  (iff (or (rigidSortal x) (antiRigidSortal x))
    (sortal x))
  )
)
```

a38 There is no x such that it is a Rigid Sortal and an AntiRigid Sortal.

$\neg\exists x(\mathsf{rigidSortal}(x) \wedge \mathsf{antiRigidSortal}(x))$

```
(cl-text  ax_sortal_partition
(not (exists (x)
  (and (rigidSortal x) (antiRigidSortal x)))
```

```
)
)
```

a39 For every x, x is a Rigid Sortal iff x is either a Kind or a Subkind.

$\forall x(\mathsf{kind}(x) \vee \mathsf{subkind}(x) \leftrightarrow \mathsf{rigidSortal}(x))$

```
(cl-text  ax_rigidSortal_taxonomy
(forall (x)
  (iff (or (kind x) (subkind x))
    (rigidSortal x))
  )
)
```

a40 There is no x such that it is a Kind and a Subkind.

$\neg\exists x(\mathsf{kind}(x) \wedge \mathsf{subkind}(x))$

```
(cl-text  ax_rigidSortal_partition
(not (exists (x)
  (and (kind x) (subkind x)))
)
)
```

a41 For every x, x is an AntiRigid Sortal iff x is either a Phase or a Role.

$\forall x(\mathsf{phase}(x) \vee \mathsf{role}(x) \leftrightarrow \mathsf{antiRigidSortal}(x))$

```
(cl-text  ax_antiRigidSortal_taxonomy
(forall (x)
  (iff (or (phase x) (role x))
    (antiRigidSortal x))
  )
)
```

a42 There is no x such that it is a Phase and a Role.

$\neg\exists x(\mathsf{phase}(x) \wedge \mathsf{role}(x))$

```
(cl-text  ax_antiRigidSortal_partition
(not (exists (x)
  (and (phase x) (role x)))
)
)
```

a43 For every x, x is a NonSortal iff x is either a Rigid NonSortal, an AntiRigid NonSortal, or a SemmiRigid NonSortal.

$\forall x(\mathsf{rigidNonSortal}(x) \vee \mathsf{semiRigidNonSortal}(x) \vee \mathsf{antiRigidNonSortal}(x) \leftrightarrow \mathsf{nonSortal}(x))$

```
(cl-text  ax_nonSortal_taxonomy
(forall (x)
  (iff (or (rigidNonSortal x) (semiRigidNonSortal x) (antiRigidNonSortal x))
    (nonSortal x))
  )
)
```

a44 There is no x such that it is an Rigid NonSortal, an AntiRigid NonSortal, and a SemiRigid NonSortal (pairwise disjoint).

$\neg\exists x((\mathsf{rigidNonSortal}(x) \wedge \mathsf{semiRigidNonSortal}(x)) \vee (\mathsf{rigidNonSortal}(x) \wedge \mathsf{antiRigidNonSortal}(x)) \vee (\mathsf{semiRigidNonSortal}(x) \wedge \mathsf{antiRigidNonSortal}(x)))$

```
(cl-text  ax_nonSortal_partition
(not (exists (x)
  (or (and (rigidNonSortal x) (semiRigidNonSortal x)) (and (rigidNonSortal x) (
    antiRigidNonSortal x)) (and (semiRigidNonSortal x) (antiRigidNonSortal x))))
)
)
```

a45 Every x that is a Rigid NonSortal is also a Category.

$\forall x(\mathsf{rigidNonSortal}(x) \leftrightarrow \mathsf{category}(x))$

```
(cl-text  ax_rigidNonSortal_taxonomy
(forall (x)
  (iff (rigidNonSortal x)
    (category x))
  )
)
```

a46 Every x that is a SemiRigid NonSortal is also a Mixin.

$\forall x(\mathsf{semiRigidNonSortal}(x) \leftrightarrow \mathsf{mixin}(x))$

```
(cl-text  ax_semiRigidNonSortal_taxonomy
(forall (x)
  (iff (semiRigidNonSortal x)
    (mixin x))
  )
)
```

a47 For every x, x is an AntiRigid NonSortal iff x is either a Phase Mixin or a Role Mixin.

$\forall x(\mathsf{phaseMixin}(x) \vee \mathsf{roleMixin}(x) \leftrightarrow \mathsf{antiRigidNonSortal}(x))$

```
(cl-text  ax_antiRigidNonSortal_taxonomy
(forall (x)
  (iff (or (phaseMixin x) (roleMixin x))
    (antiRigidNonSortal x))
  )
)
```

a48 There is no x such that it is a Phase Mixin and a Role Mixin.

$\neg\exists x(\mathsf{phaseMixin}(x) \wedge \mathsf{roleMixin}(x))$

```
(cl-text  ax_antiRigidNonSortal_partition
(not (exists (x)
  (and (phaseMixin x) (roleMixin x)))
)
)
```

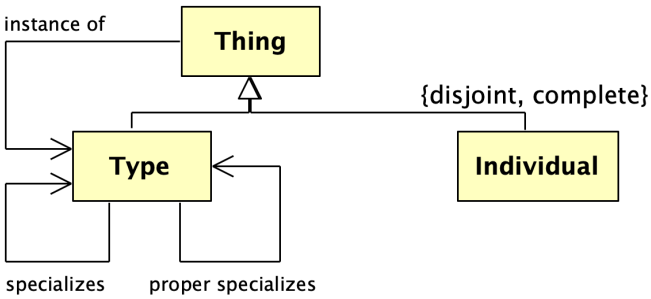


Figure 7: Types, individuals, instantiation, and specialization.

2.6 On the Definitions of Type, Individual, Instantiation, and Specialization

This subsection introduces the UFO's definitions of Type and Individual as well as the definitions for the relations of instance of, specializes, and proper specializes.

a49 The instance of relation can only hold between x, y, and w, where x is an instance of a type y in w, and w is the world where this relation holds.

$\forall x, y, w(\mathsf{iof}(x, y, w) \rightarrow \mathsf{type\_}(y) \wedge \mathsf{world}(w))$

```
(cl-text  ax_dIof
(forall (x y w)
  (if (iof x y w)
    (and (type_ y) (world w)))
  )
)
```

a50 For every x, x is a type iff there exists some y and w such that y is instance of x in the world w.

$\forall x(\mathsf{type\_}(x) \leftrightarrow \exists y, w(\mathsf{iof}(y, x, w)))$

```
(cl-text  ax_dType
(forall (x)
  (iff (type_ x)
    (exists (y w)
      (iof y x w))
    )
  )
)
```

a51 For every x, x is an individual iff there is no pair y and w such that y is instance of x in w.

$\forall x(\mathsf{individual}(x) \leftrightarrow \neg\exists y, w(\mathsf{iof}(y, x, w)))$

```
(cl-text  ax_dIndividual
(forall (x)
  (iff (individual x)
    (not (exists (y w)
      (iof y x w))
    ))
  )
)
```

a52 The instance x in an instance of relation must be either a type or an individual.

$\forall x, y, w(\mathsf{iof}(x, y, w) \rightarrow \mathsf{type\_}(x) \vee \mathsf{individual}(x))$

```
(cl-text  ax_multiLevel
(forall (x y w)
  (if (iof x y w)
    (or (type_ x) (individual x)))
  )
)
```

a53 Chains of instance of relations are limited to a depth of two.

$\neg\exists x, y, z, w(\mathsf{type\_}(x) \wedge \mathsf{iof}(x, y, w) \wedge \mathsf{iof}(y, z, w))$

```
(cl-text  ax_twoLevelConstrained
(not (exists (x y z w)
  (and (type_ x) (iof x y w) (iof y z w)))
)
)
```