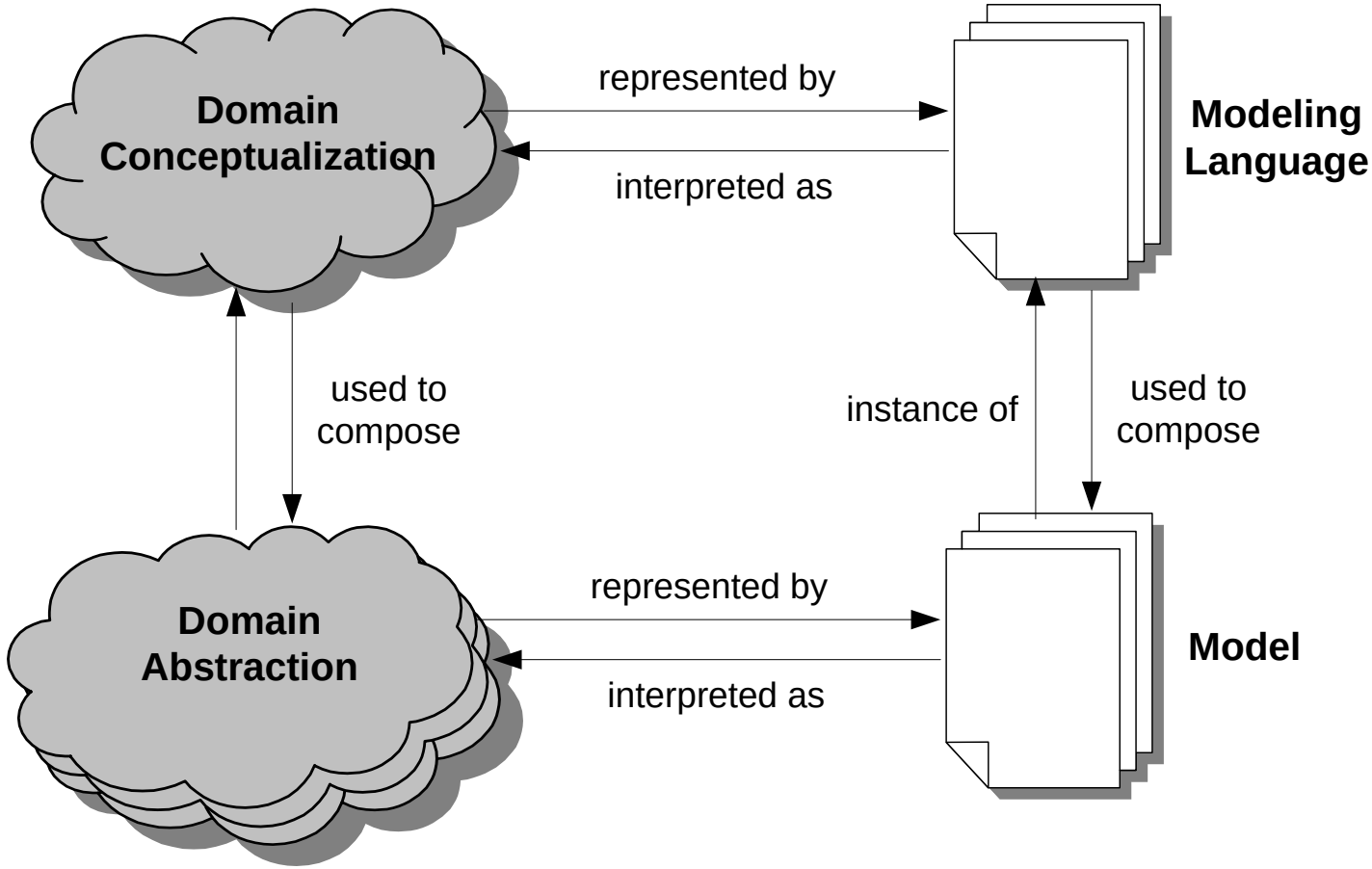
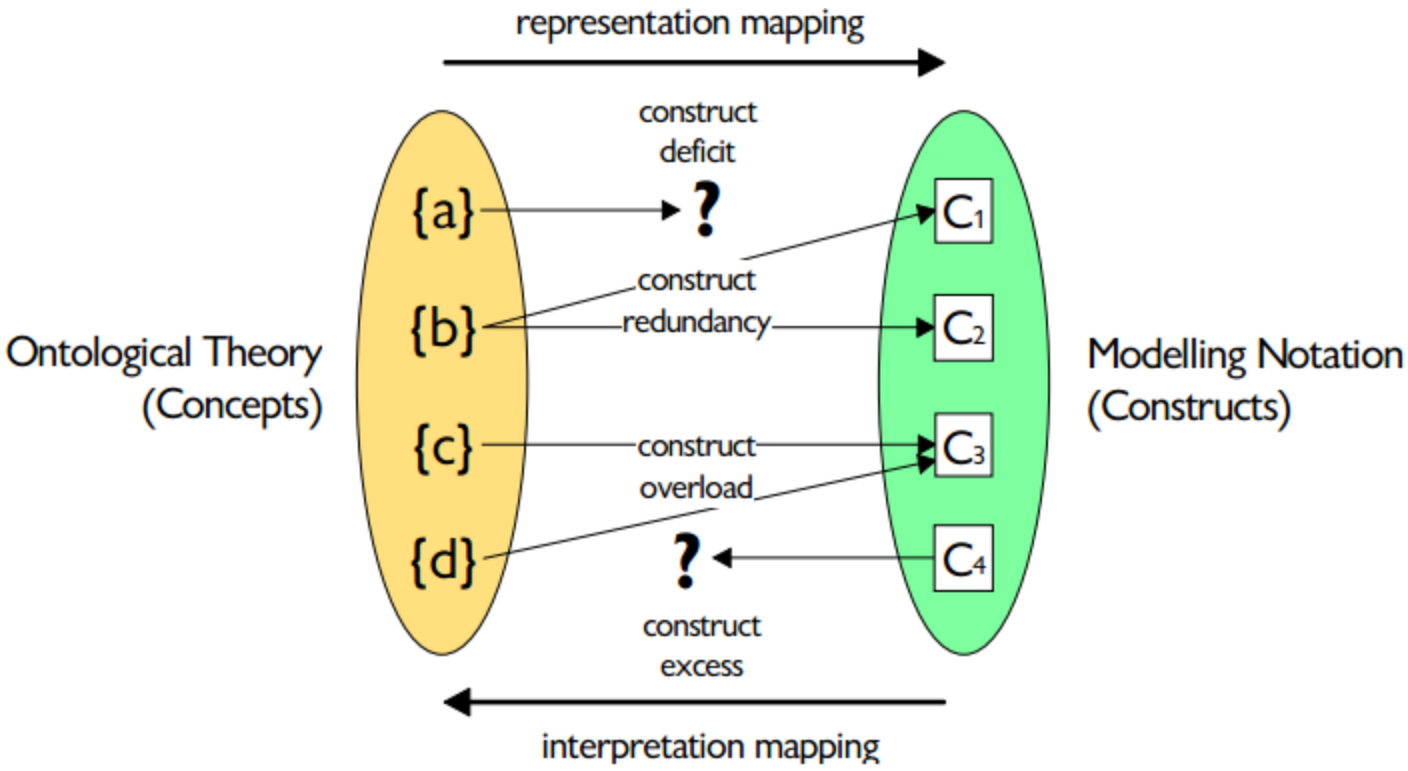
Ontology-Driven Conceptual Modeling

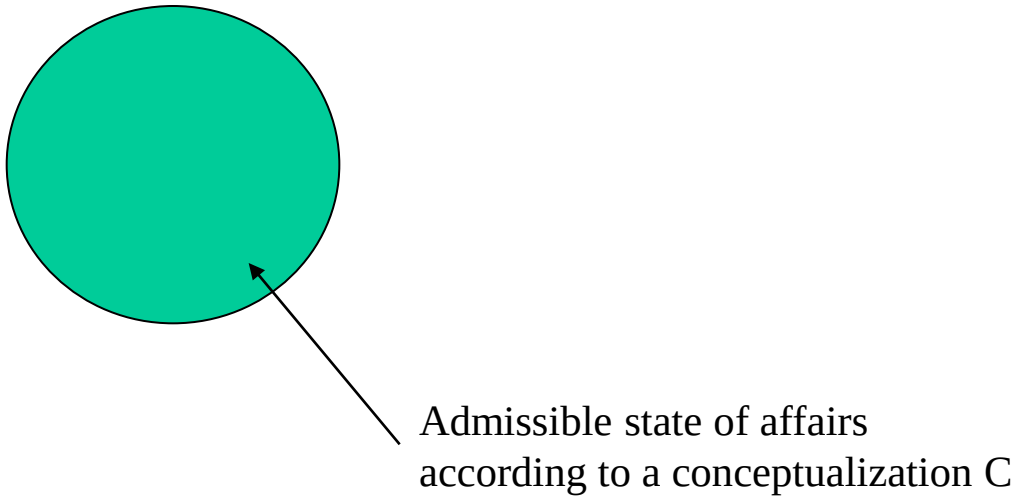
Giancarlo Guizzardi
Computer Science Department
Federal University of Espírito Santo (UFES), Brazil

IAOA Summer School
Trento, Italy
Day 2

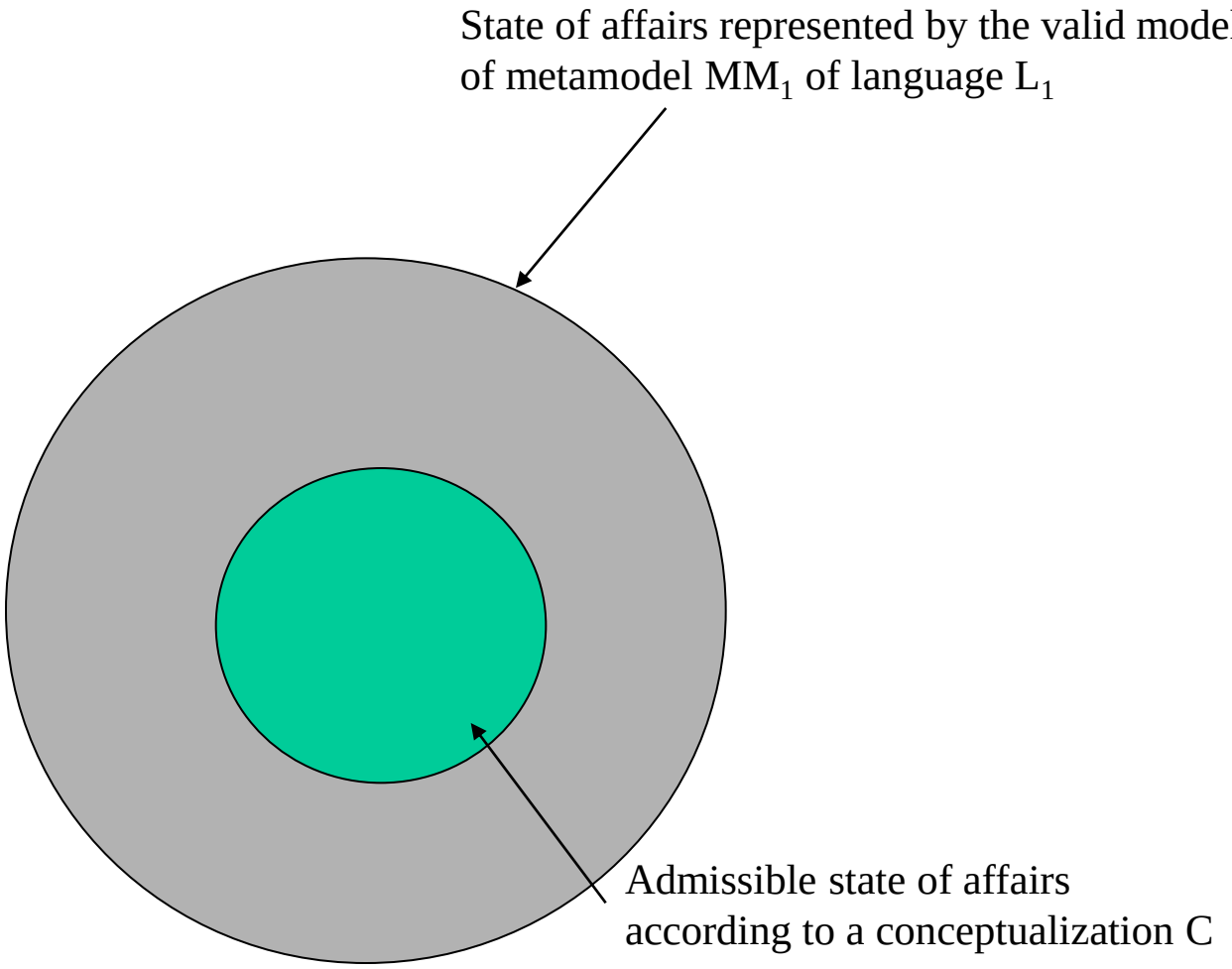

Domain
Conceptualization
represented by
interpreted as
Modeling
Language
used to
compose
instance of
used to
compose
Domain
Abstraction
represented by
interpreted as
Model



Picture by Daniel Moody

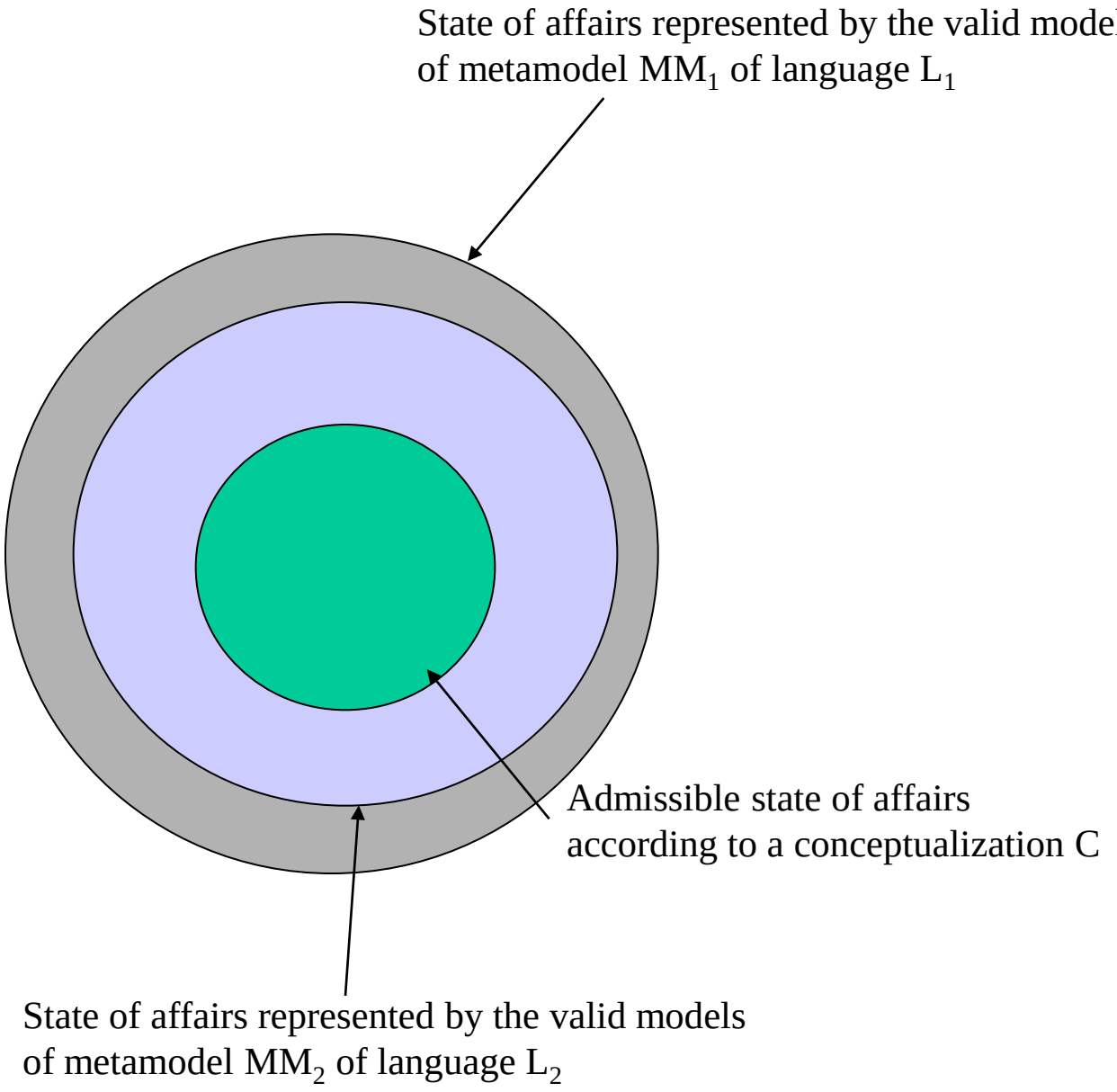

Admissible state of affairs
according to a conceptualization C

{∃x Person(x), ∃x Father(x)} (MM_1)


State of affairs represented by the valid models of metamodel MM_1 of language L_1
Admissible state of affairs according to a conceptualization C

$\{\exists x\ \text{Person}(x), \exists x\ \text{Father}(x)\}$ (MM_1)

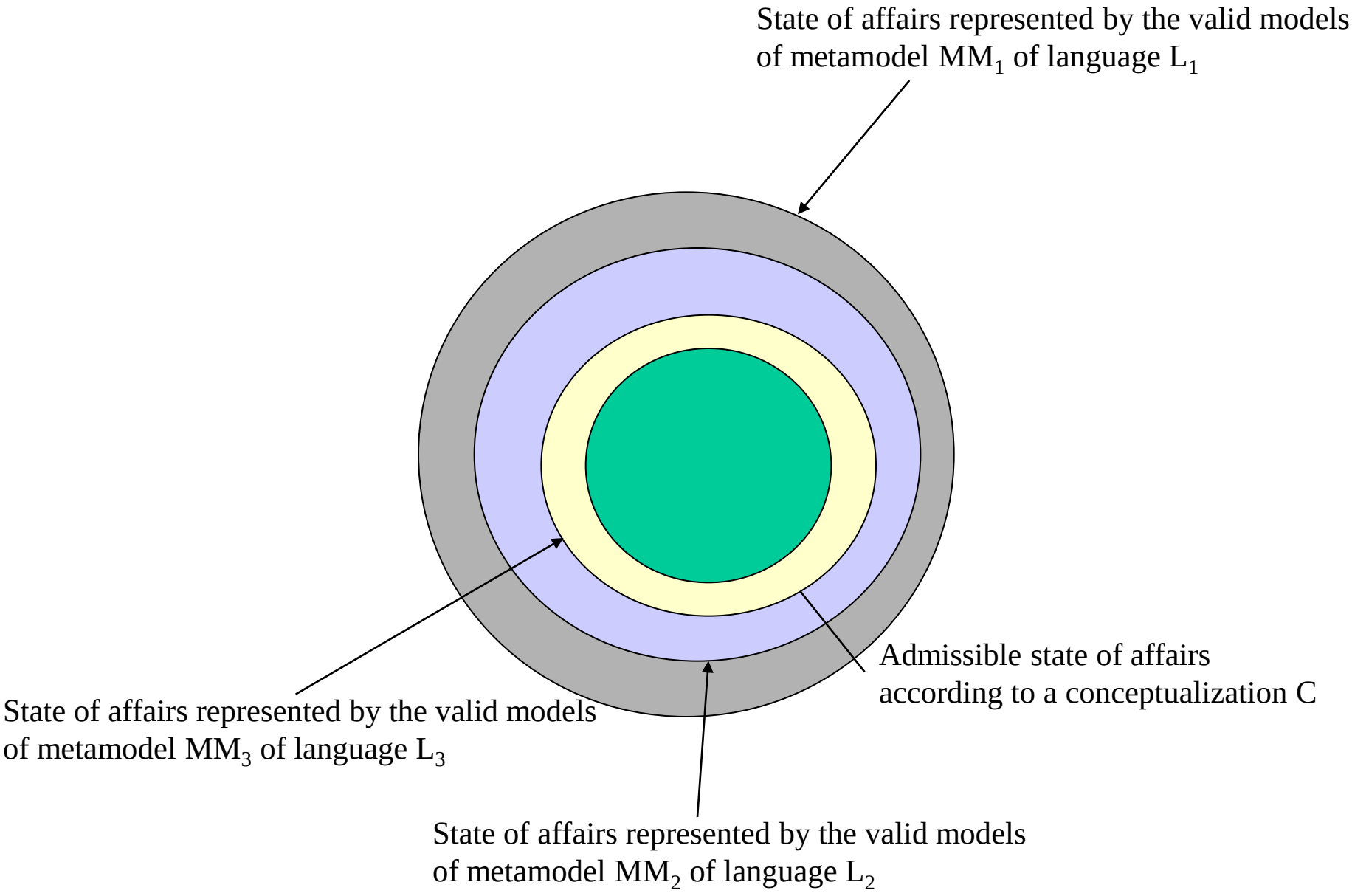
$\{\exists x\ \text{Person}(x), \exists x\ \text{Father}(x), \forall x\ \text{Father}(x) \rightarrow \text{Man}(x), \forall x\ \text{Person}(x) \leftrightarrow \text{Man}(x) \vee \text{Woman}(x), \neg\exists x\ \text{Man}(x) \wedge \text{Woman}(x), \ldots\}$ (MM_2)

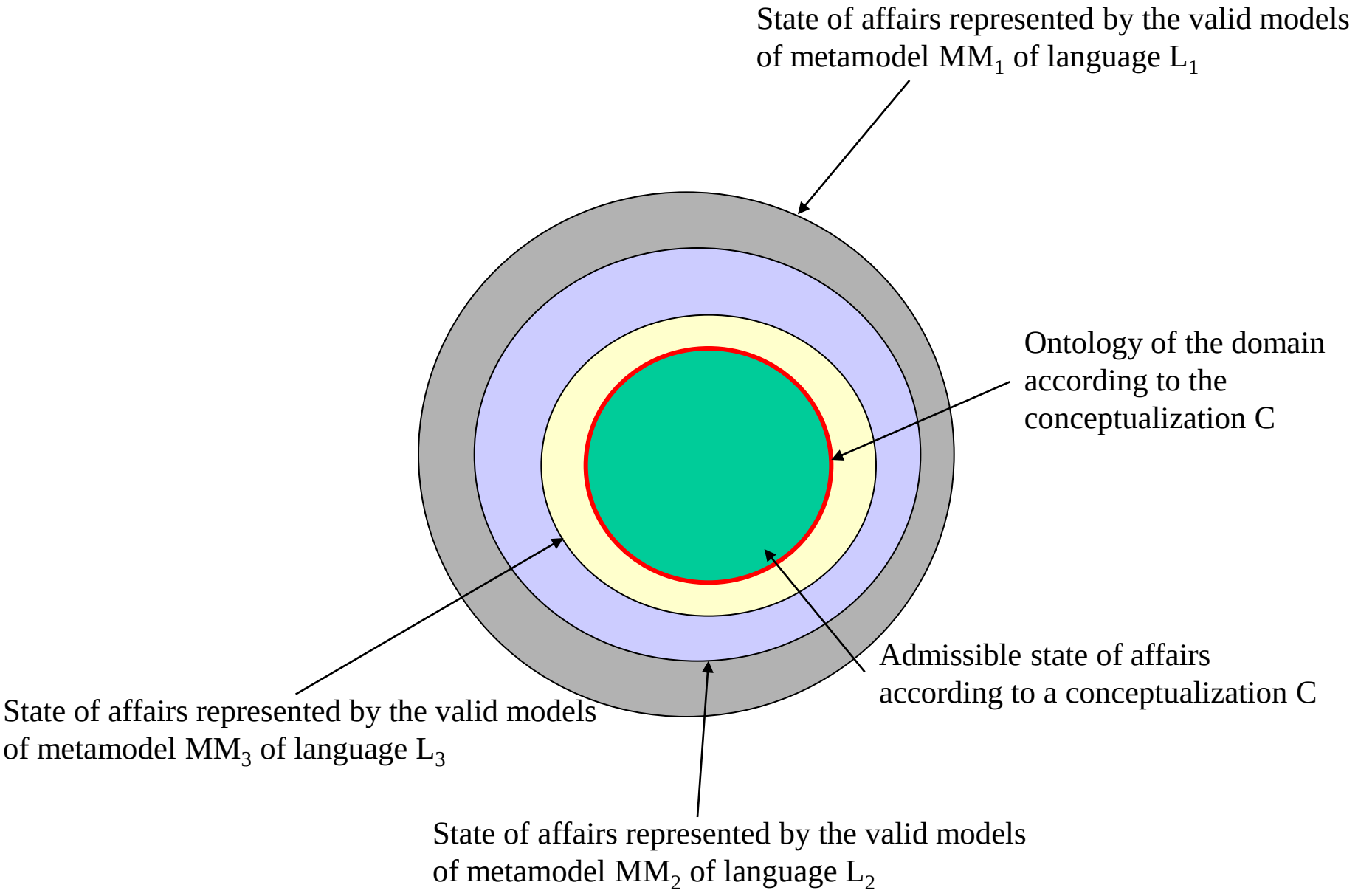

State of affairs represented by the valid models of metamodel MM_1 of language L_1
Admissible state of affairs according to a conceptualization C
State of affairs represented by the valid models of metamodel MM_2 of language L_2

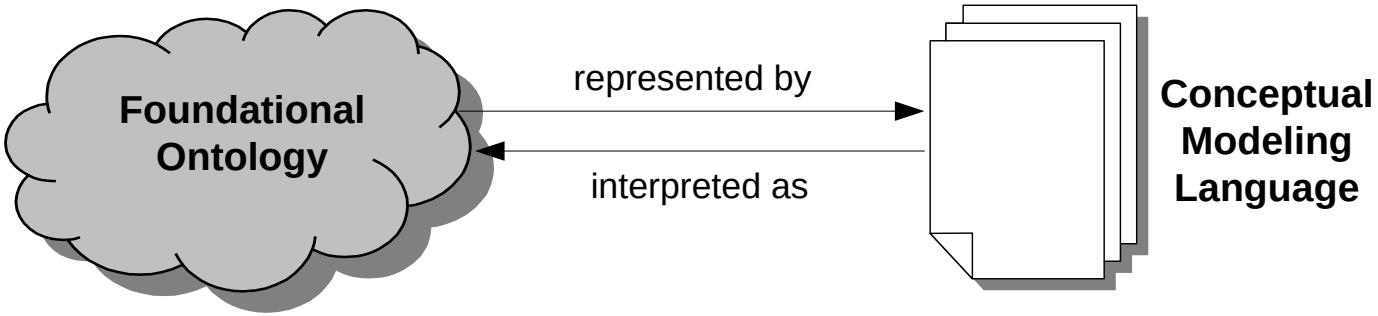
$\{\exists x\ Person(x), \exists x\ Father(x)\}$ (MM_1)

$\{\exists x\ Person(x), \exists x\ Father(x), \forall x\ Father(x) \rightarrow Man(x), \forall x\ Person(x) \leftrightarrow Man(x) \vee Woman(x), \neg\exists x\ Man(x) \wedge Woman(x),\ldots\}$ (MM_2)

$\{\exists x\ Person(x), \exists x\ Father(x), \forall x\ Father(x) \rightarrow Man(x), \forall x\ Person(x) \leftrightarrow Man(x) \vee Woman(x), \neg\exists x\ Man(x) \wedge Woman(x), \forall x\ Person(x) \rightarrow \Box Person(x), \forall x\ LivingPerson(x) \rightarrow \Diamond\neg LivingPerson(x)\ldots\}$ (MM_3)


State of affairs represented by the valid models of metamodel MM_1 of language L_1
Admissible state of affairs according to a conceptualization C
State of affairs represented by the valid models of metamodel MM_3 of language L_3
State of affairs represented by the valid models of metamodel MM_2 of language L_2

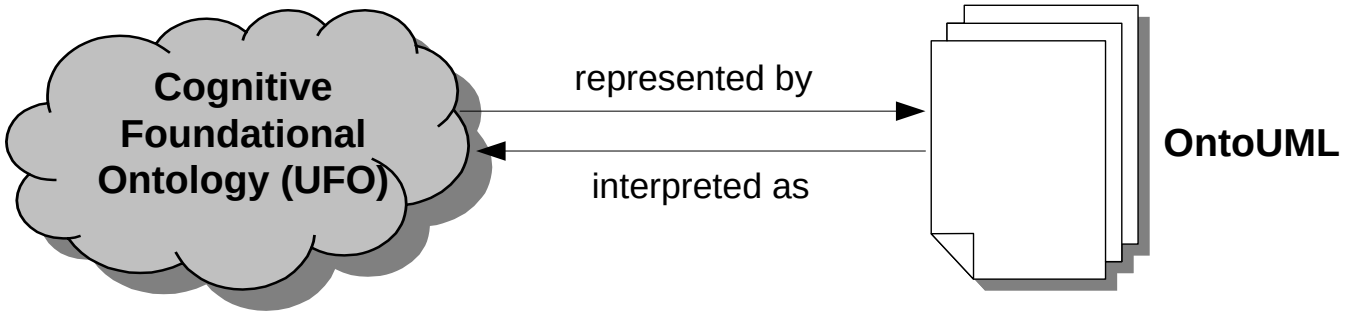

State of affairs represented by the valid models of metamodel MM_1 of language L_1
Ontology of the domain according to the conceptualization C
Admissible state of affairs according to a conceptualization C
State of affairs represented by the valid models of metamodel MM_3 of language L_3
State of affairs represented by the valid models of metamodel MM_2 of language L_2


Foundational Ontology
represented by
interpreted as
Conceptual Modeling Language

Formal Ontology

nemo

- To uncover and analyze the general categories and principles that describe reality is the very business of philosophical Formal Ontology
- Formal Ontology (Husserl): a discipline that deals with formal ontological structures (e.g. theory of parts, theory of wholes, types and instantiation, identity, dependence, unity) which apply to all material domains in reality.

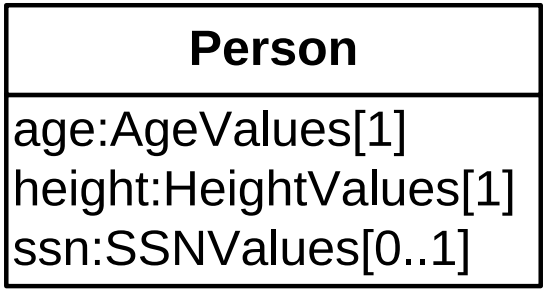

Cognitive
Foundational
Ontology (UFO)
represented by
interpreted as
OntoUML

nemo

A WHIRLWIND INTRODUCTION TO THE UML 2.0 CLASS FRAGMENT

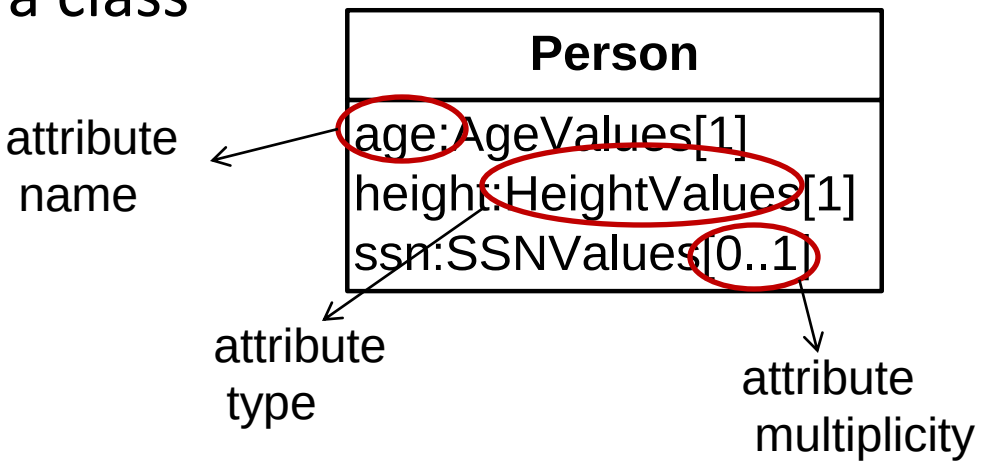
Classes and Attributes

- A class describes the common features (e.g., intrinsic and relational properties) shared by a (possible multitude of) entities which are then said to be the instances of that class
- Instances of a class must contain values for each attribute that is defined for that class, in accordance with the characteristics of the attribute, for example its type and multiplicity.
- Attributes represent (more or less intrinsic) properties of shared by members of a class



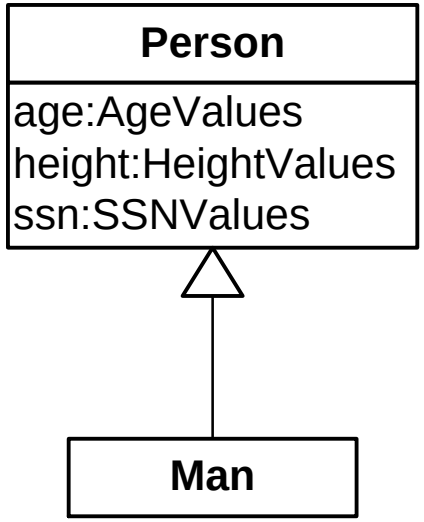
Classes and Attributes

- A class describes the common features (e.g., intrinsic and relational properties) shared by a (possible multitude of) entities which are then said to be the instances of that class
- Instances of a class must contain values for each attribute that is defined for that class, in accordance with the characteristics of the attribute, for example its type and multiplicity.
- Attributes represent (more or less intrinsic) properties of shared by members of a class



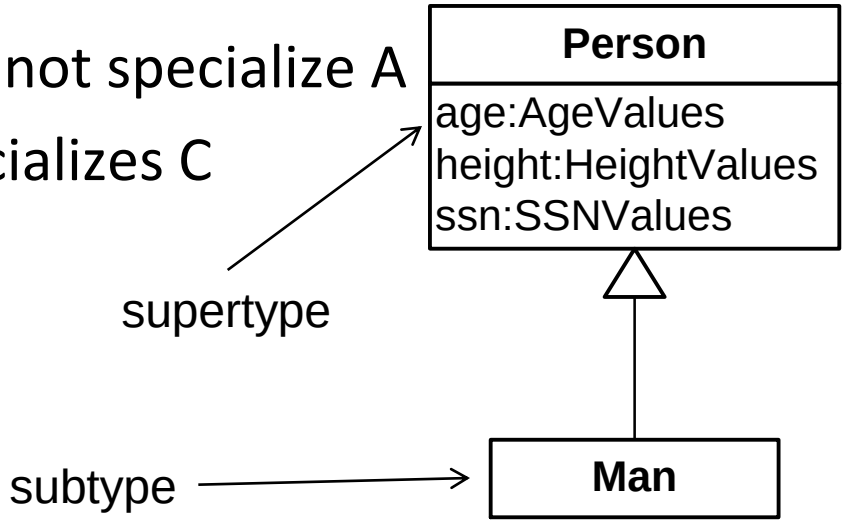
Specialization

- Classes can be related to each other via specialization relations forming a taxonomic structure
- All properties of a class are inherited through a specialization chain
- A class can be a specialization of multiple classes
- The specialization relation is a anti-symmetric and transitive relation, i.e.,
  - If A specializes B then B cannot specialize A
  - If A specializes B and B specializes C
  - then A specializes C



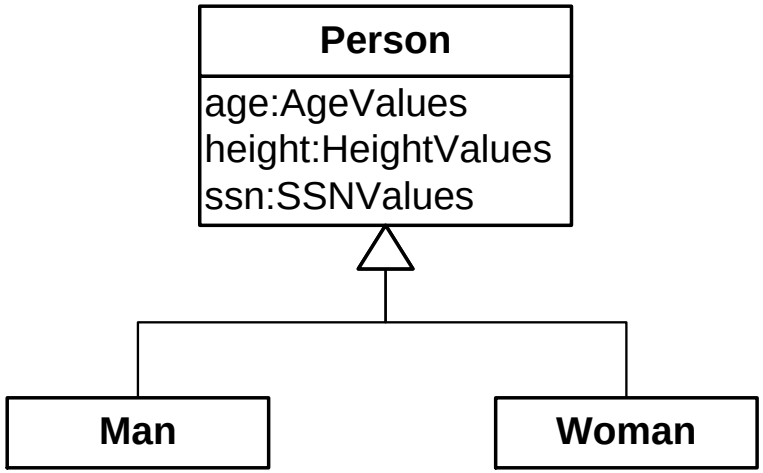
Specialization

- Classes can be related to each other via specialization relations forming a taxonomic structure
- All properties of a class are inherited through a specialization chain
- A class can be a specialization of multiple classes
- The specialization relation is a anti-symmetric and transitive relation, i.e.,
  - If A specializes B then B cannot specialize A
  - If A specializes B and B specializes C
  - then A specializes C



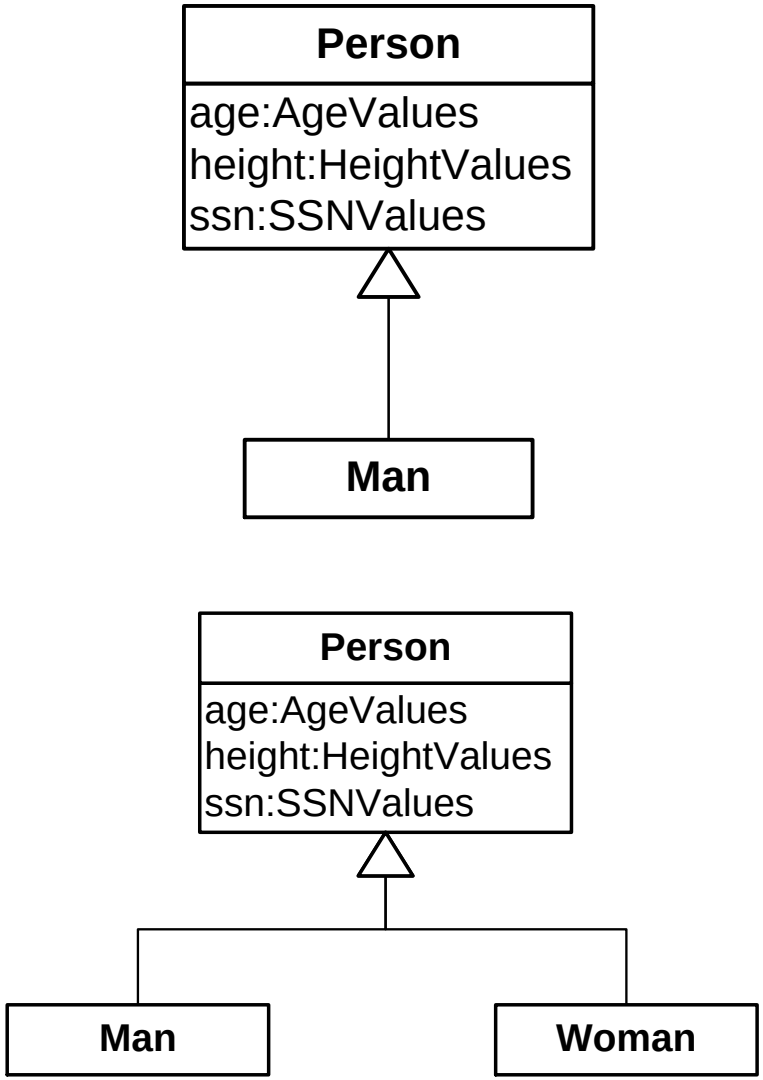
Generalization Set

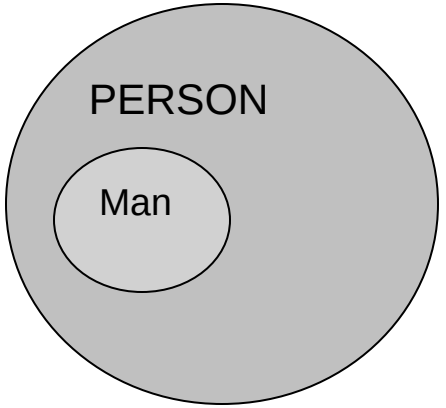
- Classes sharing a common direct supertype can be group in what is called a generalization set

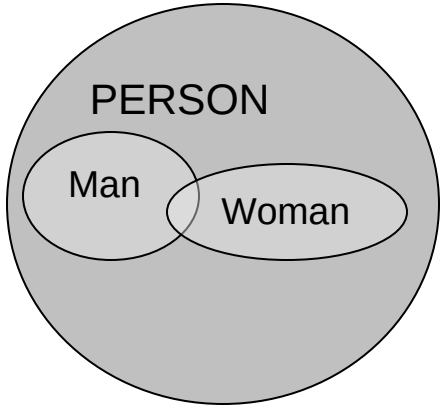


- There are two meta-attributes that can be used ascribe a stronger semantics to a generalization set, namely, the complete and disjoint meta-attributes

Specialization (Extensional Perspective)

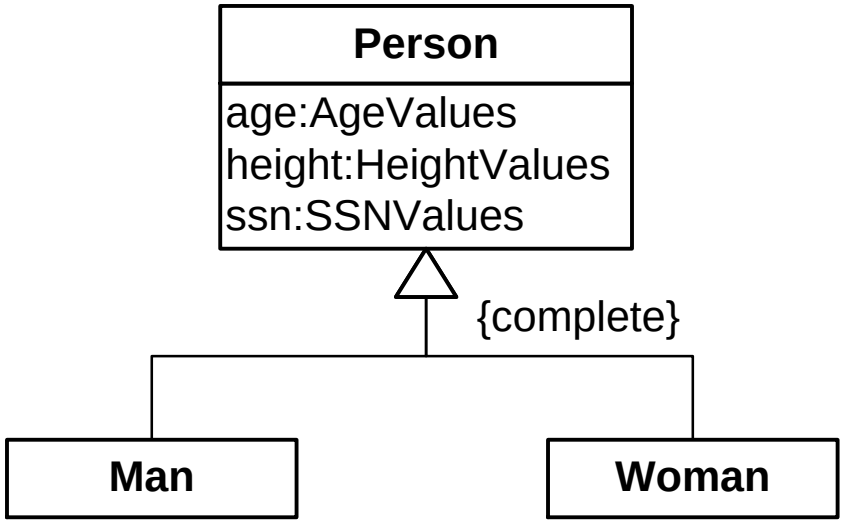


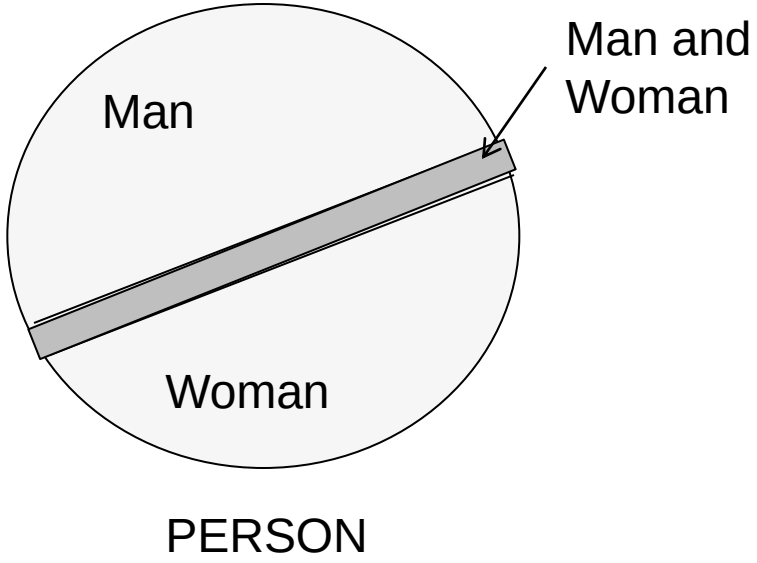




Complete

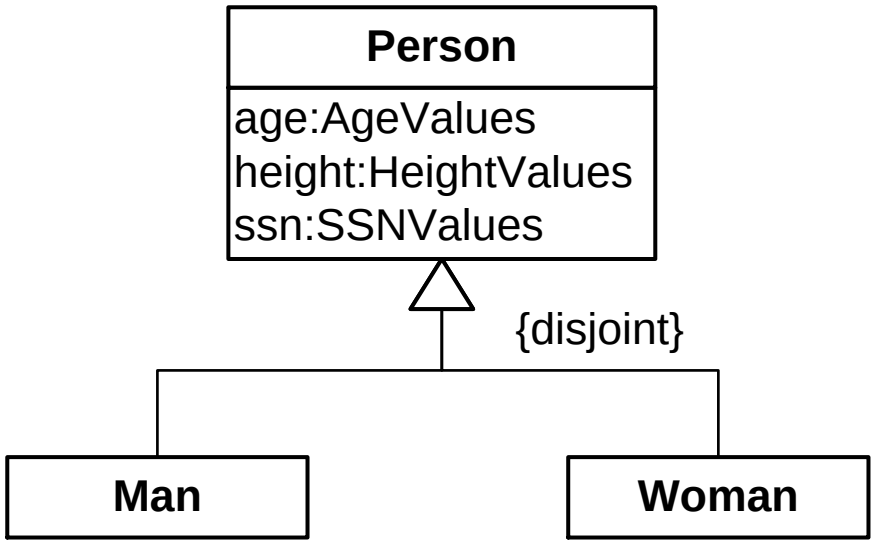
- If a generalization set is complete then the subclasses exhaust the instances of the common direct superclass.
- In other words, in this case, there is no instance of the superclass which is not an instance of one of the subclasses participating in the generalization

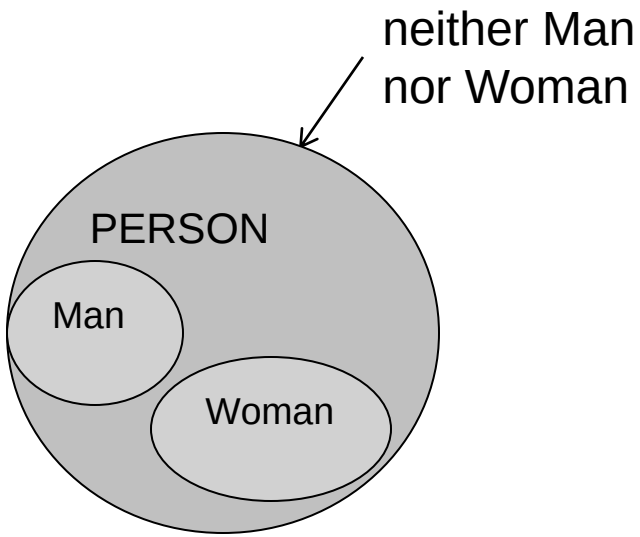




Disjoint

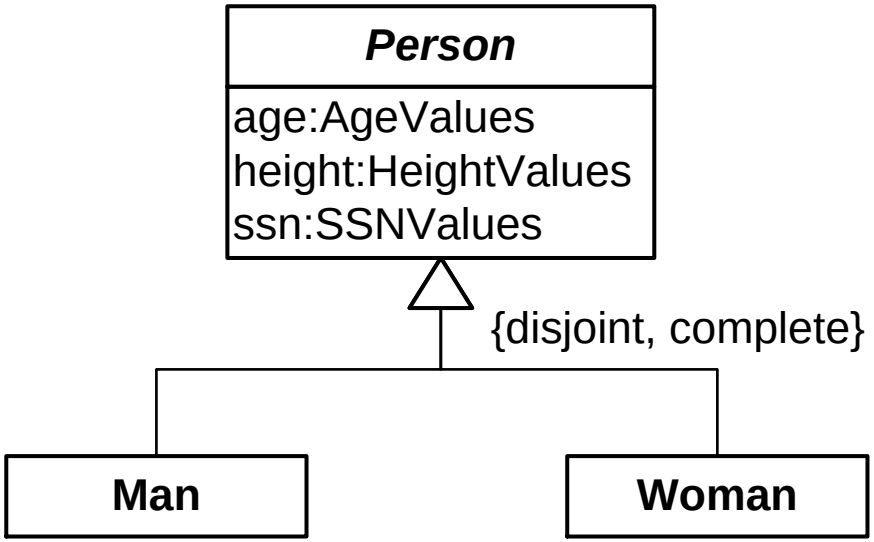
- If a generalization set is disjoint if all the subclasses participating in the generalization are mutually exclusive
- In other words, the intersection between these any of these subclasses pairwise is necessarily empty

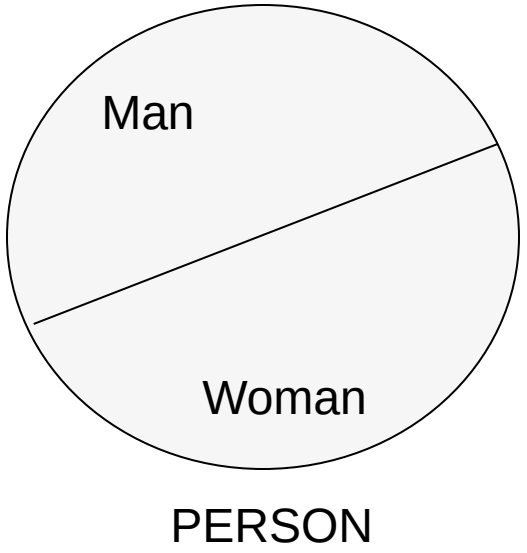




Disjoint and Complete

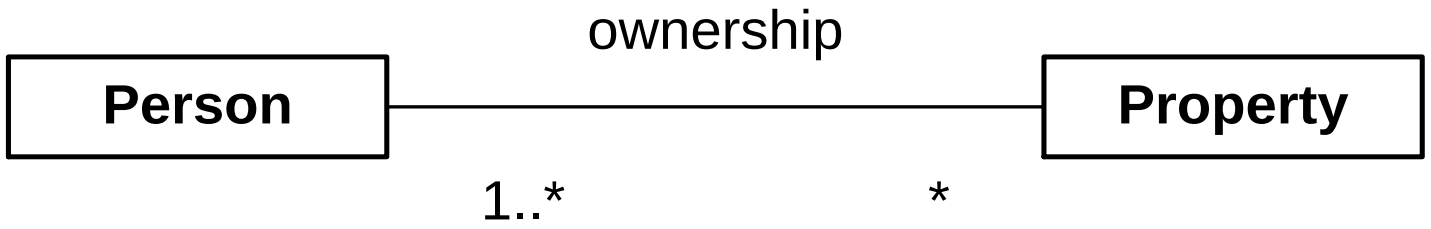
- If a generalization set is disjoint and complete then the subclasses participating in this generalization set form a partition.
- In other words, every instance of the common superclass is an instance of one and only one of the subclasses
- In this case, the common superclass is an abstract class





Associations

- An association declares that there can be links between instances of the associated types. A link is a tuple with one value for each end of the association, where each value is an instance of the type of the end.



Associations

- One specify multiplicity constraints for each end of the association

- The possibilities for cardinality constraints are:

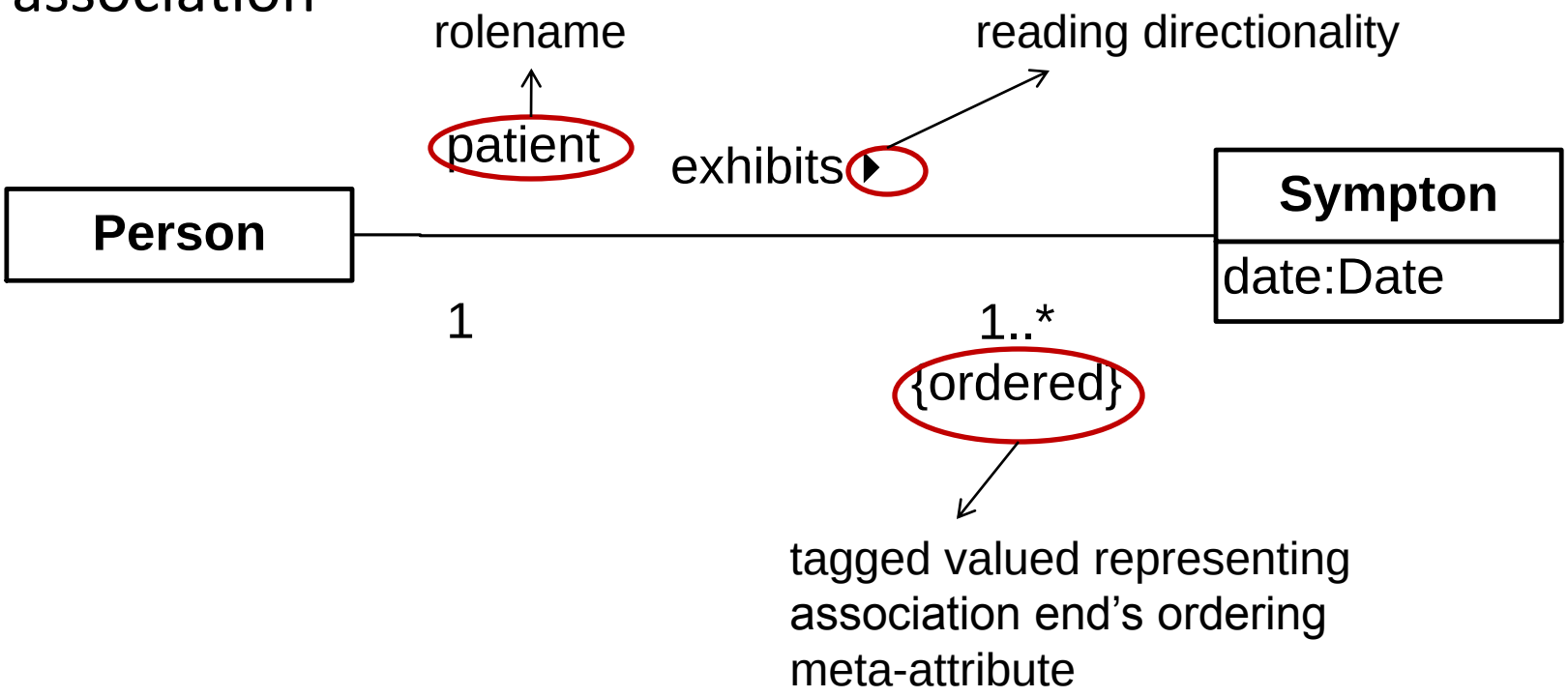
Min	Max	UML notation
0	1	0..1
0	n (n> 1 indefinido)	*
1	1	1
1	n (n> 1 indefinido)	1..*

Associations

- When one or more ends of the association are ordered, links carry ordering information in addition to their end values.
- To each association end, one can specify a rolename definying the “role played by objects of that type in the association”

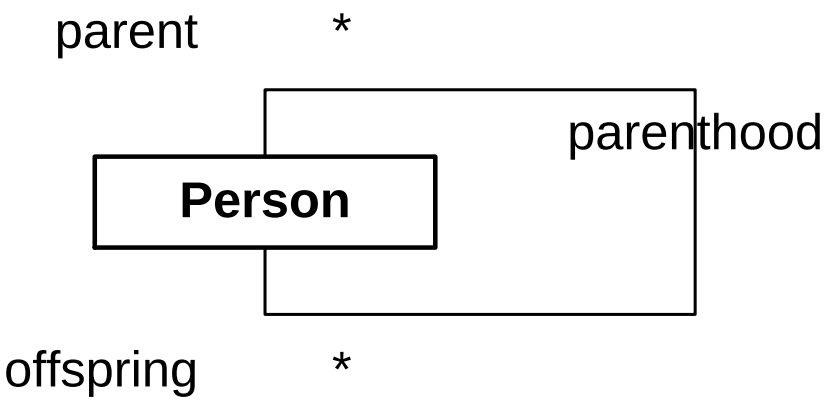
Associations

- When one or more ends of the association are ordered, links carry ordering information in addition to their end values.
- To each association end, one can specify a rolename defiying the “role played by objects of that type in the association”



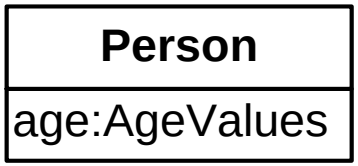
Associations

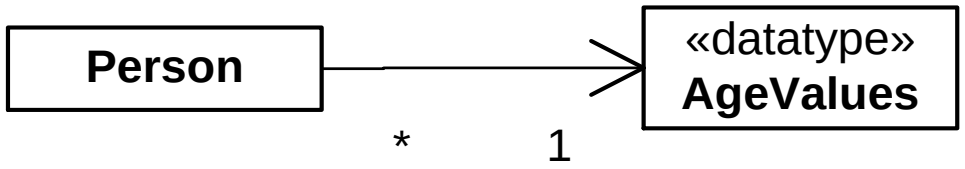
- One can define type-reflexive associations, i.e., associations in which both association ends are connected to the same type



Datatypes

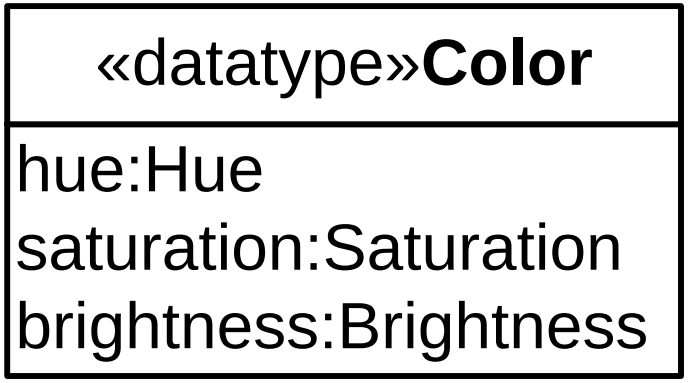
- A Datatype represents the set of possible values that an attribute can take.
- If attributes are seen as functions (mapping the extension of a class to a datatype) than they are the range of that function
- Datatypes have no instances in the real sense, they are simply sets of values. They have members which are abstract individuals, i.e., they are not explicitly created or destroyed but are simply assumed to exist



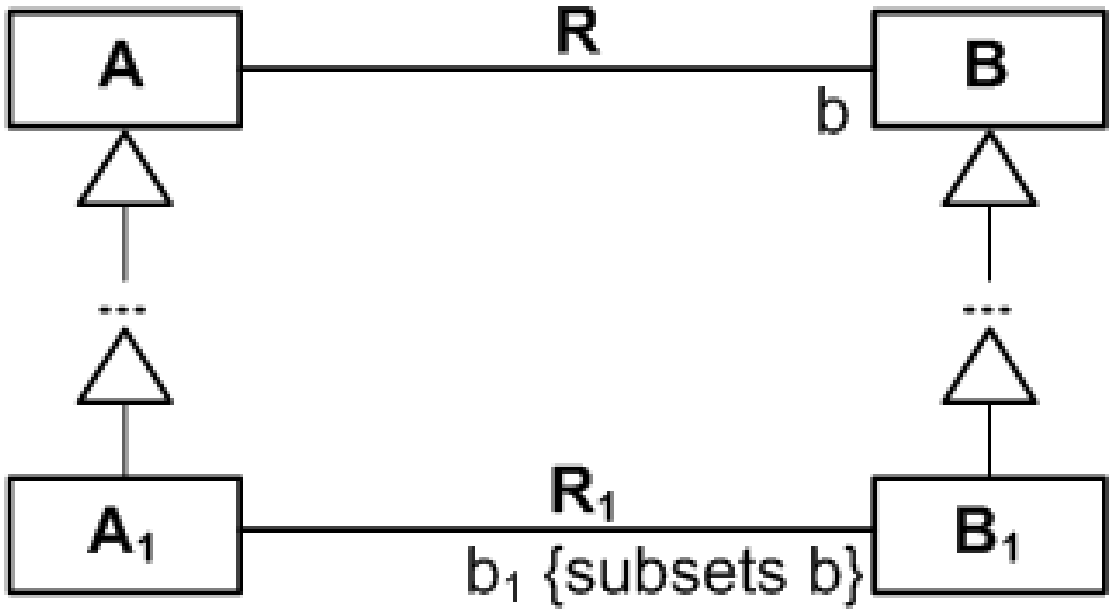


Datatypes

- There are simple and structured datatype. The “attributes” of a structured datatype are named datatype fields
- The members of a datatype with n fields are n-uples. For instance, the members of the color datatype below are triples <h,s,b> of hue, saturation and brightness values

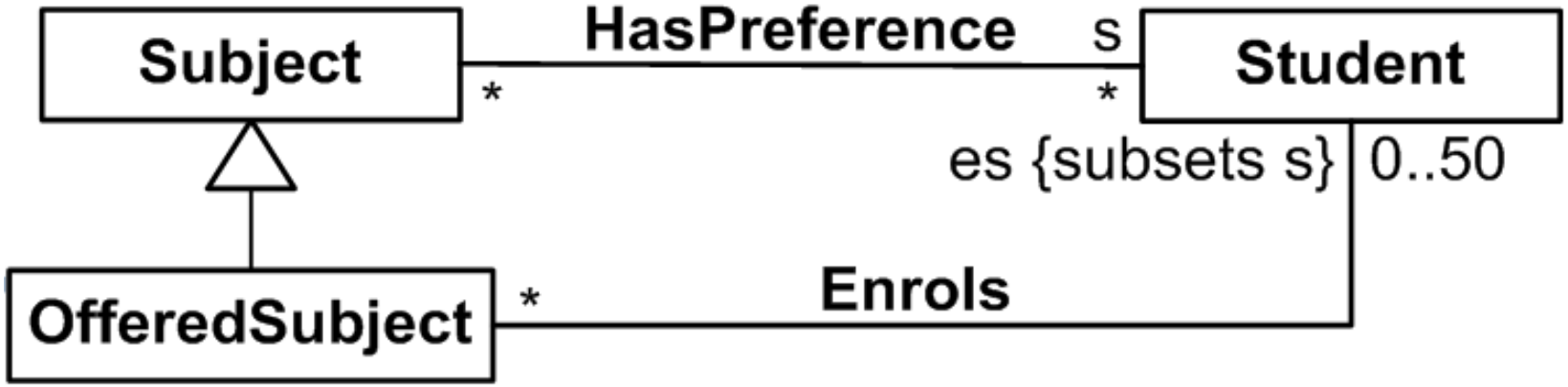


Subsetting



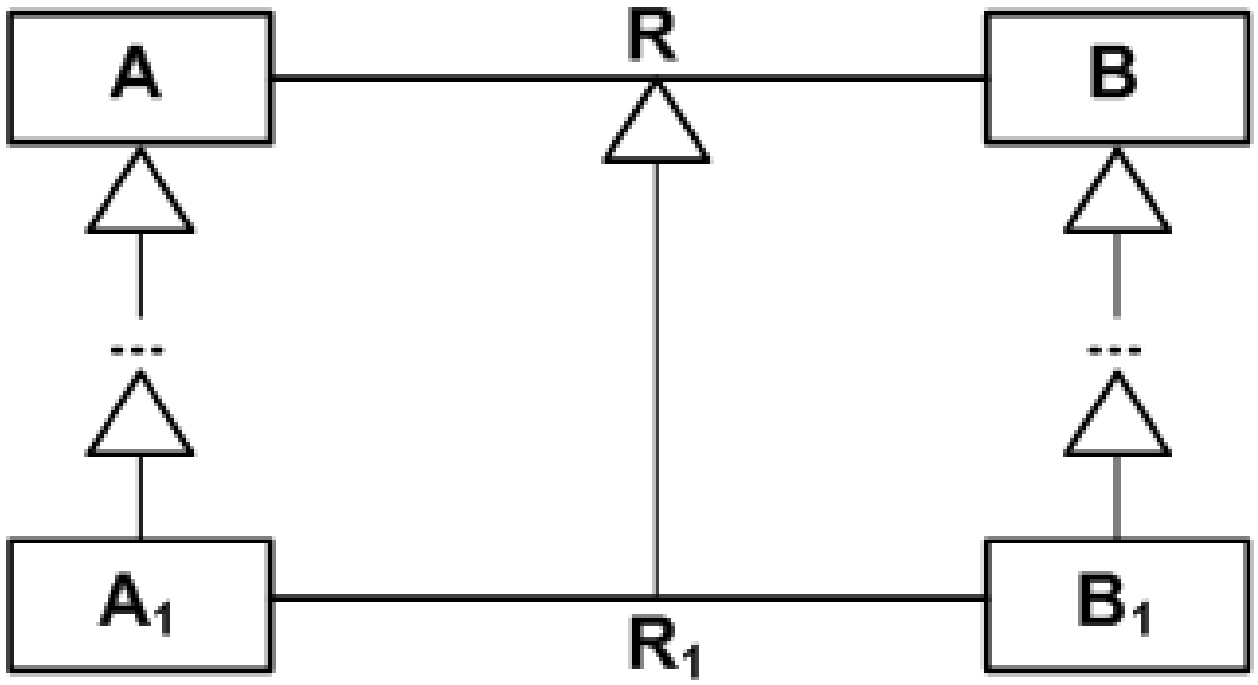
where A_1 may be the same class as A
and/or B_1 may be the same class as B
and/or A may be the same class as B (recursive case)

Subsetting



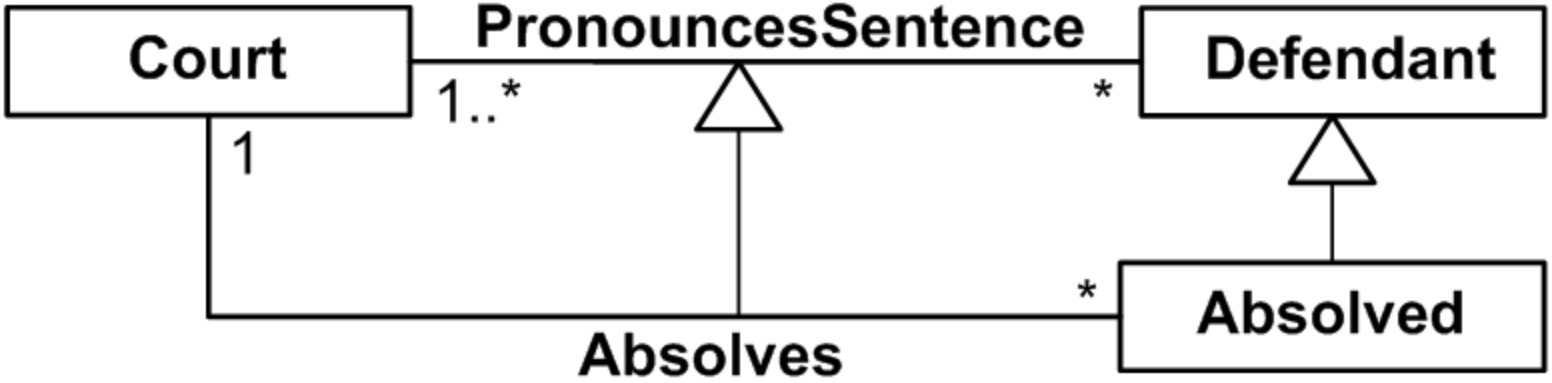
*Not necessarily defending the modeling choices in the following slides

Association Specialization

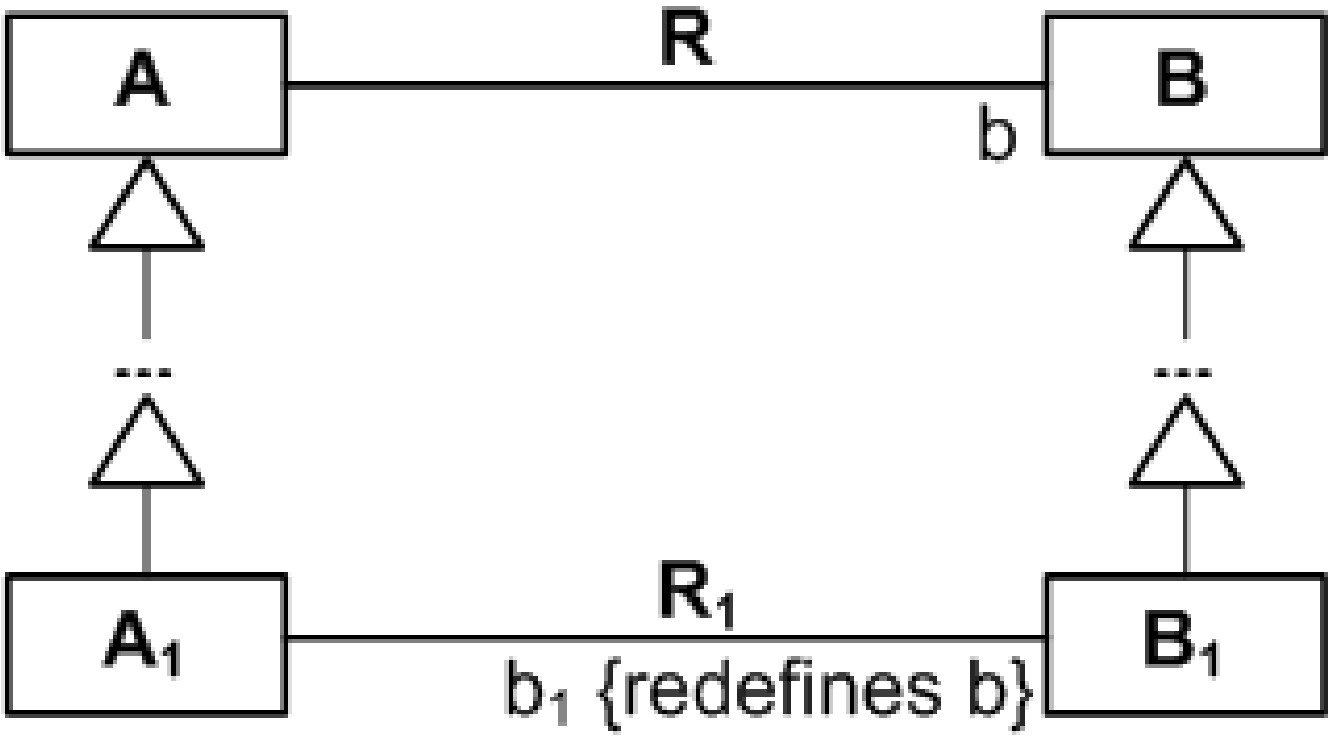


where A_1 may be the same class as A
and/or B_1 may be the same class as B
and/or A may be the same class as B (recursive case)

Association Specialization

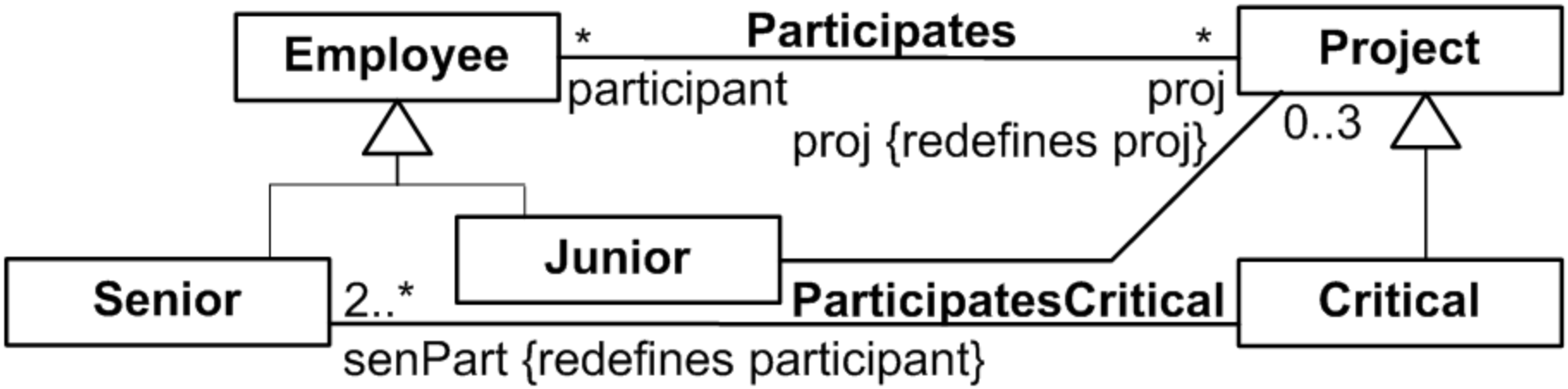


Association Redefinition



where B_1 may be the same class as B
and/or A may be the same class as B (recursive case)
and/or b_1 name may be the same as b name

Association Redefinition



Modal Logics

- For this presentation, I will use the simplest system of quantified alethic model logics (QS5);
- The model operators are □ **(necessity)** and ◊ **(possibility)**
- The accessability relation is considered to be universal (all worlds are equally accessible);
  - **□A iff in every possible world w, A holds**
  - **◊ A iff there is a possible world w in which A holds**

CATEGORIES OF OBJECT TYPES

General Terms and Common Nouns

- (i) exaclty five mice were in the kitchen last night
- (ii) the mouse which has eaten the cheese, has been in turn eaten by the cat

General Terms and Common Nouns

- (i) exactly five X ...
- (ii) the Y which is Z...

General Terms and Common Nouns

- (i) exaclty five *reds* were in the kitchen last night
- (ii) the *red* which has ..., has been in turn ...

General Terms and Common Nouns

- Both reference and quantification require that the thing (or things) which are refered to or which form the domain of quantification are determinate individuals, i.e. their conditions for individuation and numerical identity must be determinate

Sortal and Characterizing Types

- Whilst the characterizing types supply only a principle of application for the individuals they collect, sortal types supply both a principle of application and a principle of identity

CATEGORIES OF OBJECT TYPES

The Logical Level

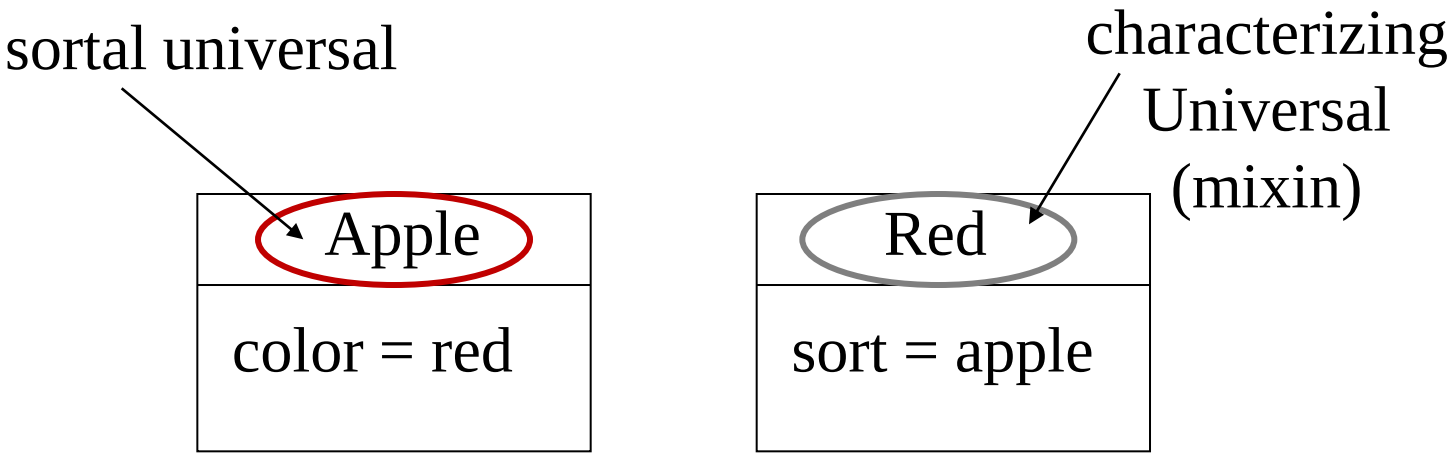
- $\exists x\ \text{Apple}(x) \wedge \text{Red}(x)$

The Epistemological Level

Apple
color = red

Red
sort = apple

The Ontological Level



Foundations

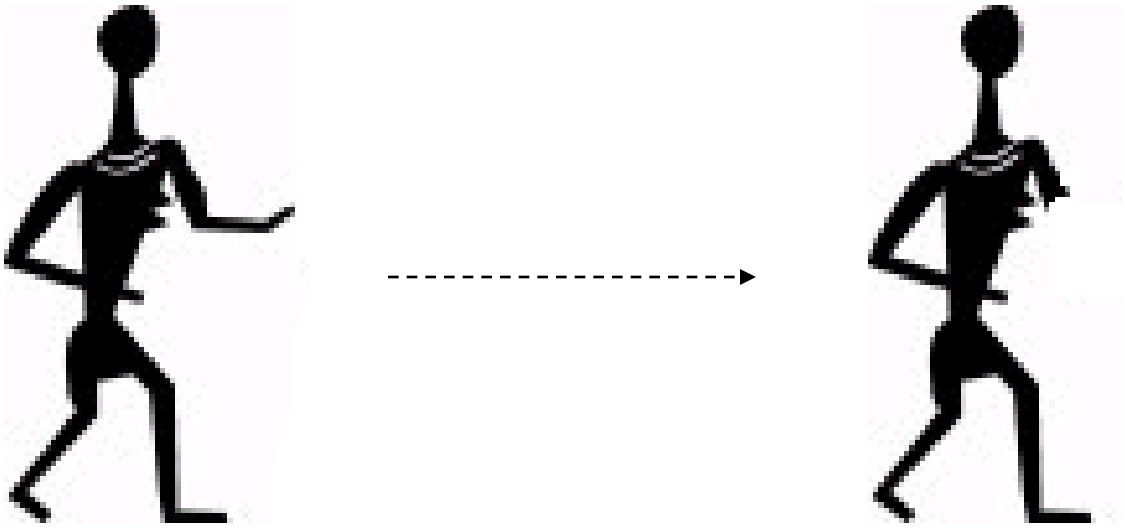
- (1) We can only make identity and identification statements with the support of a Sortal, i.e., the identity of an individual can only be traced in connection with a Sortal type, which provides a principle of individuation and identity to the particulars it collects

Every Object in a conceptual model (CM) of the domain must be an instance of a class representing a sortal type

Foundations

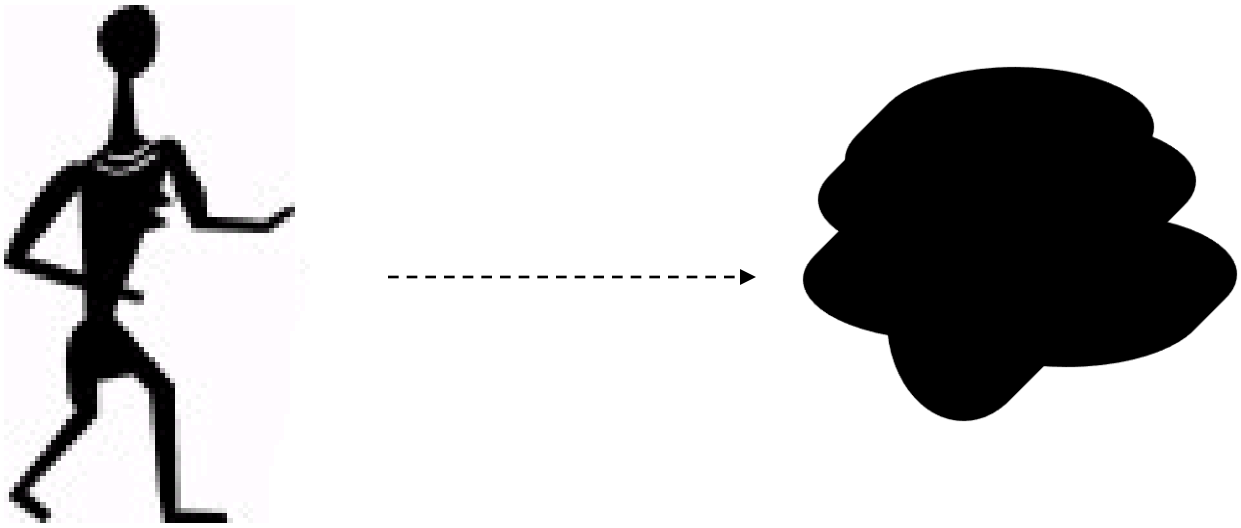
- Moreover, since Non-Sortals cannot supply a principle of identity for its instances, we have that, all Non-Sortal Types in the model must be represented as *Abstract Classes*

Unique principle of Identity



X Y

Unique principle of Identity

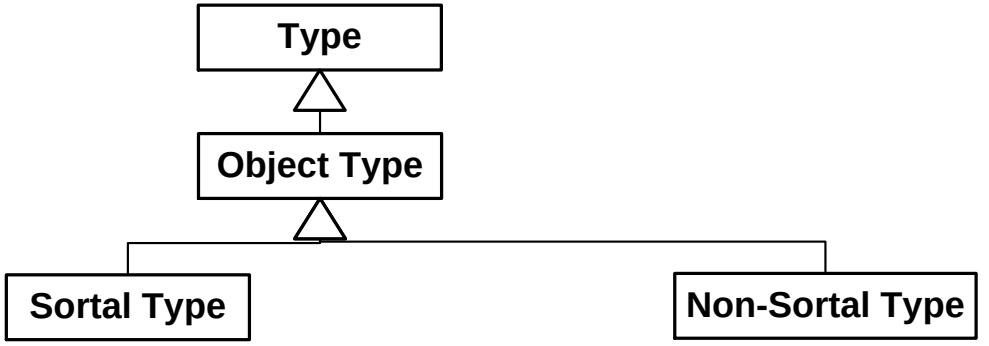


X Y

Foundations

- (2) An individual cannot obey incompatible principles of identity

Distinctions Among Categories of Object Types



{Person, Apple, Student}

{Insurable Item, Red}

Rigidity (R+)

- A type T is rigid if for every instance *x* of T, *x* is necessarily (in the modal sense) an instance of T. In other words, if *x* instantiates T in a given world *w*, then *x* must instantiate T in every possible world *w'*:

$$\mathbf{R+(T) =_{def} \Box(\forall x\ T(x) \rightarrow \Box(T(x)))}$$

e.g.,

$$\mathbf{R+(Person) =_{def} \Box(\forall x\ Person(x) \rightarrow \Box(Person(x)))}$$

Anti-Rigidity (R~)

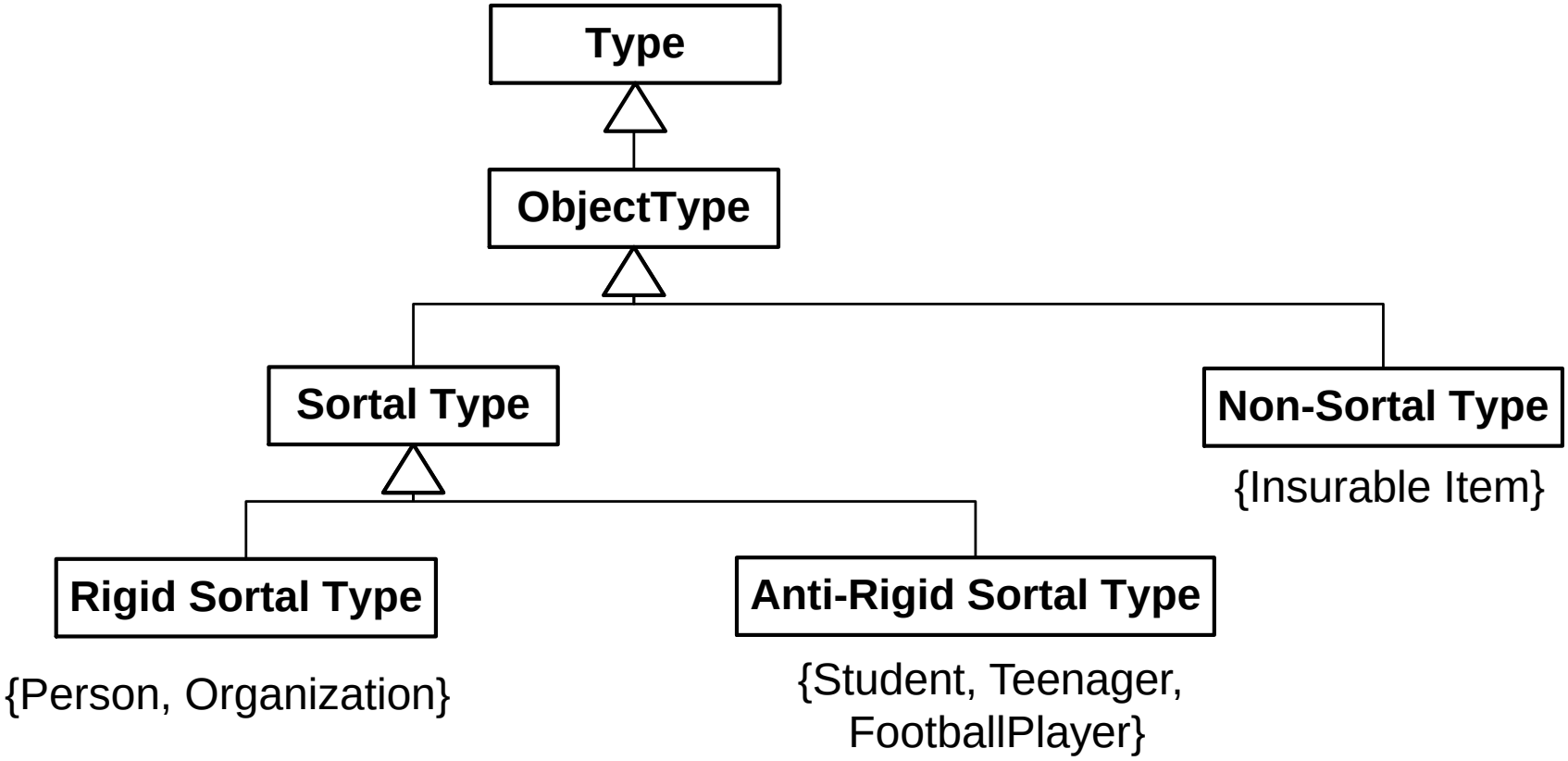
- A types T is anti-rigid if for every instance *x* of T, *x* is *possibly* (in the modal sense) not an instance of T. In other words, if *x* instantiates T in a given world *w*, then there is a possible world *w'* in which *x* does not instantiate T:

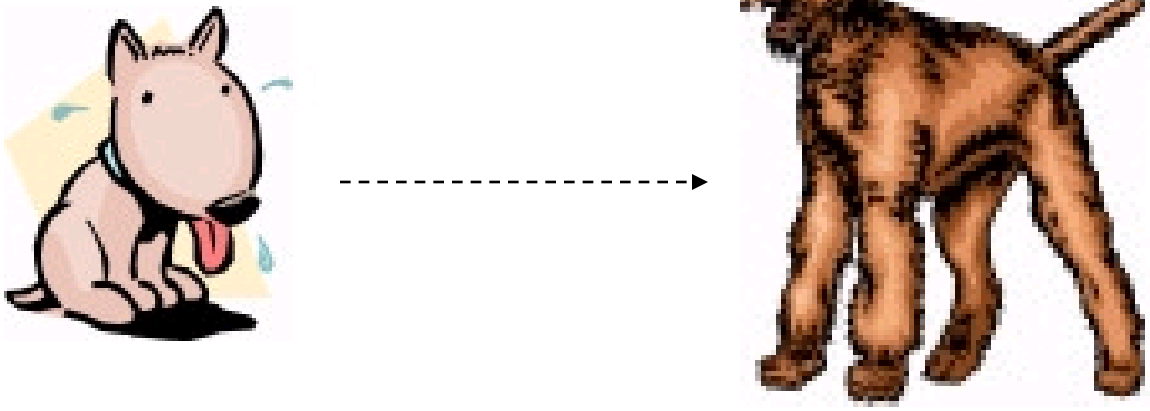
$$\mathbf{R\sim(T) =def \Box(\forall x\ T(x) \rightarrow \Diamond(\neg T(x)))}$$

e.g.,

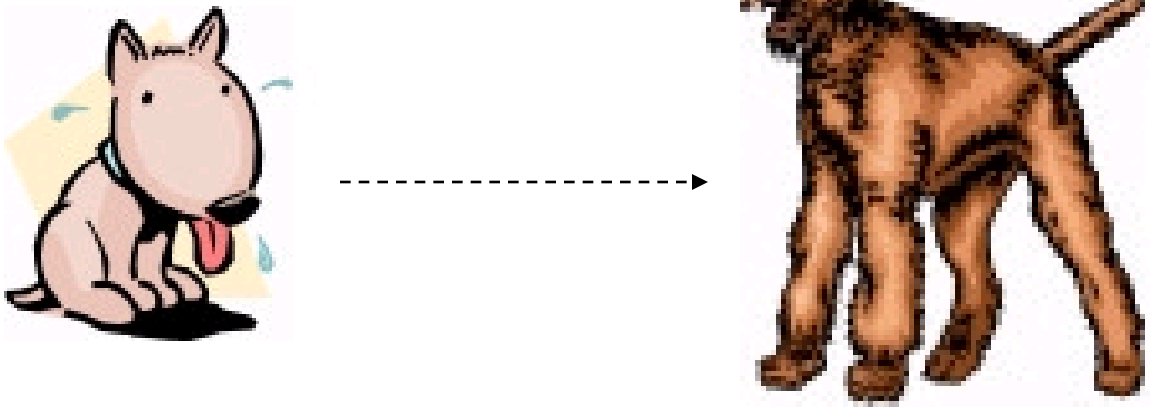
$$\mathbf{R\sim(Student) =_{def} \Box(\forall x\ Student) \rightarrow \Diamond(\neg Student(x)))}$$

Distinctions Among Categories of Object Types



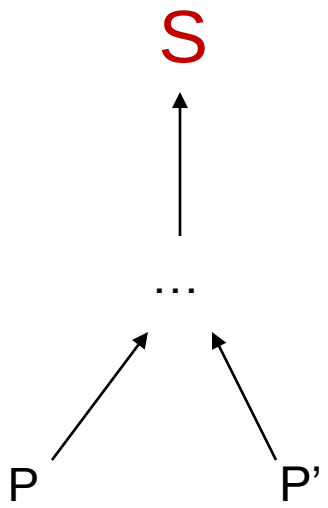


A principle of identity cannot be supplied by either of these two anti-rigid types, since it should be used to identify individuals in every possible situations!

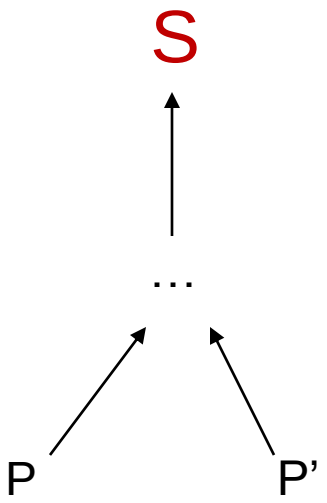


Foundations

- (3) If an individual falls under two sortals in the course of its history there must be exactly one ultimate rigid sortal of which both sortals are specializations and from which they inherit a principle of identity



Restriction Principle

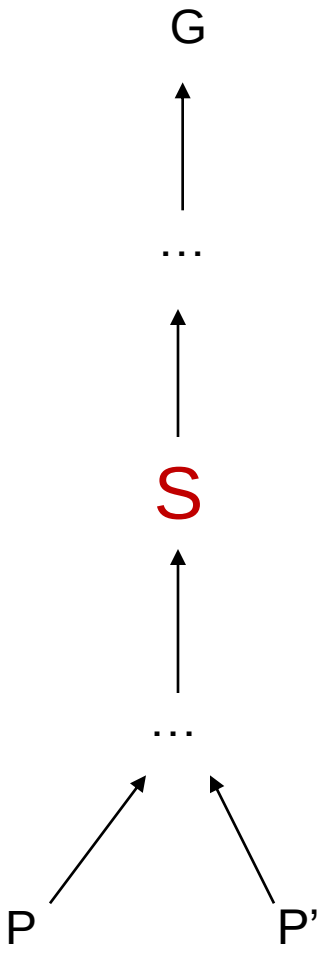


(4) Instances of P and P’ must have obey a principle of identity (by 1)

(5) The principles obeyed by the instances of P and P’ must be the same (by 2)

(6) The common principle of identity cannot be supplied by P neither by P’

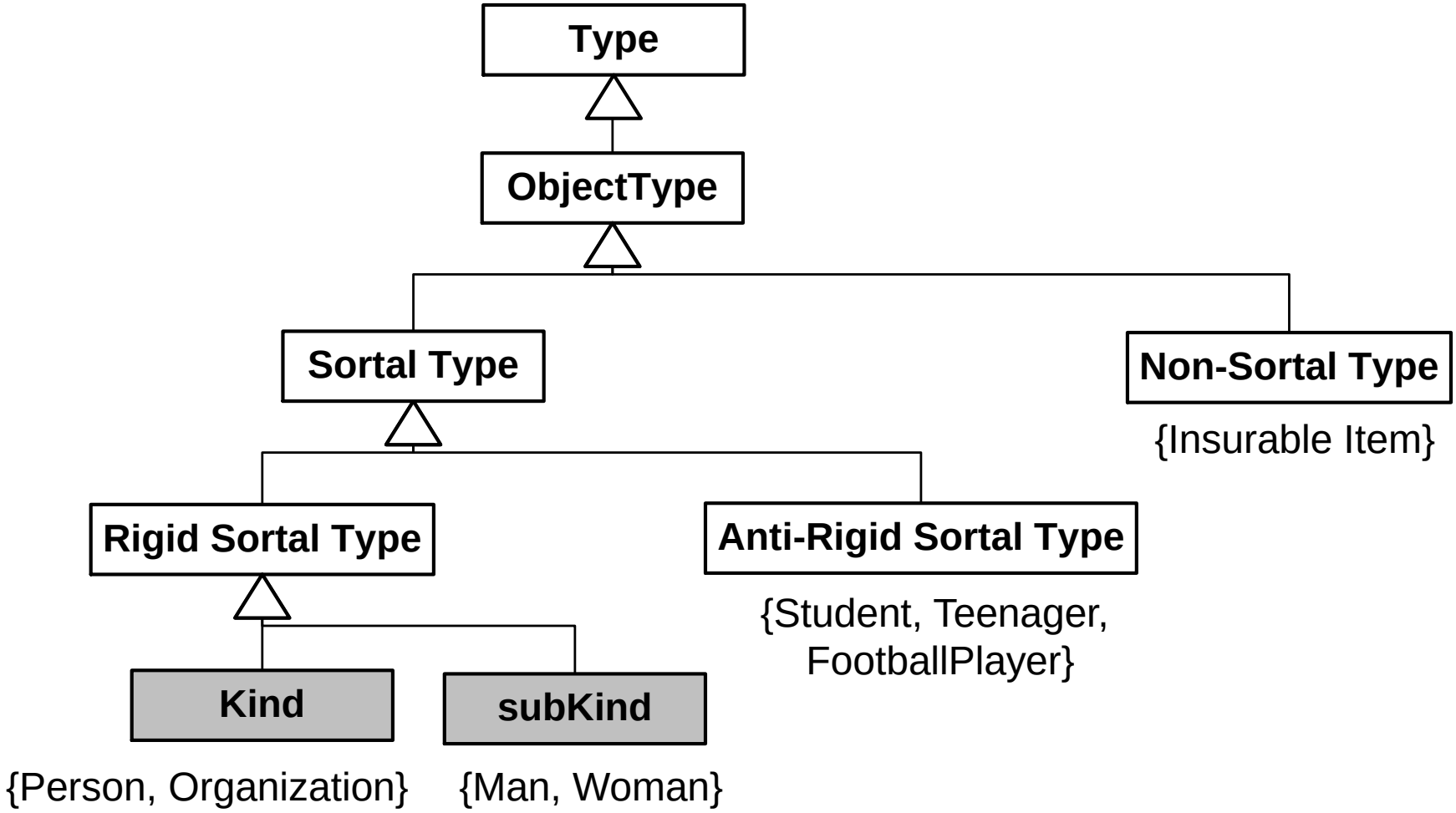
Uniqueness Principle



(7) G and S cannot have incompatible principles of identity (by 2).
Therefore, either:

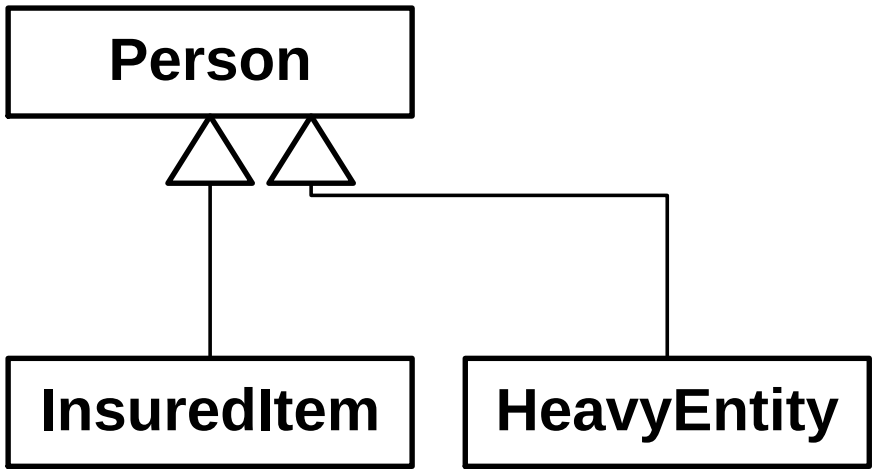
- G supplies the same principle as S and therefore G is the ultimate Sortal
- G is does not supply any principle of identity (non-sortal)

Distinctions Among Categories of Object Types



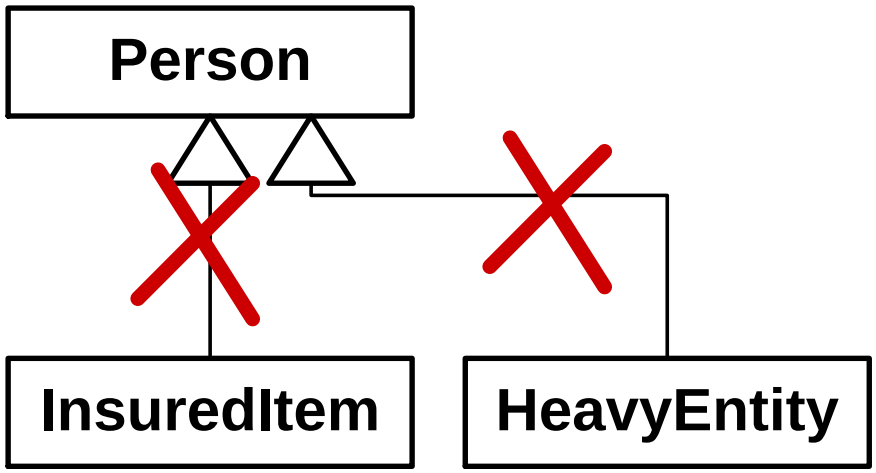
Foundations

- Since the unique principle of identity supplied by a Kind is inherited by its subclasses, we have that:
- A Non-sortal type cannot appear in a conceptual model as a subtype of a sortal



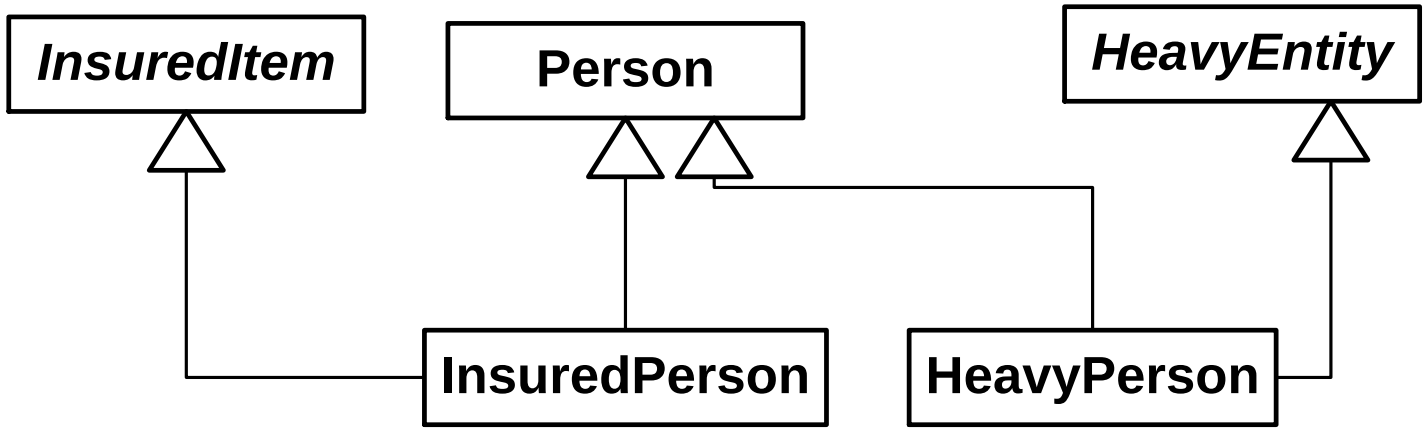
Foundations

- Since the unique principle of identity supplied by a Kind is inherited by its subclasses, we have that:
- A Non-sortal type cannot appear in a conceptual model as a subtype of a sortal



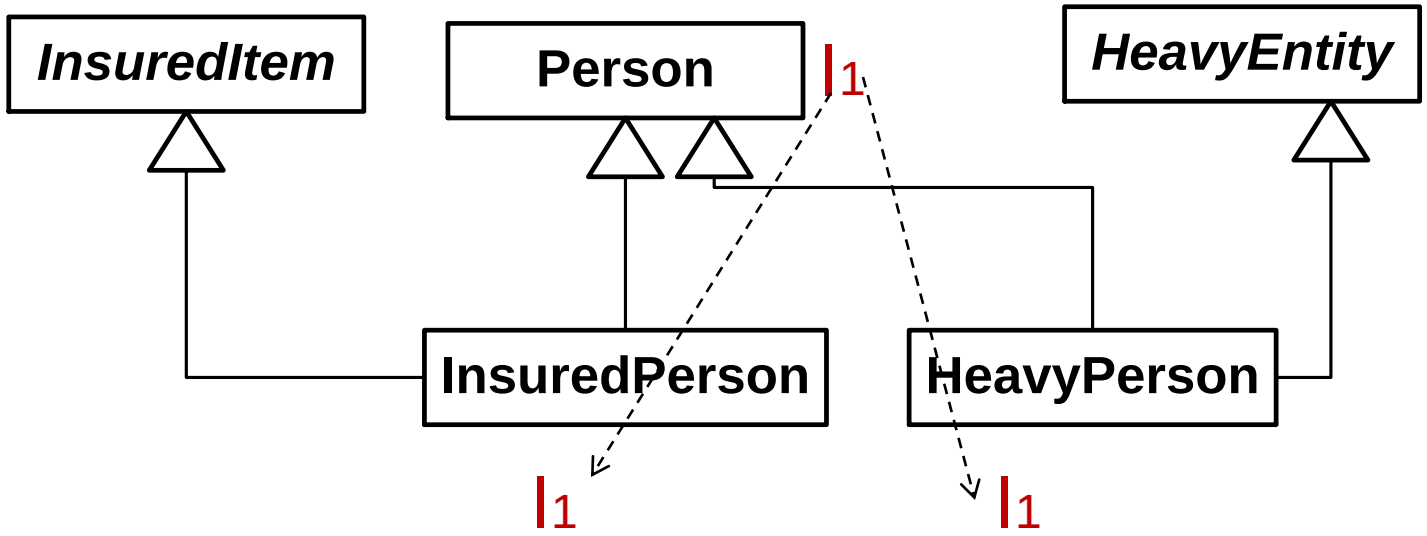
Foundations

- Since the unique principle of identity supplied by a Kind is inherited by its subclasses, we have that:
- A Non-sortal type cannot appear in a conceptual model as a subtype of a sortal



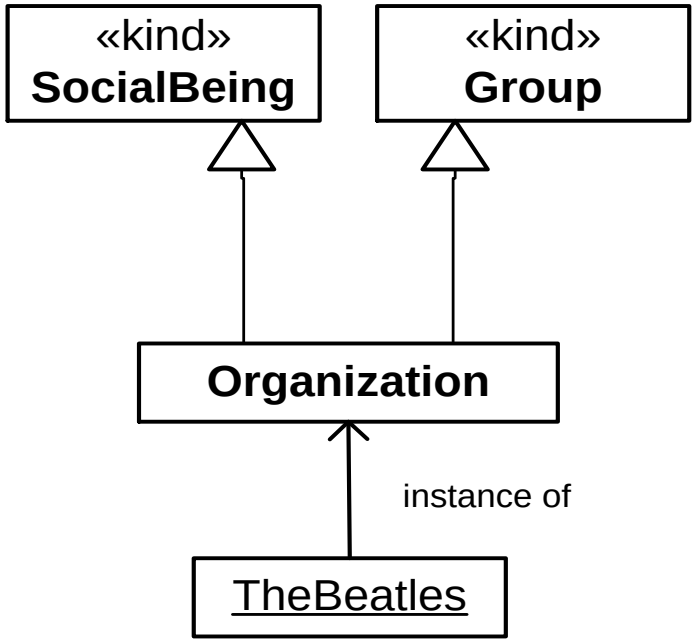
Foundations

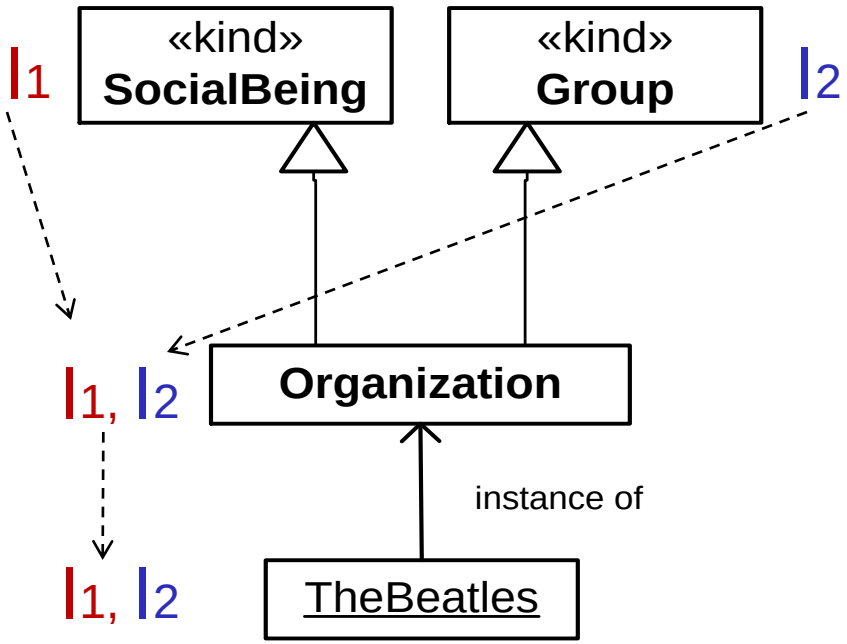
- Since the unique principle of identity supplied by a Kind is inherited by its subclasses, we have that:
- A Non-sortal type cannot appear in a conceptual model as a subtype of a sortal

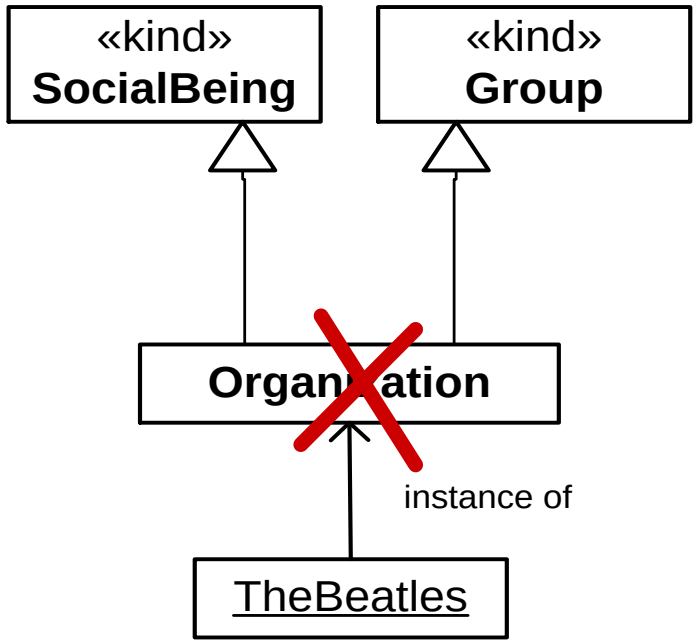


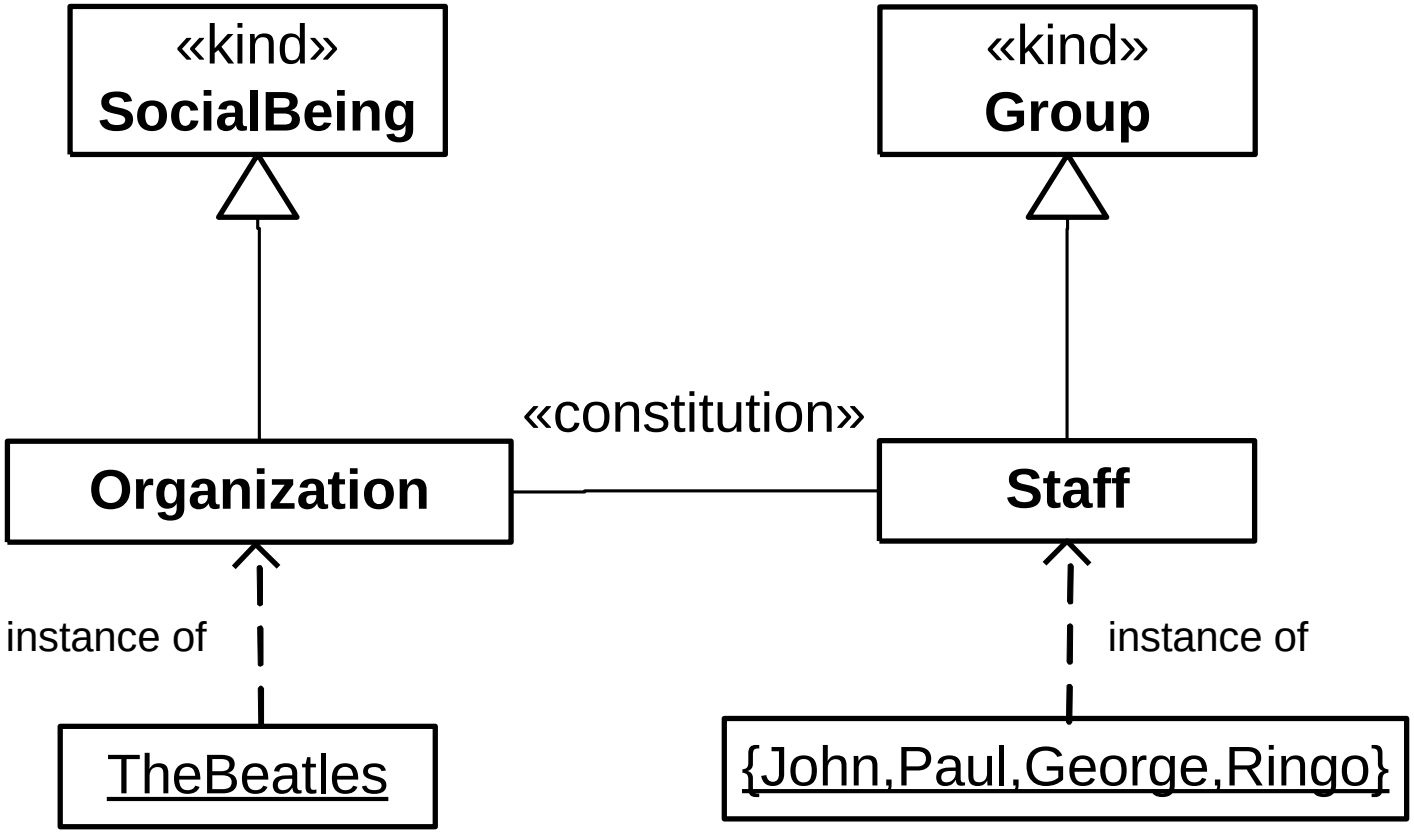
Foundations

- Since the unique principle of identity supplied by a Kind is inherited by its subclasses, we have that:
- An Object in a conceptual model of the domain cannot instantiate more than one ultimate Kind




«kind»
SocialBeing
«kind»
Group
I1
I2
I1, I2
Organization
instance of
I1, I2
TheBeatles


«kind»
SocialBeing
«kind»
Group
Organization
instance of
TheBeatles

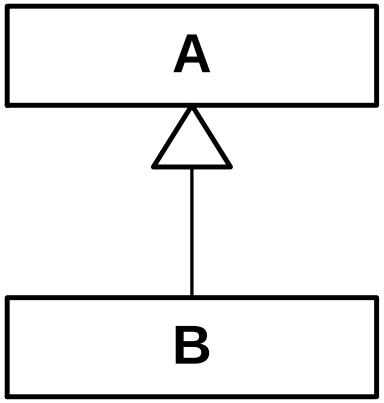

«kind»
SocialBeing
«kind»
Group
«constitution»
Organization
Staff
instance of
instance of
TheBeatles
{John,Paul,George,Ringo}

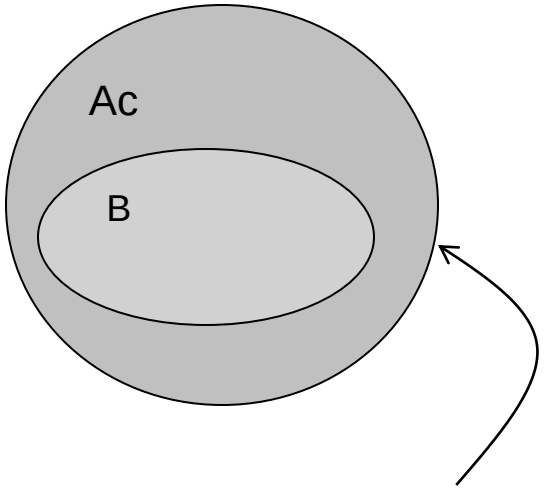
Foundations

- It is not the case that we cannot have multiple supertyping. Only that we a type cannot have multiple kinds as supertypes!

Foundations

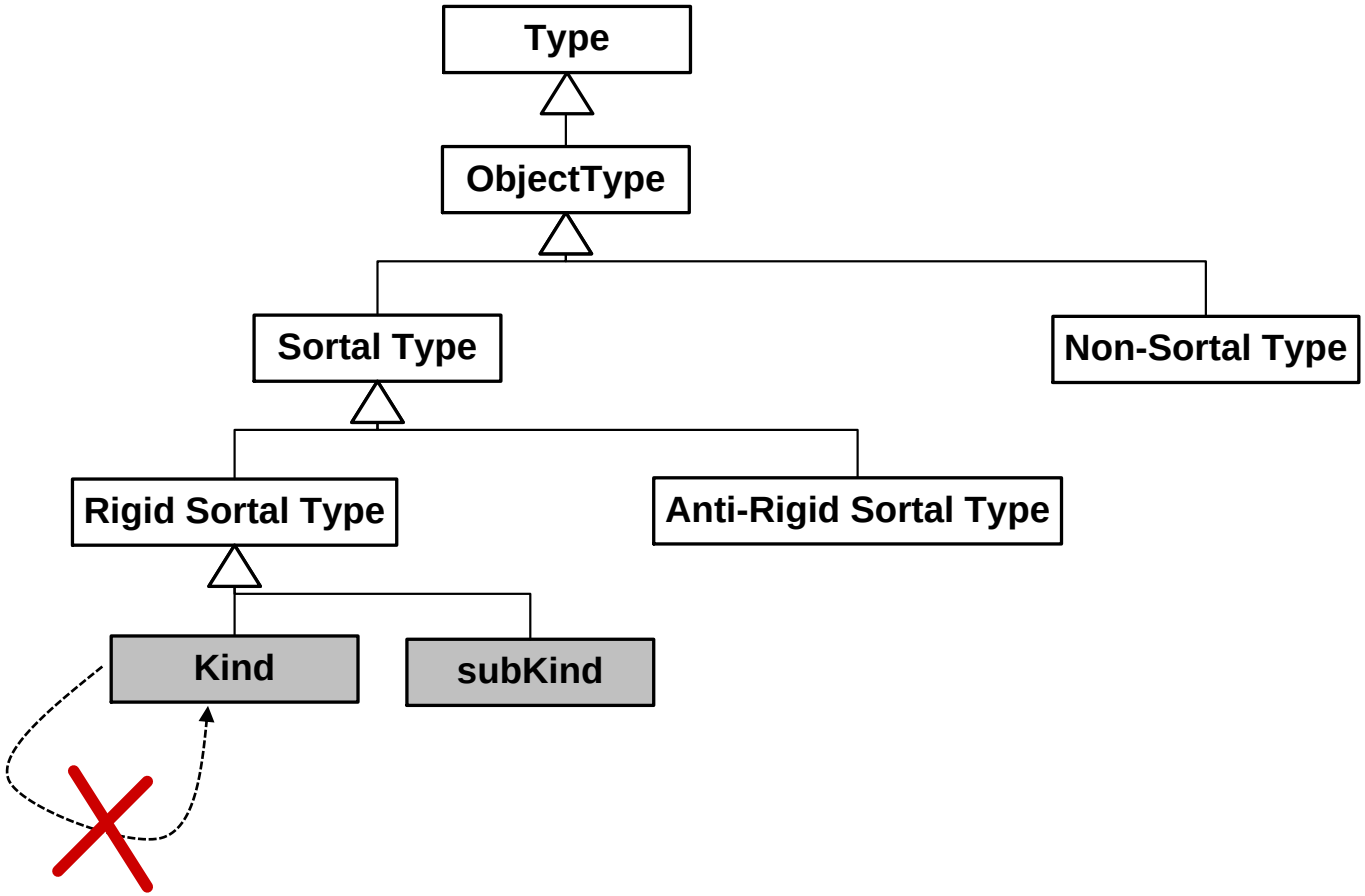
- It is important to emphasize the supertyping is a modal relation as well, i.e., if A is supertype of B then A is necessarily a supertype of B



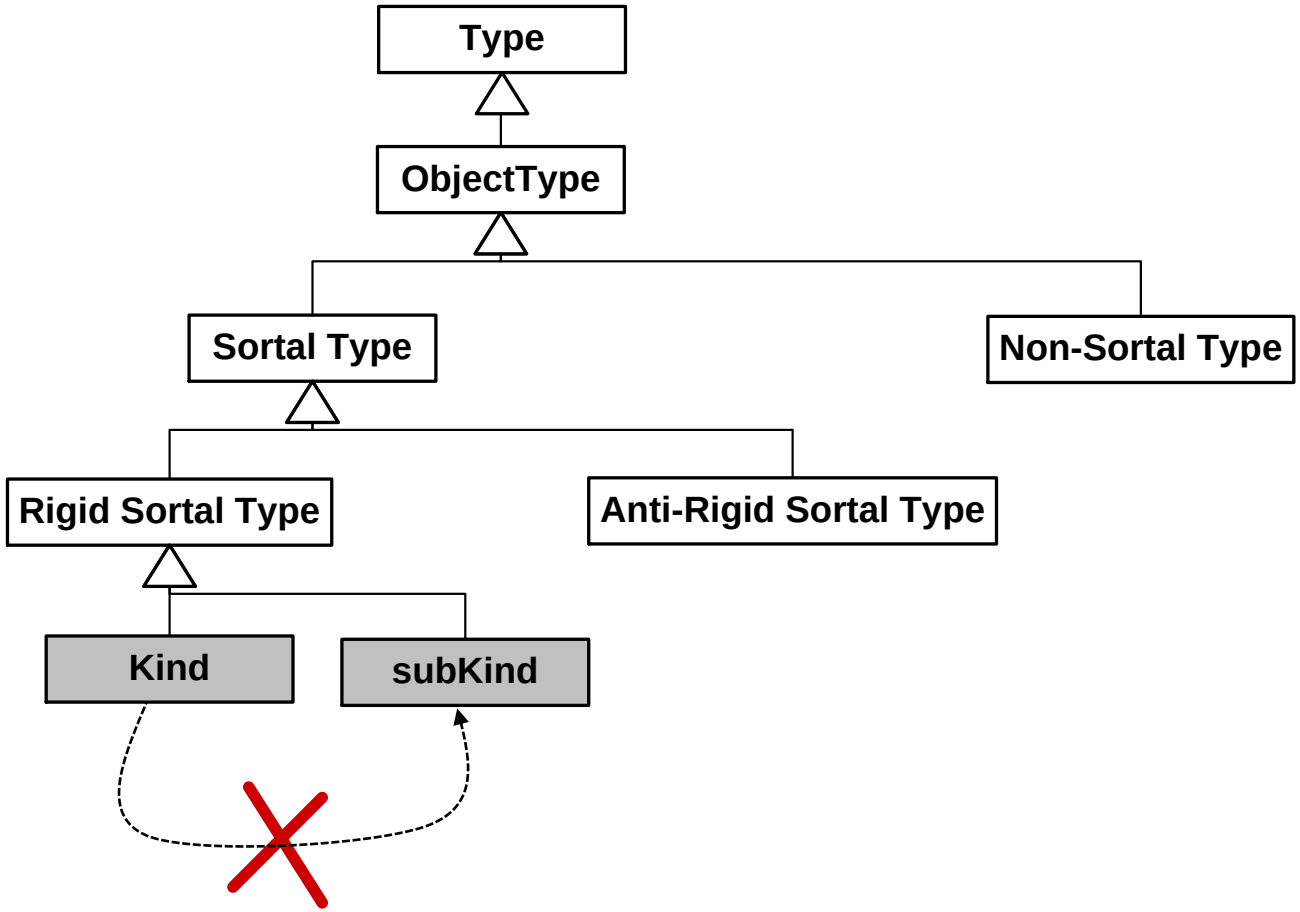


Supertype(A,B) =$_{def}$ $\Box(\forall x\ B(x) \rightarrow A(x))$

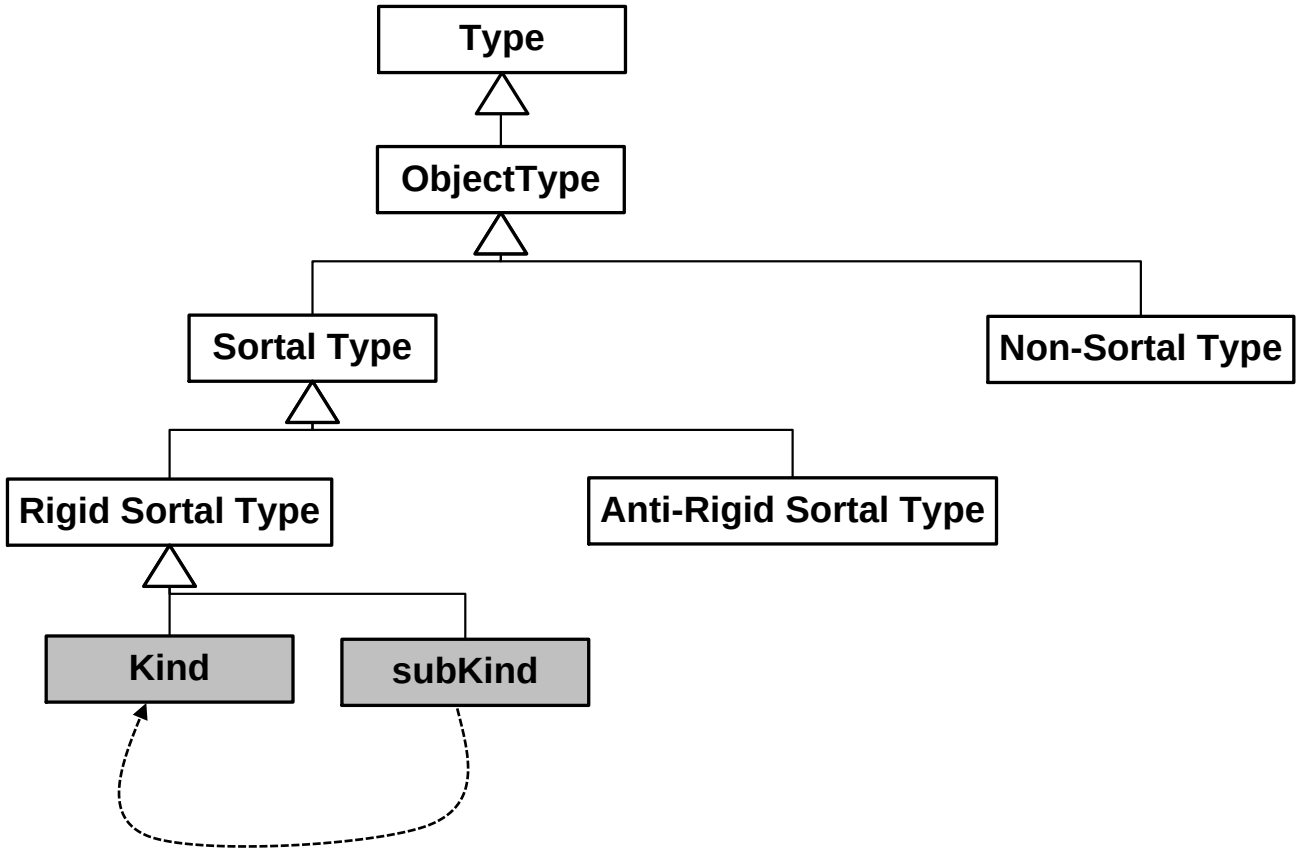
Fixed Configuration



A Kind cannot be a supertype of another Kind



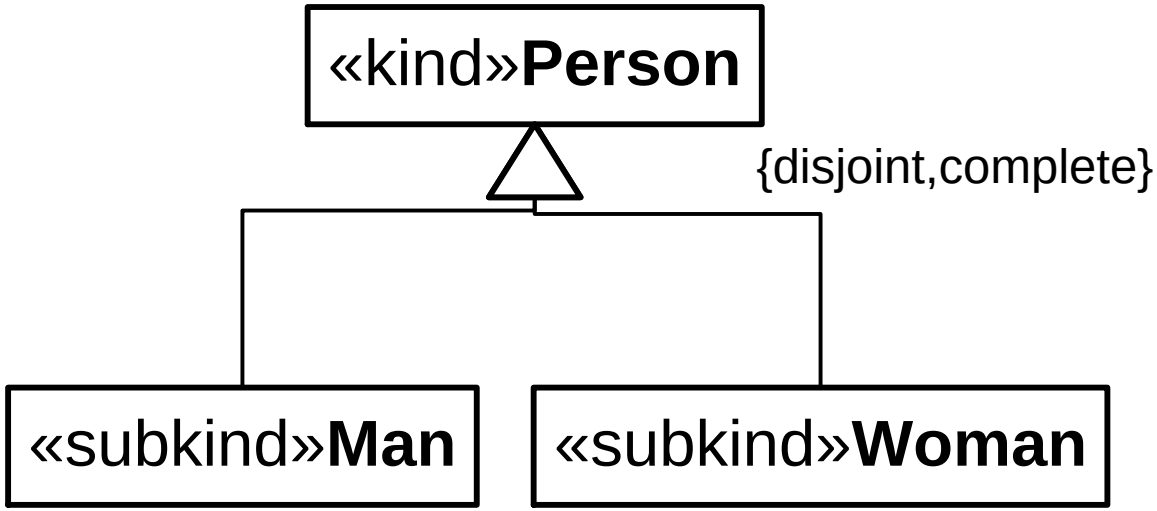
A subKind cannot be a supertype of a kind



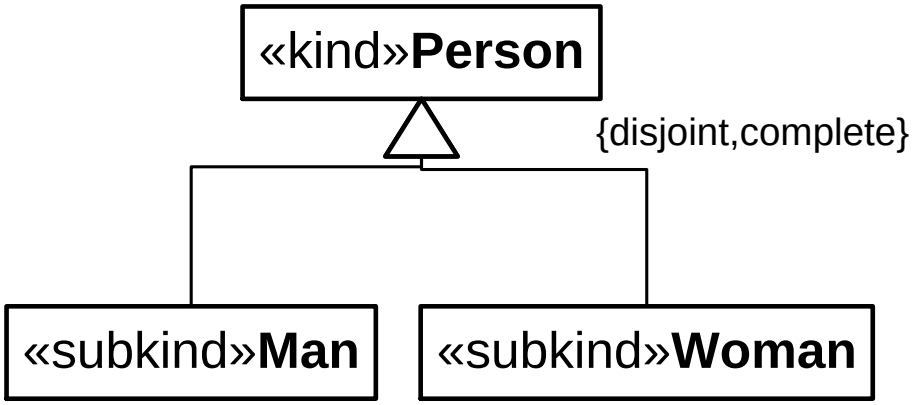
A subKind type MUST have as a supertype a (unique) Kind

Subkind Partitions

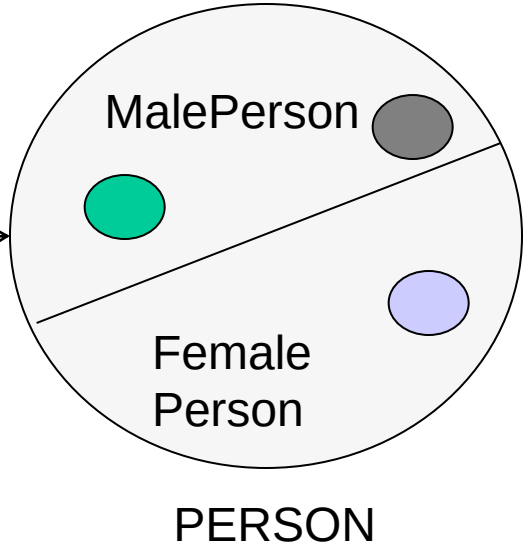
- It is typical that subkinds are defined in structures called Subkind Partitions
- These are not always partitions in the strong sense, i.e., they defined as disjoint but rarely complete generalization sets



Subkind Partitions

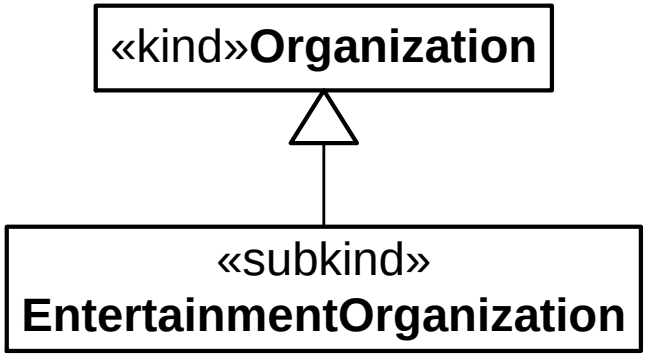


These partitions are the same
In every possible world!

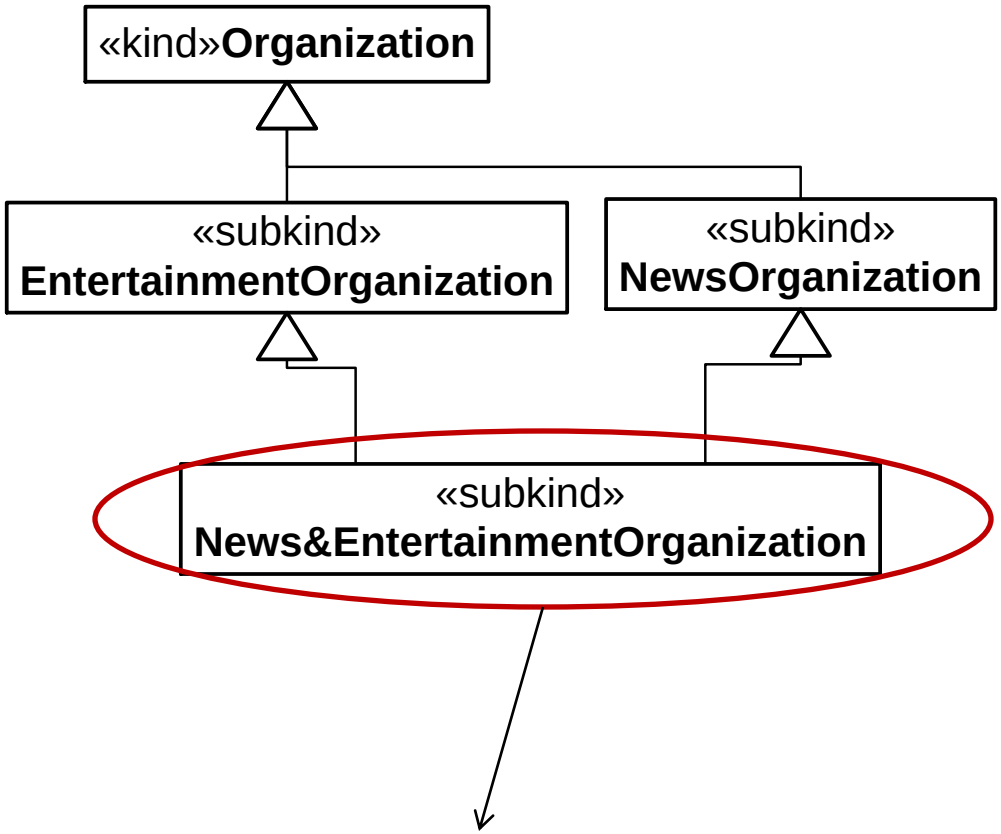


Subkinds

- However, subkinds also appear outside generalization sets as specializations of kinds



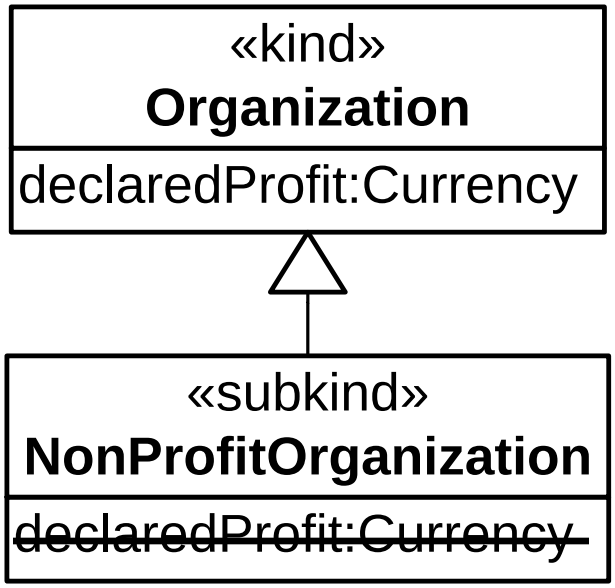
Subkinds



This only makes sense if there are genuine (intrinsic or relational) properties to be defined for the intersection type

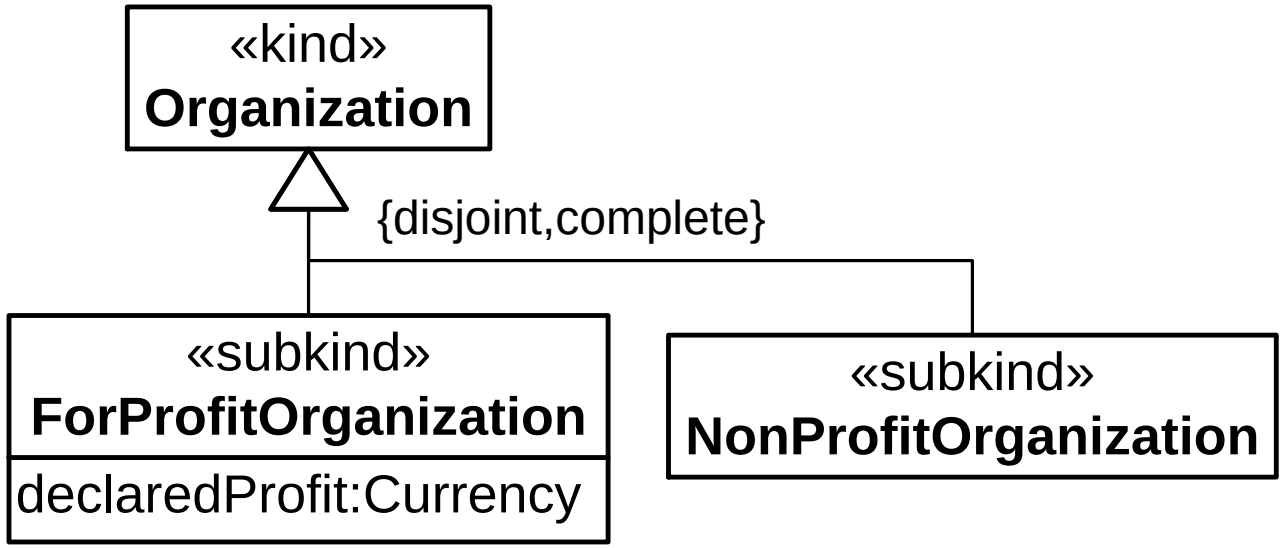
Subkind

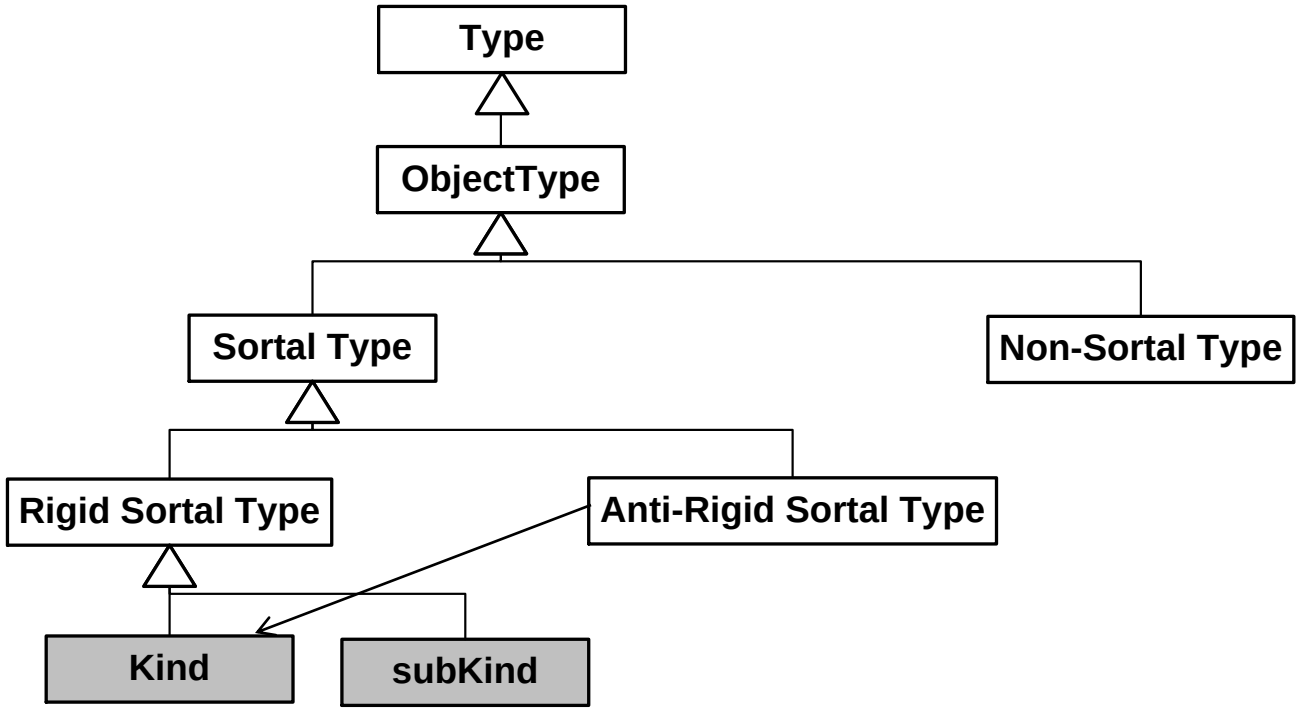
- Remember that property overriding is always a bad idea and it is always caused by a conceptual modeling mistake



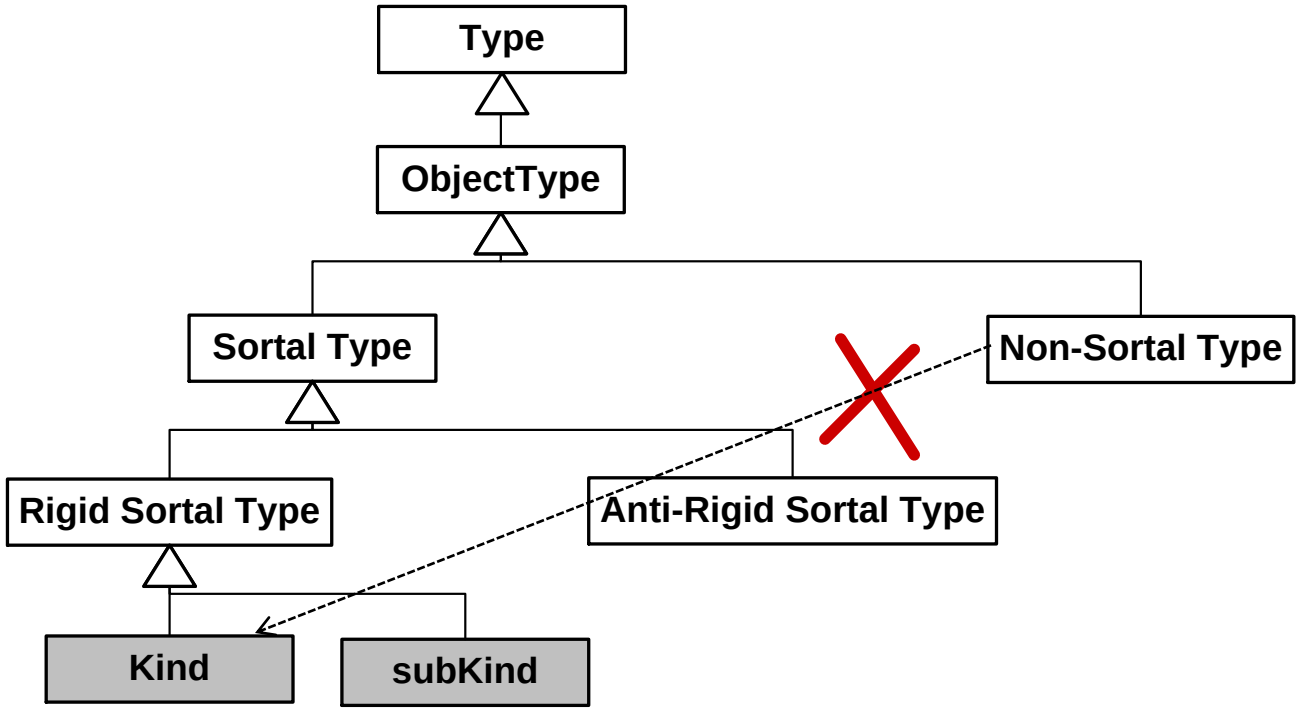
Subkind

- Remember that property overriding is always a bad idea and it is always caused by a conceptual modeling mistake

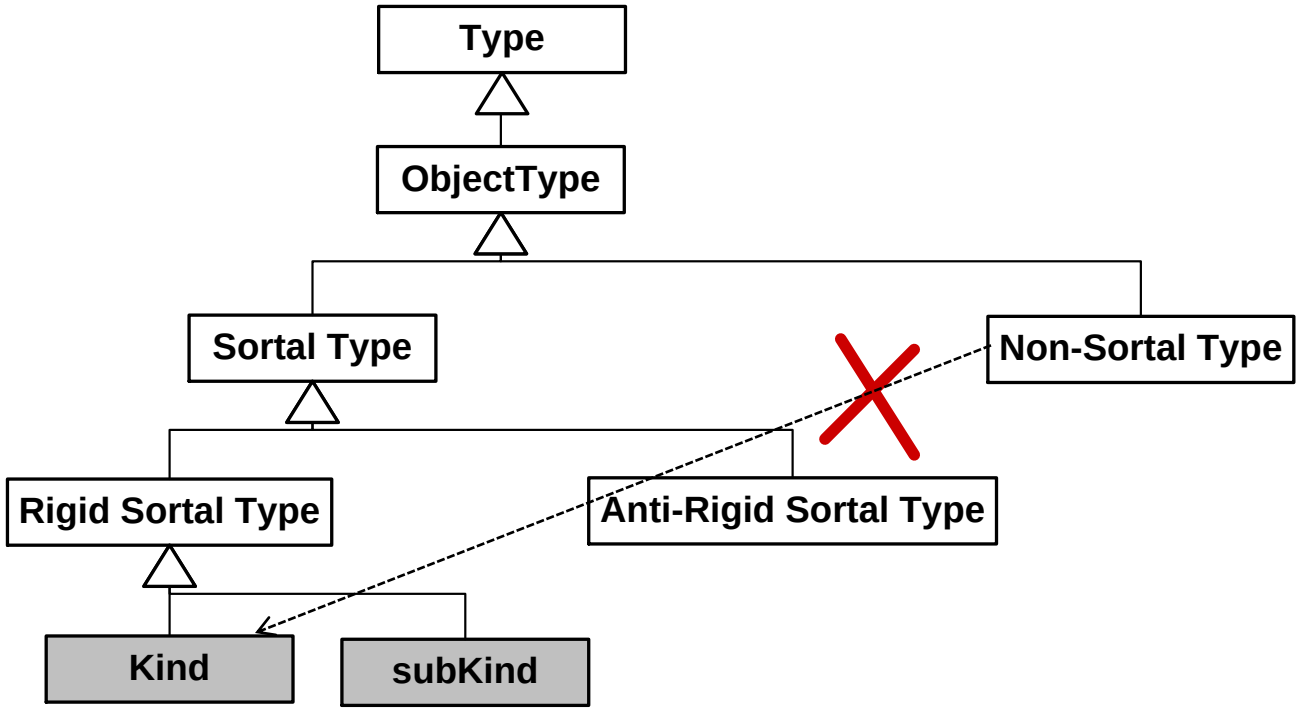




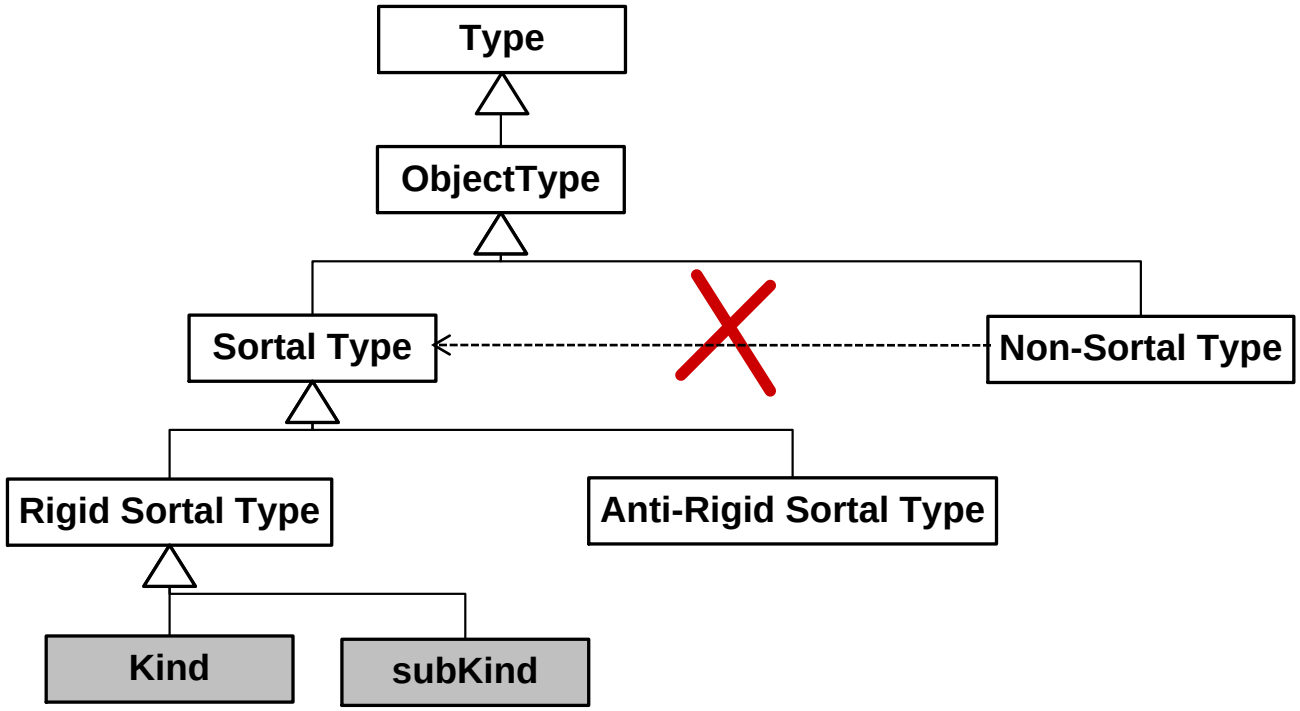
An Anti-Rigid type MUST have as a supertype a (unique) Kind



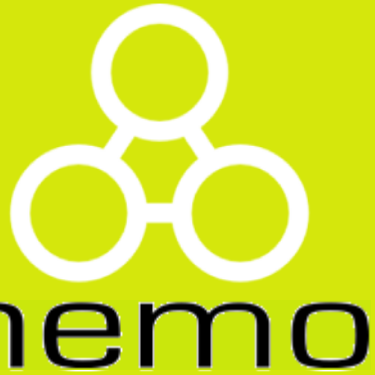
A Kind cannot be a supertype of a Non-Sortal Type



A Kind cannot be a supertype of a Non-Sortal Type in fact, since all sortals will inherit a principle of Identity from a Kind



A Sortal Type cannot be a supertype of a Non-Sortal Type



gguizzardi@inf.ufes.br