OntoUML specification Documentation

Marek Suchánek

Oct 09, 2018

Contents

Welcome to the documentation of *OntoUML* ontology-driven conceptual modelling language based on upper ontology *UFO*. We welcome any form of *contribution* and questions that will make this documentation better as it is community-driven hosted on github.com.

CHAPTER 1

Introduction

1.1 OntoUML

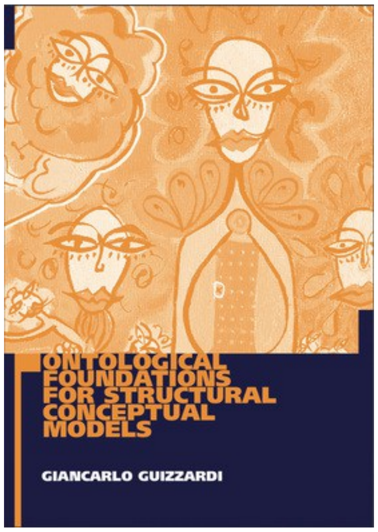
OntoUML is an ontologically well-founded language for Ontology-driven Conceptual Modeling. OntoUML is built as a UML extension based on the *Unified Foundational Ontology (UFO)*. The foundations of UFO and OntoUML can be traced back to Giancarlo Guizzardi's Ph.D. thesis "Ontological Foundations for Structural Conceptual Models". In his work, he proposed a novel foundational ontology for conceptual modeling (UFO) and employed it to evaluate and re-design a fragment of the UML 2.0 metamodel for the purposes of conceptual modeling and domain ontology engineering. OntoUML has been adopted by many academic, corporate and governmental institutions worldwide for the development of conceptual models in a variety of domains. It has also been considered as a candidate for addressing the OMG SIMF (Semantic Information Model Federation) Request for Proposal, as is explicitly recognized as the foundations for the "Data Modeling Guide (DMG) For An Enterprise Logical Data Model (ELDM)" initiative. Finally, some of the foundational theories underlying OntoUML have also influenced other popular conceptual modeling languages such as ORM 2.0.

Source: wikipedia.org

1.2 UFO

The **Unified Foundational Ontology** (UFO), developed by Giancarlo Guizzardi and associates, incorporating developments from GFO, DOLCE and the Ontology of Universals underlying OntoClean in a single coherent foundational ontology. The core categories of UFO (UFO-A) have been completely formally characterized in Giancarlo Guizzardi's Ph.D. thesis and further extended at the Ontology and Conceptual Modelling Research Group (NEMO) in Brazil with cooperators from Brandenburg University of Technology (Gerd Wagner) and Laboratory for Applied Ontology (LOA). UFO-A has been employed to analyze structural conceptual modeling constructs such as object types and taxonomic relations, associations and relations between associations, roles, properties, datatypes and weak entities, and parthood relations among objects. More recent developments incorporate an ontology of events in UFO (UFO-B), as well as an ontology of social and intentional aspects (UFO-C). The combination of UFO-A, B and C has been used to analyze, redesign and integrate reference conceptual models in a number of complex domains such as, for instance, Enterprise Modeling, Software Engineering, Service Science, Petroleum and Gas, Telecommunications, and Bioinformatics. Another recent development aimed towards a clear account of services and service-related concepts, and provided for a commitment-based account of the notion of service (UFO-S), UFO is the foundational ontology for *OntoUML*, an ontology modeling language.

Source: wikipedia.org



CHAPTER 2

Theory

2.1 Types and Individuals

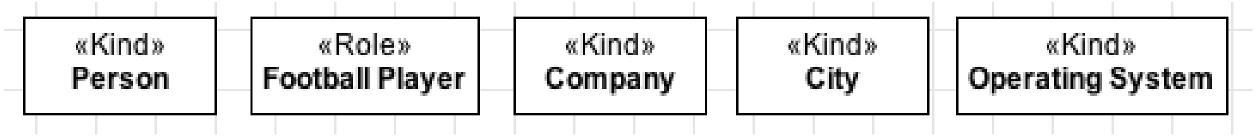
OntoUML is built upon the fundamental distinction between **Types** and **Individuals**. And that is because we like classify things.

Types are abstract *things* we create to help us perceive and classify the world around us. These *things* work as bundles of characteristics we can expect to encounter in other particular *things* - the **individuals**.

Let's consider the type Person. Which characteristics does every Person have? We could say a head, a heart, arms, hands, legs, feet, eyes… Every person also has a weight, a height, an age. Maybe a name, place of birth, birthdate.

Now let's consider you and me. I am individual. And so are you. If you are reading this, I am confident to say that we are both people. We both have a heart, we both have a particular height and weight. We *exemplify* what it is to be a Person. The relation that holds between us and the type Person is called **instantiation**.

In OntoUML, we represent classes as boxes, just like in UML. Every class must have a name and a stereotype, as depicted in the figure below:



Now, let's see some other examples of types and individuals them:

- **Person**: Bill Gates, Linus Torvalds, Barack Obama, Steve Jobs, Alan Turing, Messi
- **Football Player**: Neymar, Messi, Cristiano Ronaldo, Pelé, Maradona
- **City**: Rio de Janeiro, Milano, Barcelona, New York City, London, Lisbon
- **Operating System**: Windows, OS X, Ubuntu
- **Company**: Apple, Samsung, Microsoft, Facebook, Nokia

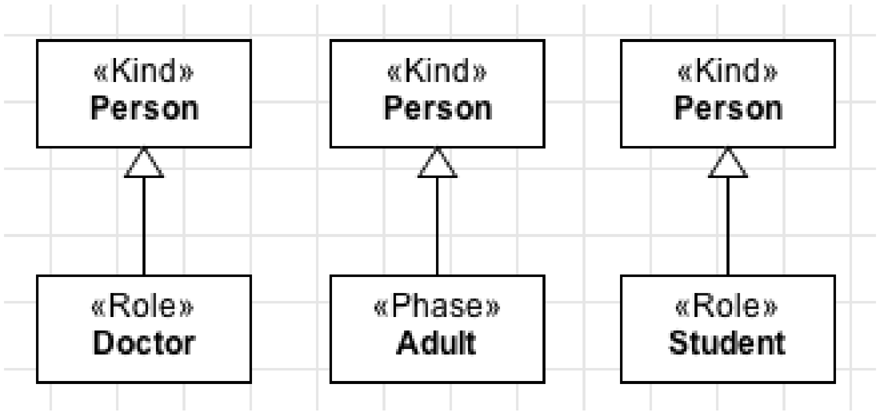
If you pay close attention on the list, you will see that we included Messi's name as an instance of Person and Football Player. And that is fine! In fact, it very common that an individual simultaneously instantiates many types. Me, for example, besides being a Person, I'm a Software Developer, a Brazilian, an Adult and a Man.

Whenever we refer to the term **extension of a type**, we mean every individual that instantiates that type in a particular instant of time. As an example, let's assume that the type Web Browser. Last year, we could say that its extension contained 5 individuals: Chrome, Internet Explorer, Safari, Firefox, Opera. This year, however, after Microsoft Edge's release, the extension of Browser grew by 1.

Whenever the extension of a type is always included in the extension of another type, we say that the former is a subtype of the latter. To represent this constraint in OntoUML models, we use the **generalization** (some people call it **specialization** instead) relation. We find countless examples of type specializations:

- Doctor, Student and Child are subtypes of Person
- Table, Mouse and Ball are subtypes of Object
- Fridge, Stove and Microwave are subtypes of Appliance

We represent generalizations are lines with arrow heads on the end connected to the super-type, as shown in the figure below:

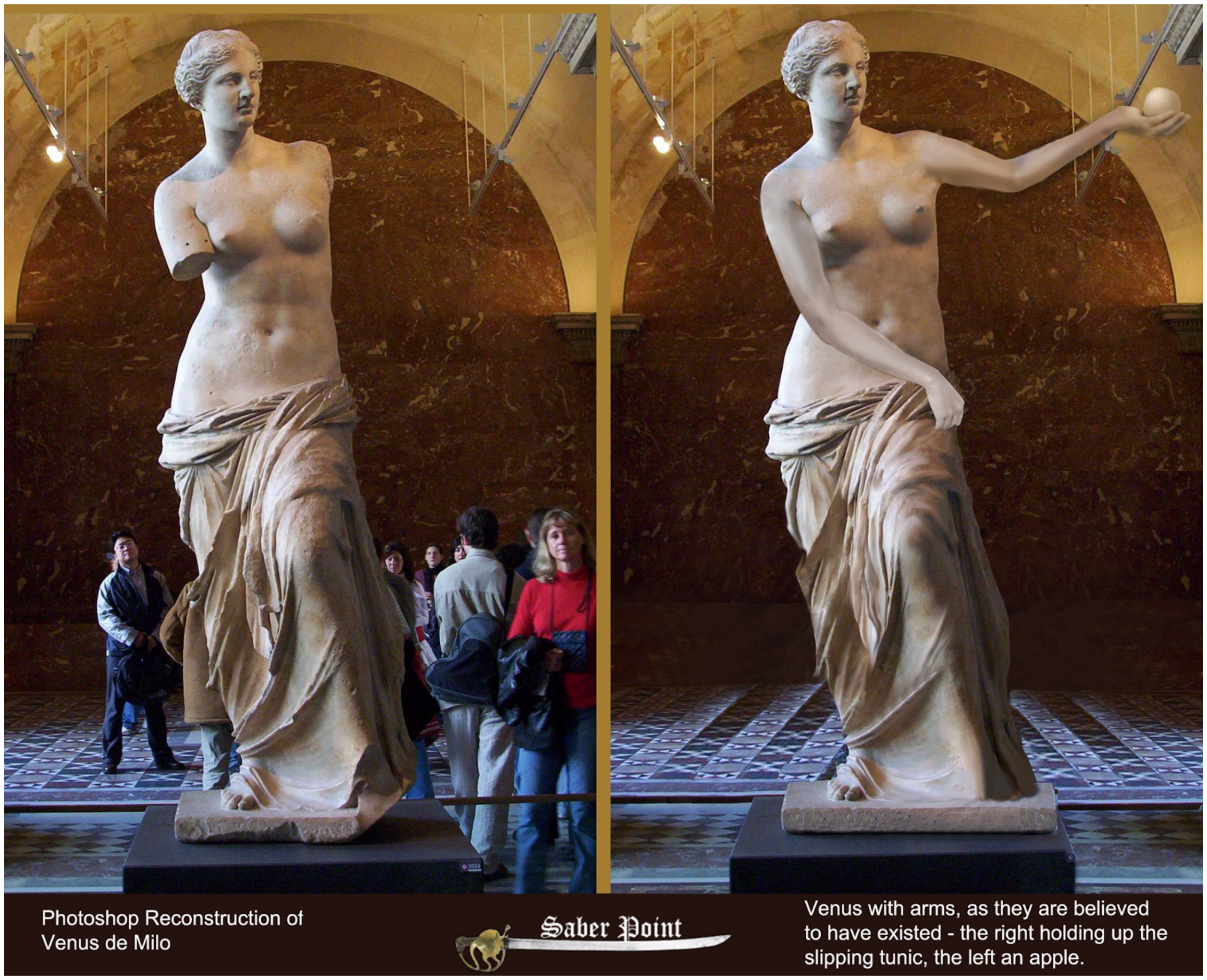


When we build a model in OntoUML we are formally defining types by specifying the characteristics they impose on their instances.

Warning: OntoUML *ONLY* supports the specification of *TYPES*. Therefore, you *CANNOT* specify an *INDIVIDUAL* in an OntoUML model. Making an analogy to regular UML, you can create Class Diagrams, but there is no Object diagram.

2.2 Identity

Another fundamental ontological notion you need to grasp before you start modelling is the ontological notion of identity. To start the discussion, let's take a look at the picture below:

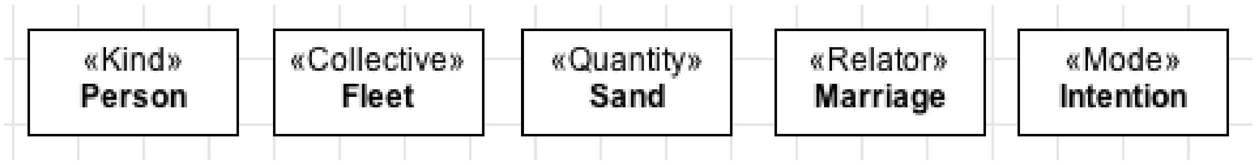

Photoshop Reconstruction of Venus de Milo
Saber Point
Venus with arms, as they are believed to have existed - the right holding up the slipping tunic, the left an apple.

As you might know, that is *Aphrodite of Milos*, better known as the *Venus de Milo*, an ancient Greek statue and one of the most famous works of ancient Greek sculpture (Wikipedia). On the left side, it's the statue's current state, and on the right, it's how it was supposably built. My question for you is: *Do these pictures portrait the same individuals or different ones?*. Is it the same statue that went through some changes or these changes destroyed the first individual (the statue with arms) and created a new one (the statue without arms)? If you think like most people, your answer would be: "Yes, they are the same individual.". Now, what if the statue was broken into very little pieces, like in the picture below:

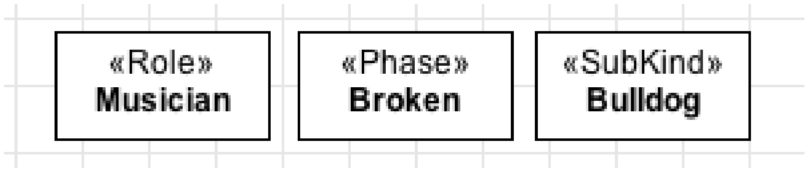
Would you say that these marble debris are still the statue? Somehow our intuition says no, right? These debris cannot be Venus anymore. But why do we say "Yes" to the first question and "No" to the second one? Because of our common sense identity principle for statue. An **identity principle** is a sort of function we use to distinguish two individuals. Let's use the simplest example of all: the identity principle of sets. Two sets, A and B, are the same if, and only if, they have the same elements. Therefore, if A = {1,2} and B = {2,3} then A != B. So the identity of a set is defined by its members. Changing a member of a set changes the identity of the set. Now, let's think about a morencomplicated example. Let's say, the identity principle we adopt for people. Could we say that someone's identity depends on their name? Or some sort of identification code, like the American 'social security', the Brazilian 'CPF' or the Italian 'codice fiscale'? The answer is *NO*! These can't be used as our identification function. And I'll tell you why…

Let's start with a Person's name. Did you ever meet two folks with the very same name? I have. If you don't believe, just go on Facebook and experiment search for common names of your country. I just searched for "João Carlos da Silva", a fairly common Brazilian name, and I found at least 5 guys with that exact name. If name was our identity function, we would not be able to distinguish between them. Another problem with using name as identity is that often, people change their names. Our function needs to be not only able to distinguish two individuals in the same moment in time, but also through time. How else would we be able to meet someone today and recognize that same person tomorrow? So, our function needs to always return the same individual for a given input. Now, let's analyze the reason why the social security number (SSN), the codice fiscale and the CPF are not very good identity principles for people. The answer is quite simple, our function needs to apply to everybody. If you are not American or never worked in the USA, you probably don't have a SSN, right? Even young children born in the USA might not have. The last important fact about identity principle is that every individual must have *exactly one*. So, what is the identity principle for a person? One's fingerprint, iris pattern, DNA? Well, it is really hard to define it, even though we know it is there.

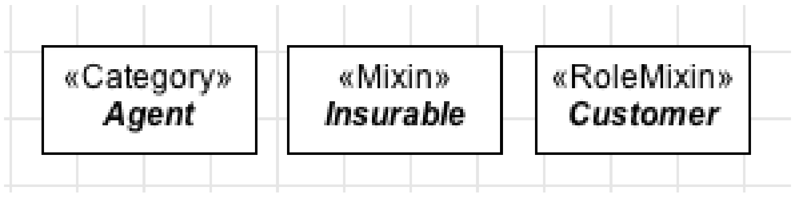
What we can "touch" are what's called the **identity conditions**. These are "parts" of the identity function, necessary conditions for identity but not sufficient by themselves. In order for me to consider A and B as the same Person they need to have the same birth date. And the statue need to be made of the same material. Why identity principles and conditions are important for us? Because by thinking about them we are guided in the construction of our types hierarchy. They impose constraints on how we can combine the different OntoUML constructs to design our conceptual models. Will talk about these constraints when we present the stereotypes usage. For now, just keep in mind that: Some types have the characteristic of providing identity principles for their instances. They are stereotype as: «*Kind*», «*Collective*», «*Quantity*», «*Relator*», «*Mode*» and «*Quantity*». Here are some examples:



Some other types don't provide identity principle for their instances, but they all share a common one. They are stereotyped as: «*Subkind*», «*Role*» and «*Phase*». Here are some examples:



Some other types don't provide identity and their instances follow different identity principles. They are stereotyped as: «*RoleMixin*», «*Mixin*» and «*Category*». Here are some examples:

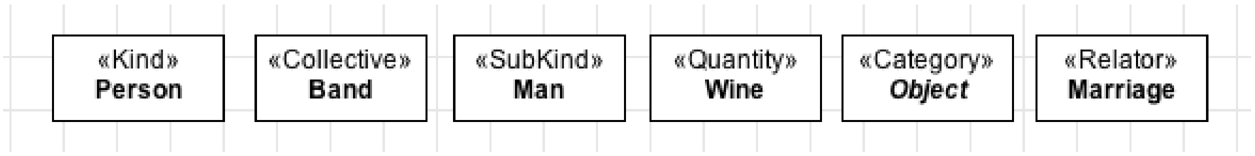


2.3 Rigidity

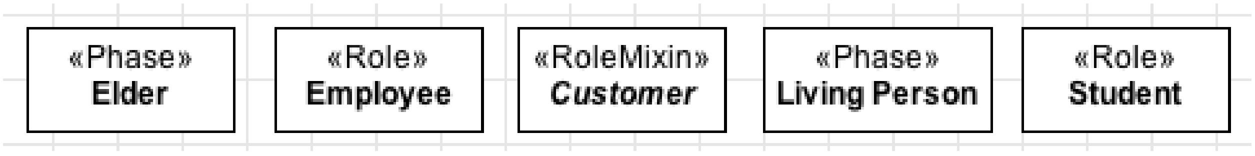
Now that you are already familiar with the notion of type, individual and instantiation, let's go through a fundamental ontological meta-property of types - rigidity. To start, let's take a look at the following pictures:

They show a dog's development through the years (let's call him Rex for now). In the first frame (and maybe also in the second) Rex is a Puppy. In the third one he is not a Puppy anymore, but an Adult. However, in all three frames Rex is a Dog and a French Bulldog. Let's focus on the types Dog and French Bulldog. Can you imagine any other point in time, besides the three shown in the pictures, in which Rex ceased to be either a Dog or a Bulldog? I guess not. Let's expand our imagination a little. Can you imagine any individual that used to be a Dog but is not anymore? I bet the answer is also no.

If an individual must instantiate a given type in all possible scenarios in which the individual exists, we call that type **RIGID**. In other words, rigid types are the ones who define essential characteristics to their instances. Other examples of rigid types are: Person, Car, Band, Apple, Country and Company. List of rigid stereotypes: «*Category*», «*Collective*», «*Kind*», «*Mode*», «*Quality*», «*Quantity*», «*Relator*», and «*Subkind*».



Now, let's focus solely on the type Puppy. By looking at the pictures, we can see that Rex used to be a puppy, but stopped being one after he grew older. Just like Rex, every other dog was once a puppy or will cease to be one someday. If every individual that instantiate a given type in a particular time can cease to do so and still exists, then we call that type **ANTI-RIGID**. Examples of anti-rigid types are: Student, Employee, Spouse, Elder, Living Person and Healthy Person. List of anti-rigid stereotypes: «*Role*», «*Phase*» and «*RoleMixin*»



2.4 Dependency

Todo: This topic will be covered soon…

2.5 Objects & Events

Todo: This topic will be covered soon…

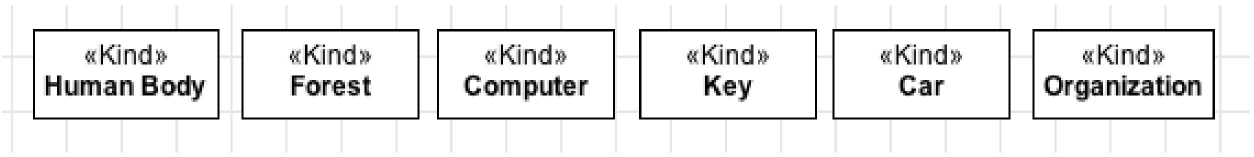
CHAPTER 3

Class stereotypes

3.1 Kind

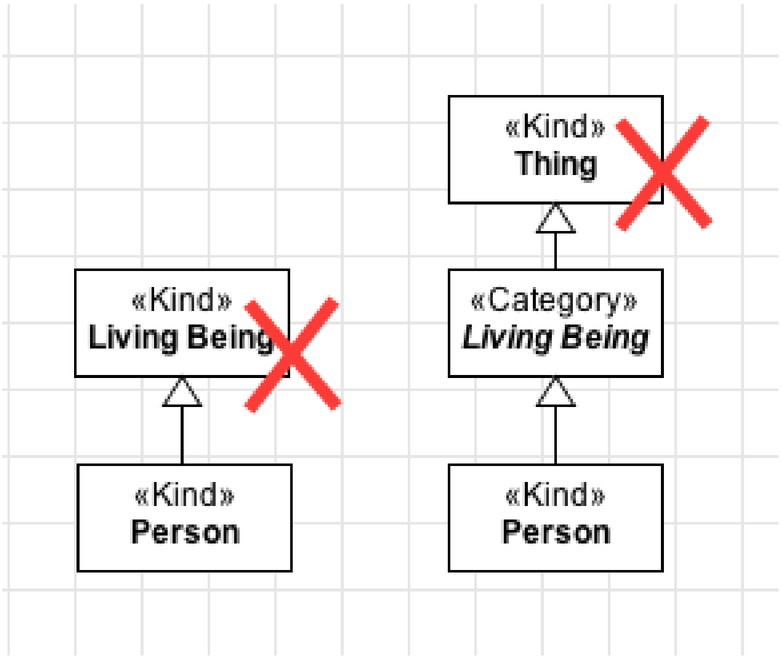
3.1.1 Definition

A «*Kind*» is construct you are going to use in most of your models. It is used to represent *rigid* concepts that provide an *identity principle* for their instances and do not require a *relational dependency*. A «*Kind*» represent a **Functional Complex**, i.e., a whole that has parts contributing in different ways for its functionality (see the ComponentOf relation for more details about functional parts). Let's see some examples:

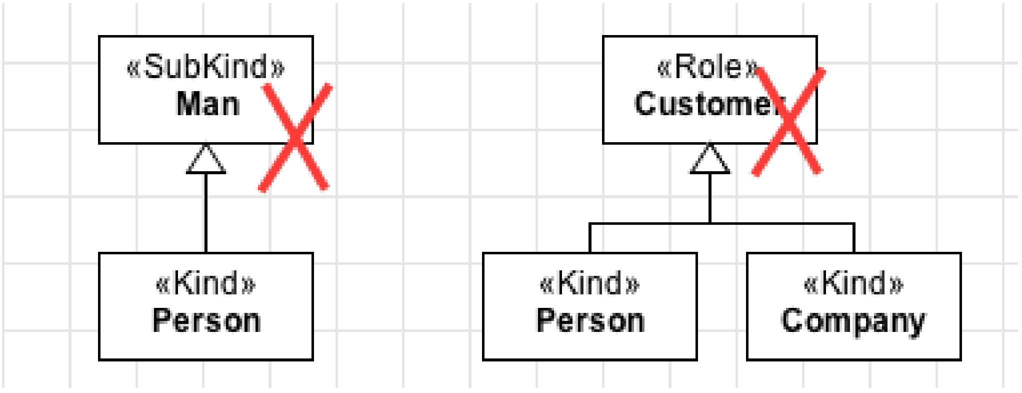


3.1.2 Constraints

C1: A «*Kind*» cannot have an *identity* provider («*Kind*», «*Collective*», «*Quantity*», «*Relator*», «*Mode*» and «*Quantity*») as its direct or indirect super-type.



C2: A «*Kind*» cannot have types that inherit *identity* («*Subkind*», «*Role*» and «*Phase*») as its direct or indirect super-type.



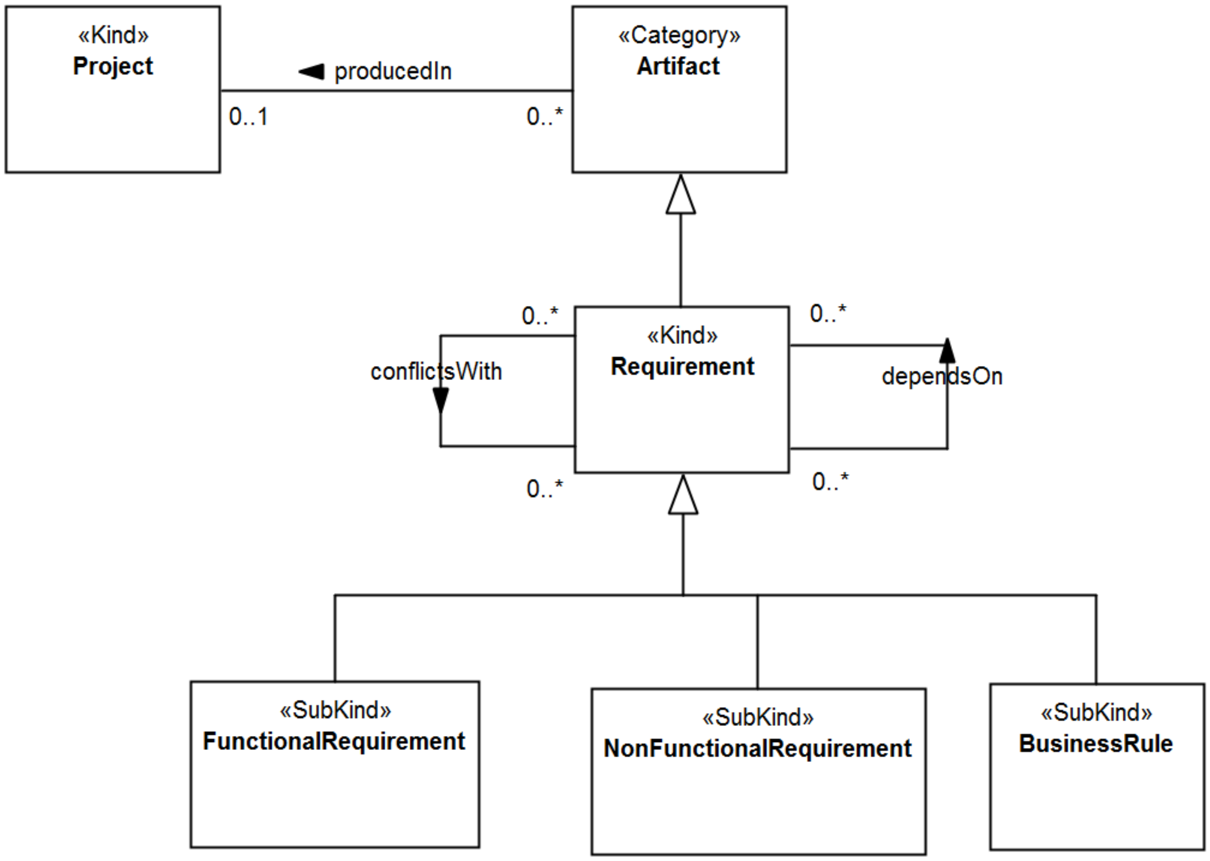
C3: A «*Kind*» cannot have types that aggregate individuals with *different identity principles* («*Category*», «*RoleMixin*» and «*Mixin*») as its direct or indirect subtypes.

C4: As a *rigid* type, a «*Kind*» cannot have any *anti-rigid* type («*Role*», «*RoleMixin*» and «*Phase*») as its direct or indirect super-type.

3.1.3 Common questions

Q1: If a «*Kind*» is relationally independent, does that mean we cannot define relations for theses types?

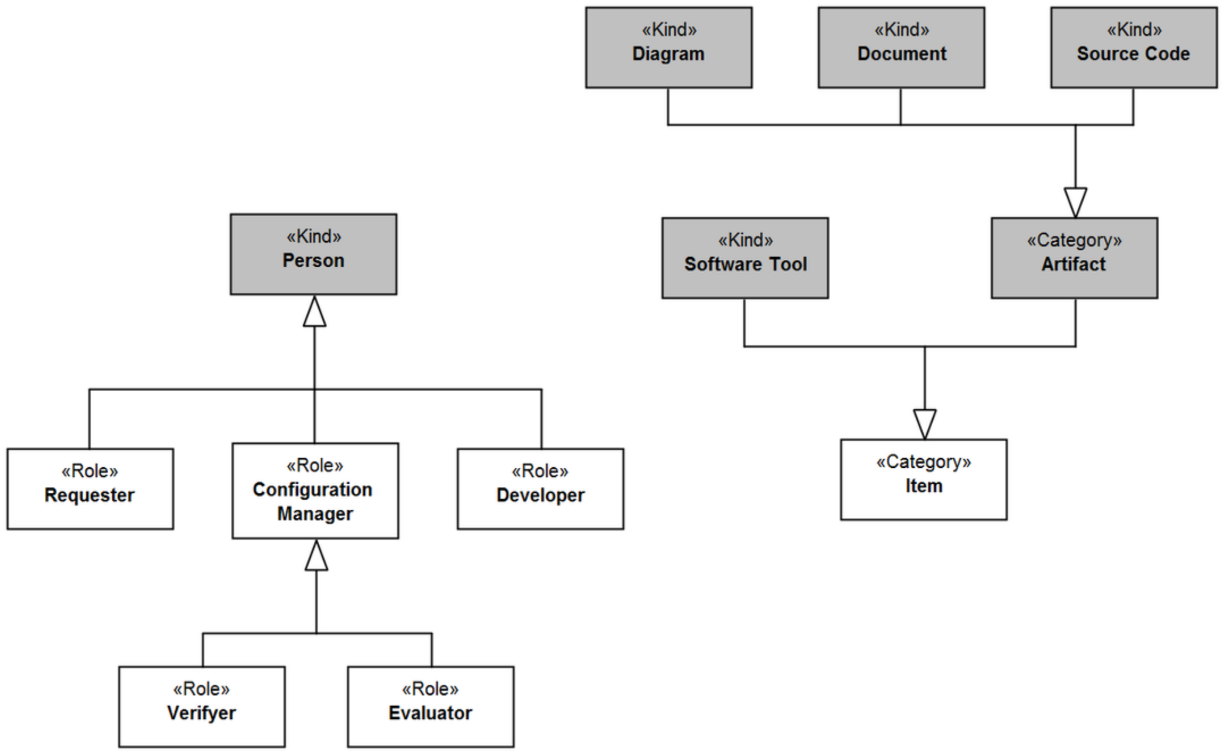
A1: No! When we say that a «*Kind*» is relationally independent, we mean that it does not necessarily require a relation to be defined, like a «*Role*» does. Here is an example in which a «*Kind*» has a *dependency*.



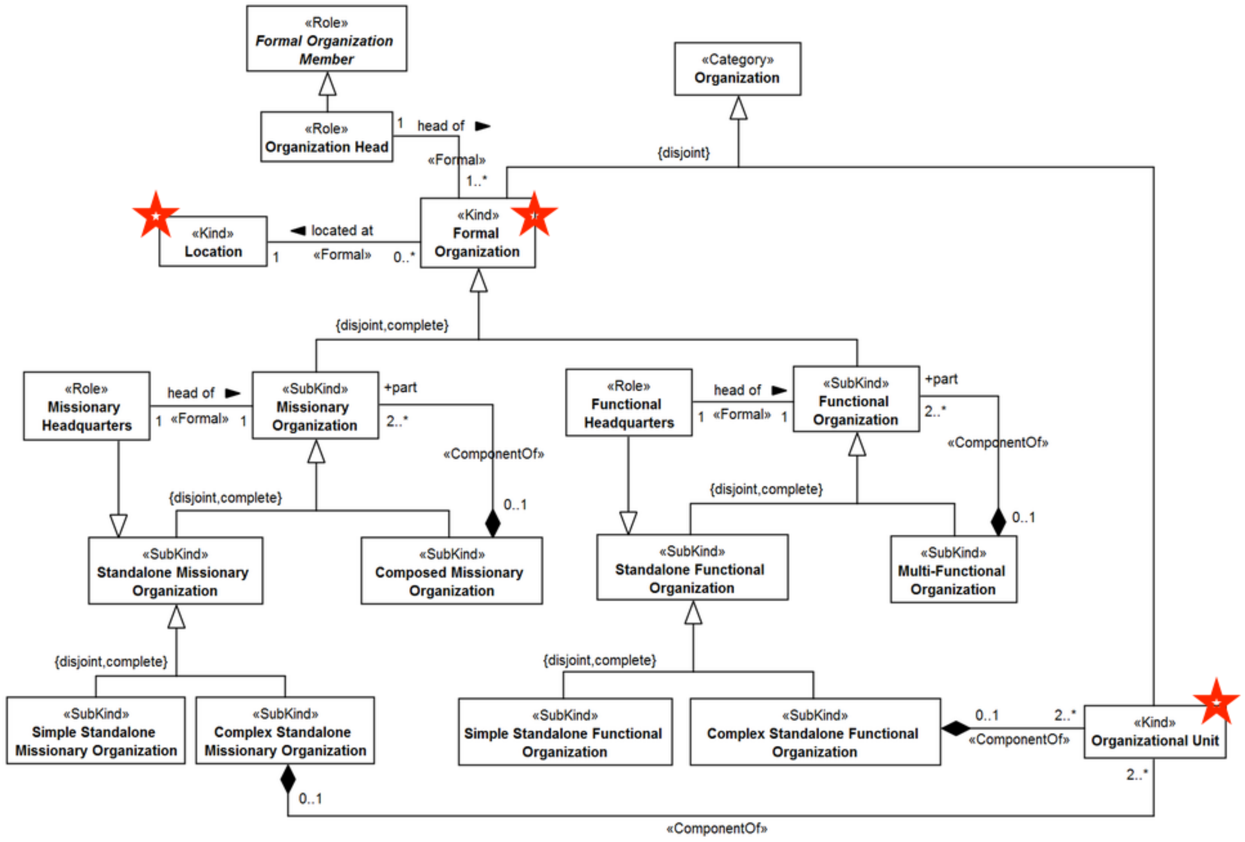
This example was extracted from the Software Requirements Reference Ontology (SRRO). Click here to take a look at it.

3.1.4 Examples

EX1: Fragment from the Configuration Management Task Ontology (see more):



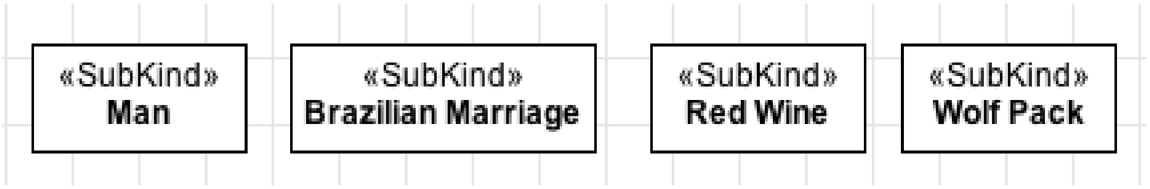
EX2: Fragment from the OntoUML Org Ontology (O3) (see more):


«Role» Formal Organization Member
«Category» Organization
«Role» Organization Head
1 head of
«Formal»
1..*
{disjoint}
«Kind» Location
located at
1 «Formal» 0..*
«Kind» Formal Organization
{disjoint,complete}
«Role» Missionary Headquarters
head of
1 «Formal» 1
«SubKind» Missionary Organization
+part
2..*
«ComponentOf»
0..1
{disjoint,complete}
«SubKind» Standalone Missionary Organization
«SubKind» Composed Missionary Organization
«Role» Functional Headquarters
head of
1 «Formal» 1
«SubKind» Functional Organization
+part
2..*
«ComponentOf»
0..1
{disjoint,complete}
«SubKind» Standalone Functional Organization
«SubKind» Multi-Functional Organization
{disjoint,complete}
«SubKind» Simple Standalone Missionary Organization
«SubKind» Complex Standalone Missionary Organization
{disjoint,complete}
«SubKind» Simple Standalone Functional Organization
«SubKind» Complex Standalone Functional Organization
0..1
2..*
«ComponentOf»
«Kind» Organizational Unit
2..*
0..1
«ComponentOf»

3.2 Subkind

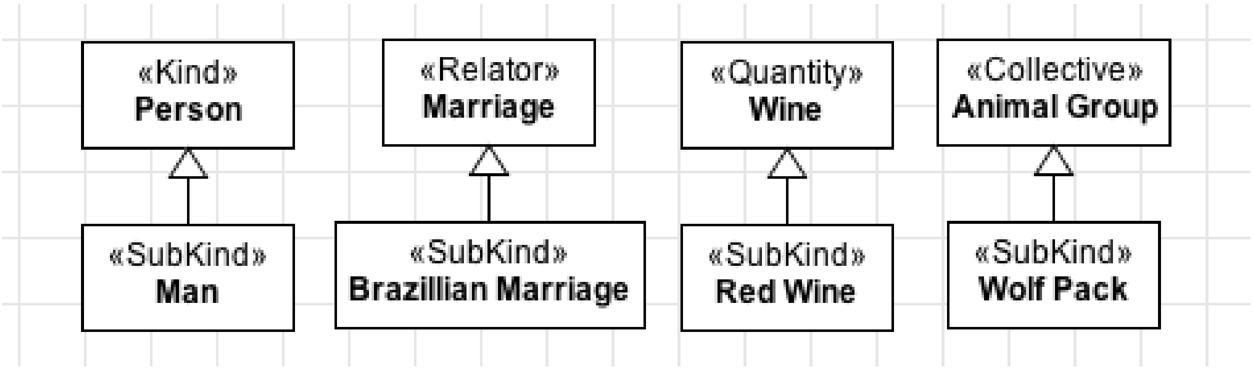
3.2.1 Definition

A «*Subkind*» is a construct used to represent *rigid* specializations of *identity providers* («*Kind*», «*Collective*», «*Quantity*», «*Relator*», «*Mode*» and «*Quantity*»). By default, its usage do not require a *relational dependency*. Let's see some examples:

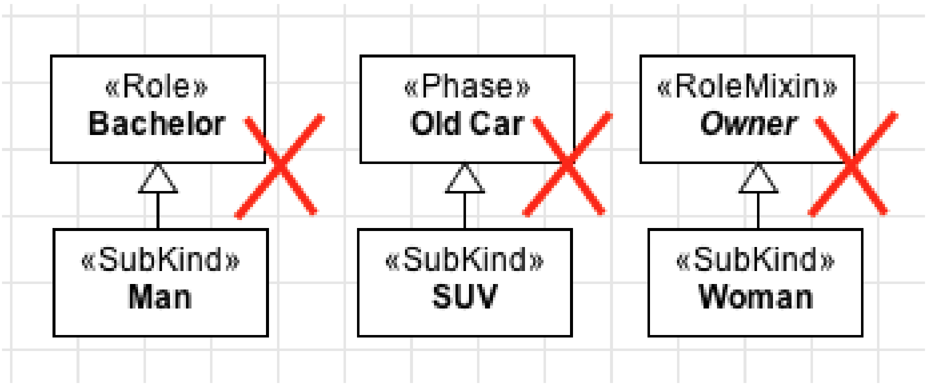


3.2.2 Constraints

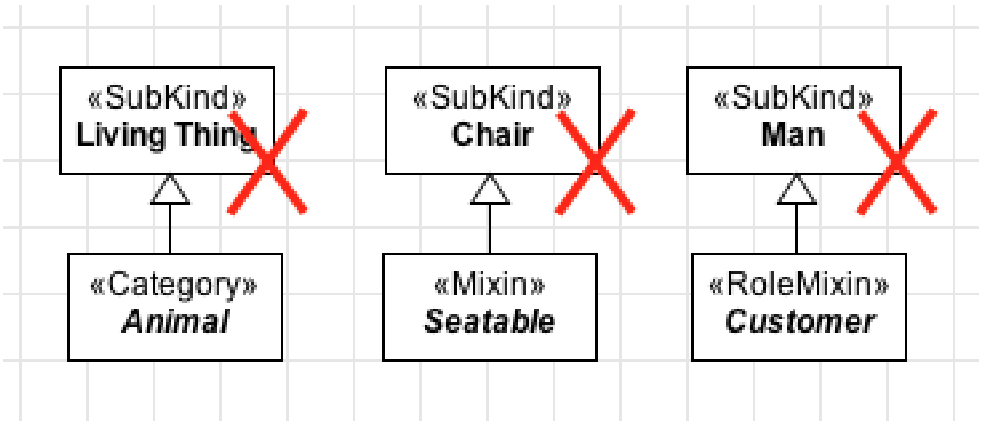
C1: A «*Subkind*» must **always** have exactly one *identity provider* («*Kind*», «*Collective*», «*Quantity*», «*Relator*», «*Mode*», «*Quantity*») as an ancestor (a direct or indirect super-type). Therefore, our examples in the first figure should be modelled as:



C2: Because it is a *rigid* type, a «*Subkind*» cannot have an *anti-rigid* type («*Role*», «*Phase*», «*RoleMixin*») as an ancestor. Therefore, the following fragments would not be allowed:



C3: Since every instance of a «*Subkind*» follows the same *identity principle*, a «*Subkind*» cannot have an mixin type («*Category*», «*Mixin*», «*RoleMixin*») as a descendant, i.e., a direct or indirect subtype. Fragments like the ones below are not allowed:



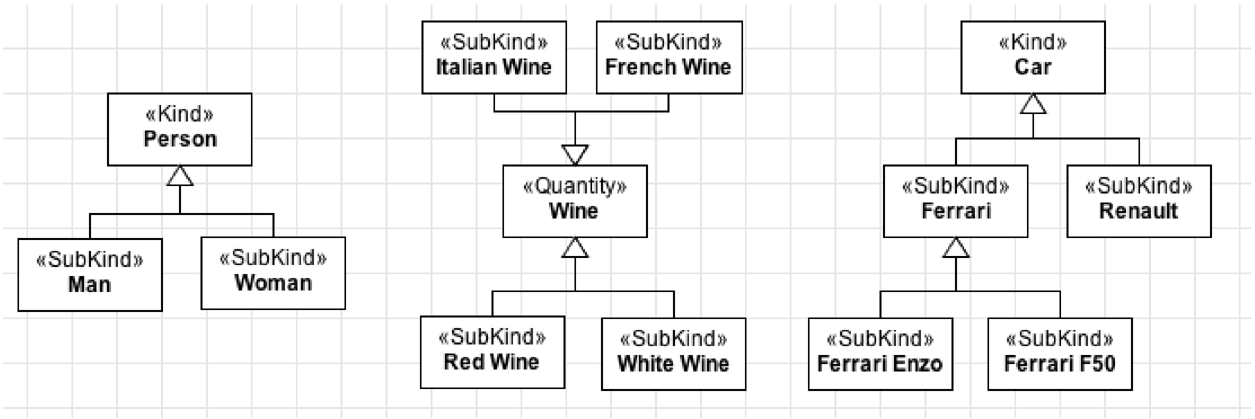
3.2.3 Common questions

Q1: Are subkinds only used to specialize kinds?

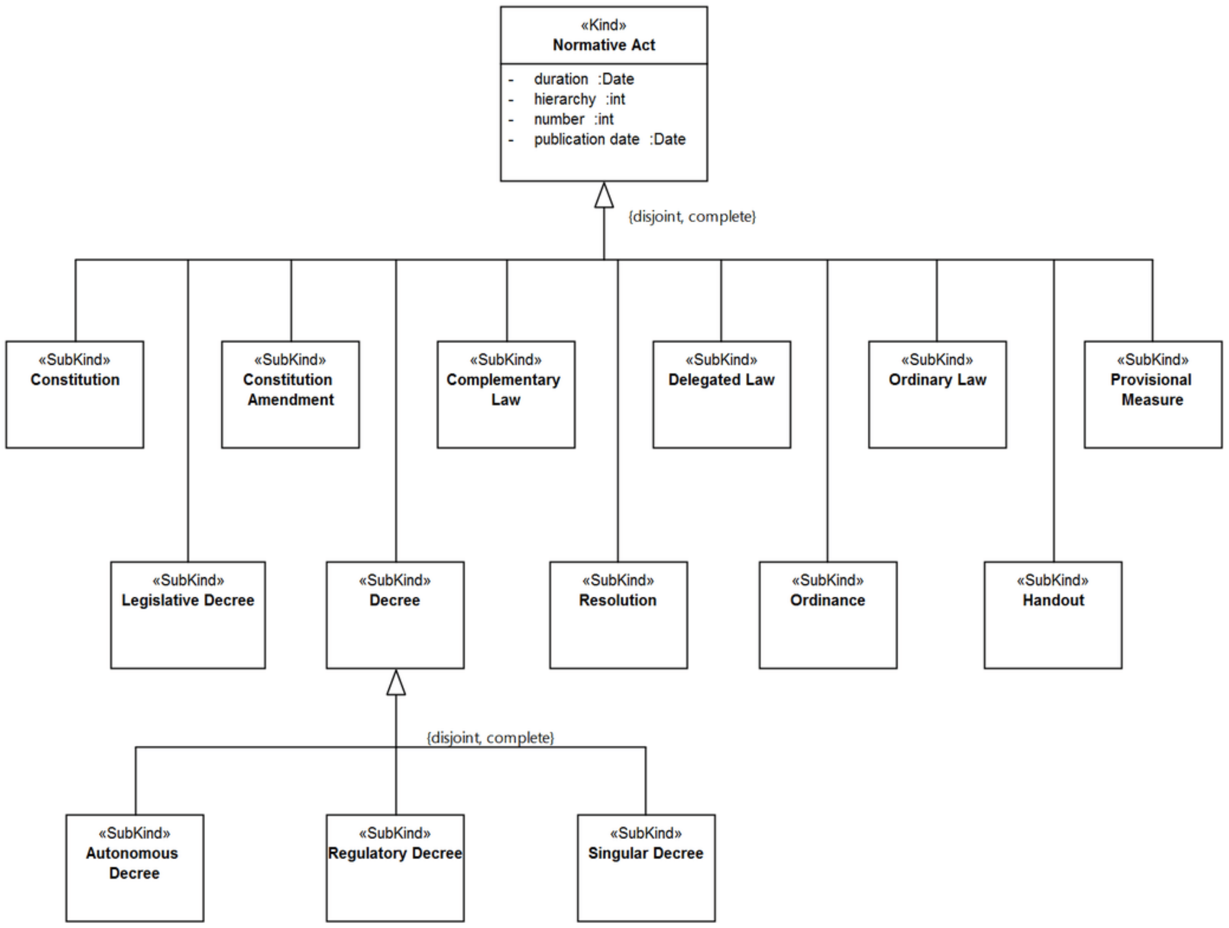
A1: No! Even though the name might be a little misleading, a «*Subkind*» may be used to specialize any identity provider, which includes «*Collective*», «*Quantity*» and «*Relator*».

3.2.4 Examples

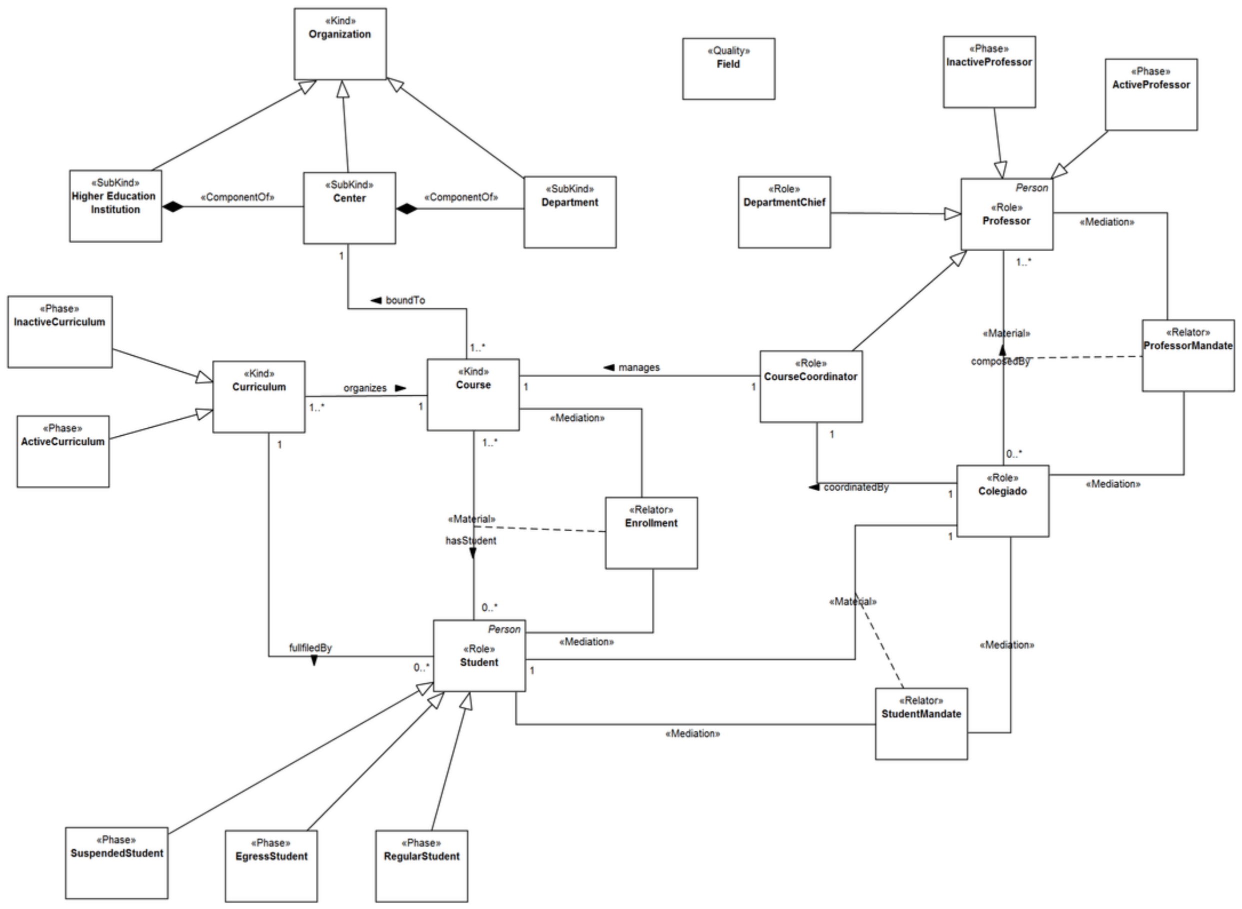
EX1: Usually, subkinds come in groups, like in the examples below:



EX2: Fragment from the Normative Acts Ontology (see more):



EX3: Fragment of a conceptual model about Brazilian Universities (see more):



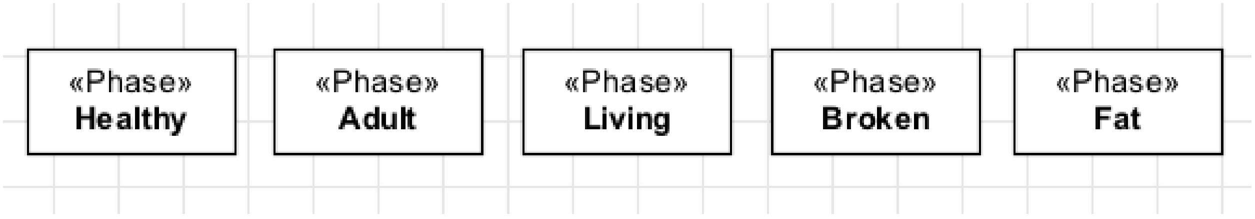
3.3 Phase

3.3.1 Definition

The «*Phase*» stereotype is used to represent *anti-rigid* subtypes of *identity providers* («*Kind*», «*Collective*», «*Quantity*», «*Relator*», «*Mode*» and «*Quantity*») that are instantiated by changes in intrinsic properties (e.g. the age of a person, the color of an object, the condition of a car). All instances of a particular «*Phase*» must follow the same *identity principle*. Phases always come in *partitions*.

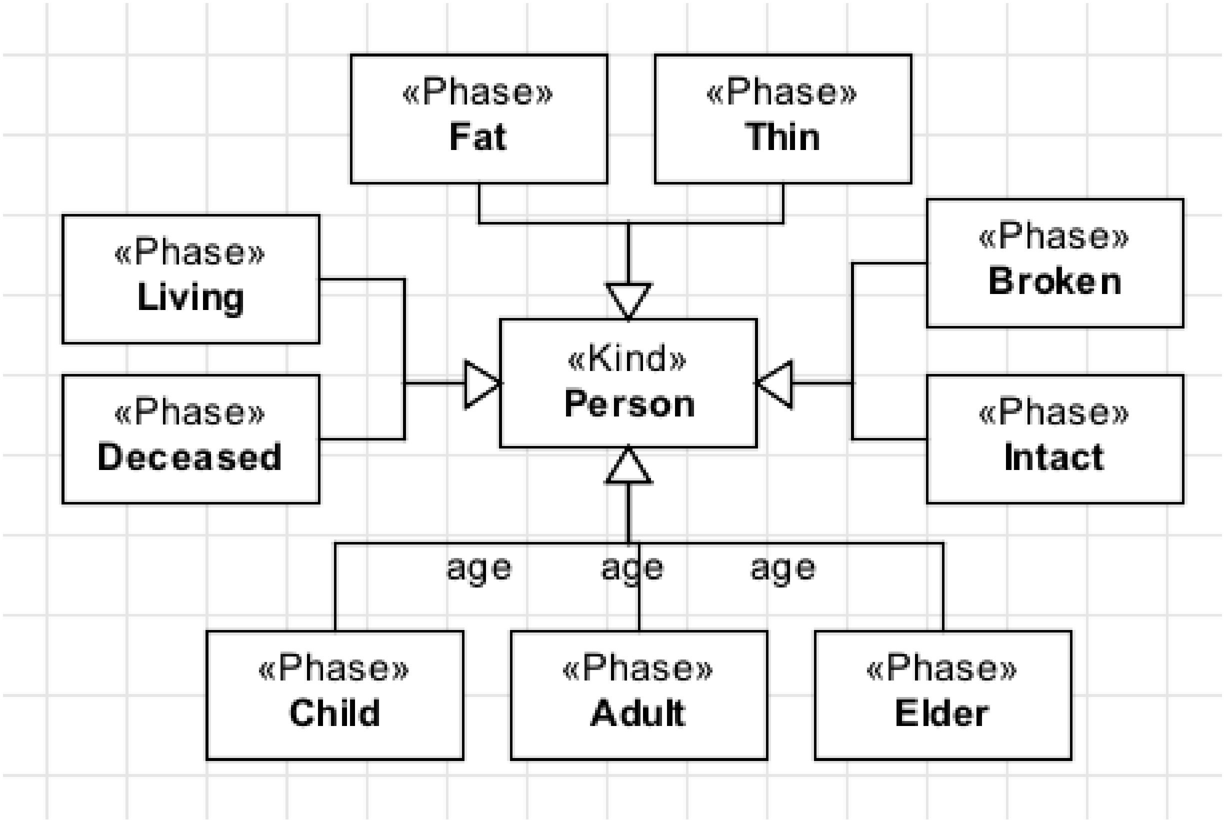
Note: *Tip*: When defining a phase *partition*, think about which property (or properties) variation is causing the instantiation of the phases and include it in your model. For instance, when defining the phases Child, Adult and Elder for Person, you should include an age property for the type Person.

Here are some examples of phases:

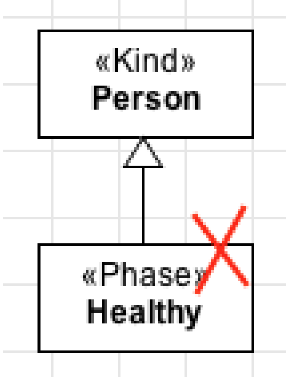


3.3.2 Constraints

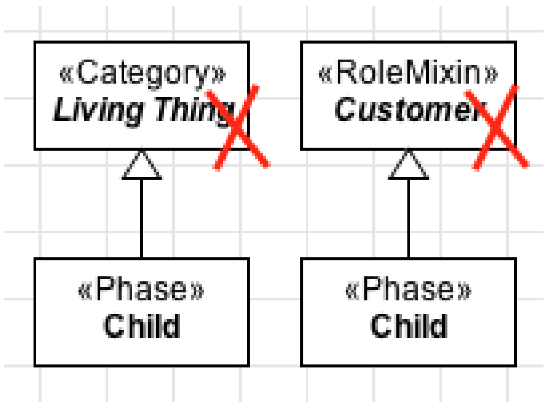
C1: A «*Phase*» must always have **exactly one identity provider** («*Kind*», «*Collective*», «*Quantity*», «*Relator*», «*Mode*», «*Quantity*») as an ancestor (a direct or indirect super-type). Our examples above should be modelled as:



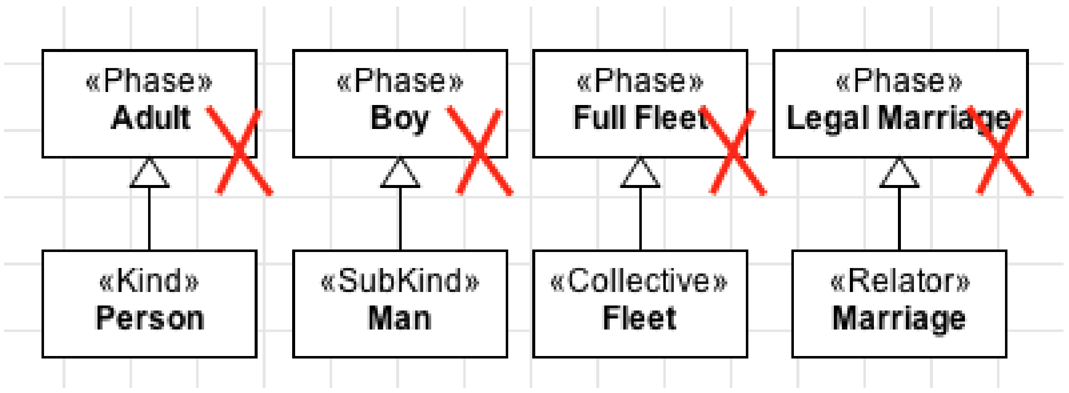
C2: A «*Phase*» must always be part of a partition (a generalization set disjoint and complete). Modeling a «*Phase*» as in example below is forbidden:



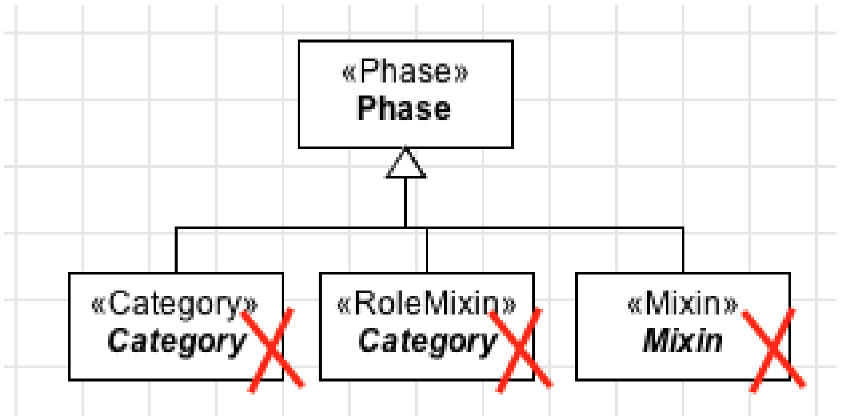
C3: A «*Phase*» cannot be a direct subtype of a «*RoleMixin*» or «*Category*».



C4: A «*Phase*» cannot be a super-type of a *rigid* type («*Kind*», «*Collective*», «*Quantity*», «*Relator*», «*Mode*», «*Quantity*», «*Subkind*», «*Category*»).



C5: A «*Phase*» cannot be a super-type of a mixin type («*Category*», «*RoleMixin*», «*Mixin*»).



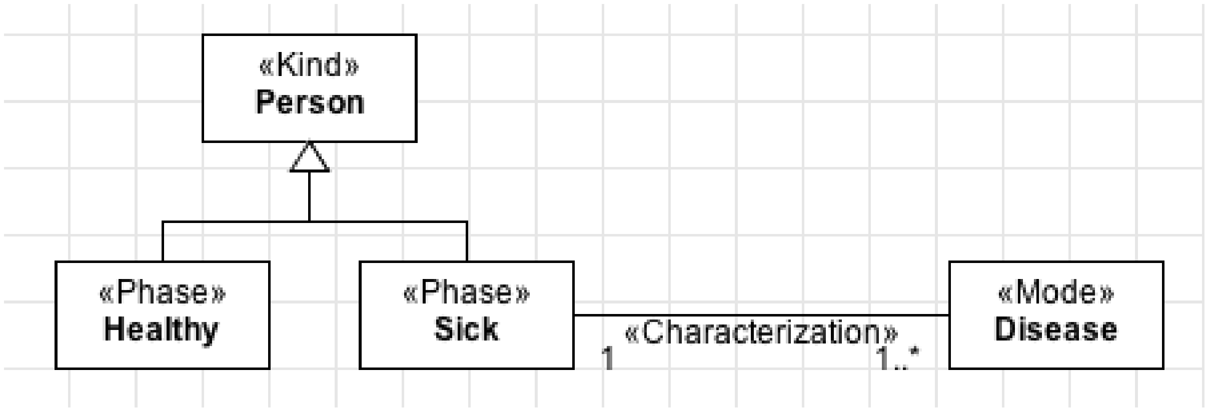
3.3.3 Common questions

Q1: Do I have to represent the intrinsic property which is affecting the instantiation of the phase?

A1: No, OntoUML does not require you to do that. However, whenever it is possible, you should represent everything needed to define the phase. On one hand, if you want to represent the Living and Deceased phases of a Person, it is ok. On the other hand, if representing Adult and Child, your model would be a lot more precise if you include the age property on your model and the OCL constraint defining the instantiation of the two phases.

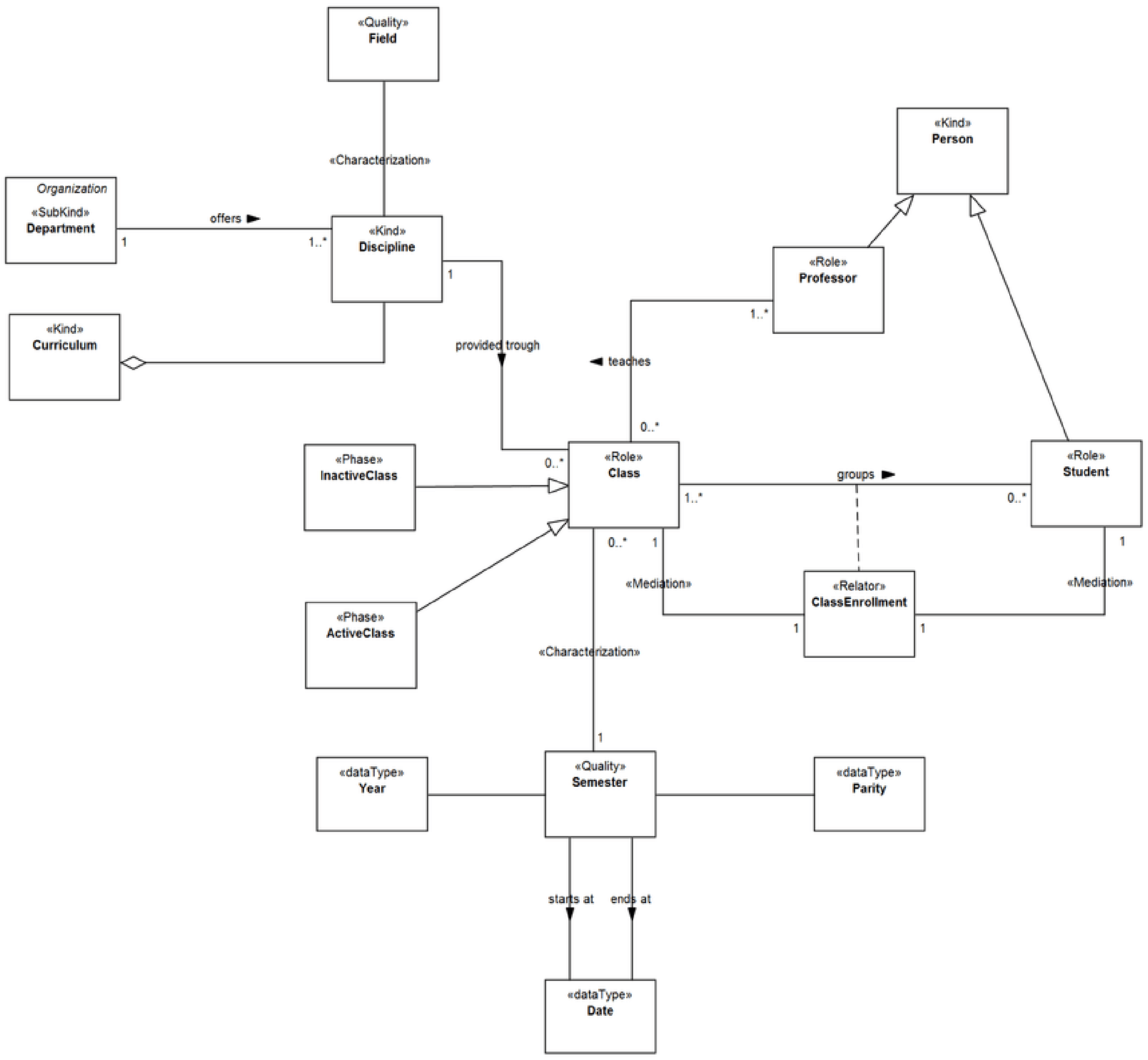
Q2: Can I define phases using modes?

A2: Yes. The fragment below is an example of how to do that.



3.3.4 Examples

EX1: Conceptual model about Brazilian Universities (see more):

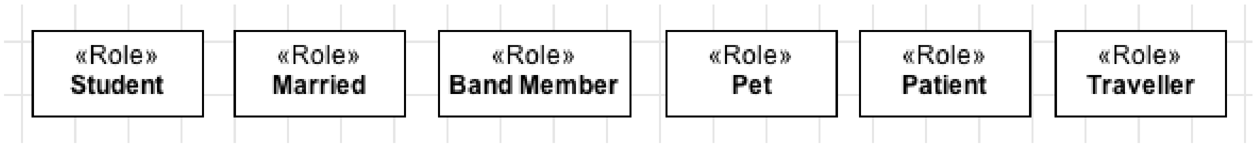


Errata: Phase as subtype of Role (Class), no multiplicity on part-whole, not marked as material and multiplicity does not correspond with mediations, Role (Professor) has optional relation, no multiplicity on <<characterization>> relation with Field Quality, (Department gets identity from kind in different diagram), Class has no identity

3.4 Role

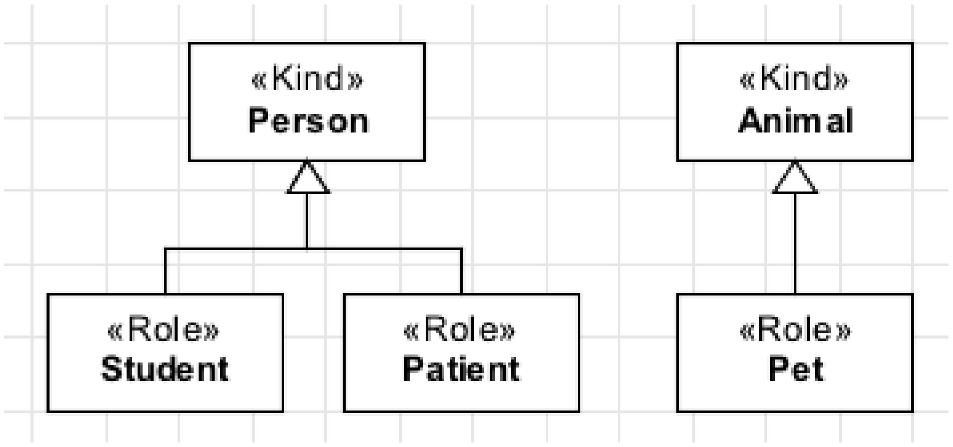
3.4.1 Definition

A «*Role*» is a construct used to represent *anti-rigid* specializations of *identity providers* («*Kind*», «*Collective*», «*Quantity*», «*Relator*», «*Mode*» and «*Quantity*») that are instantiated in *relational contexts*. All instances of a particular «*Role*» must follow the same *identity principle*. Here are some examples of roles:

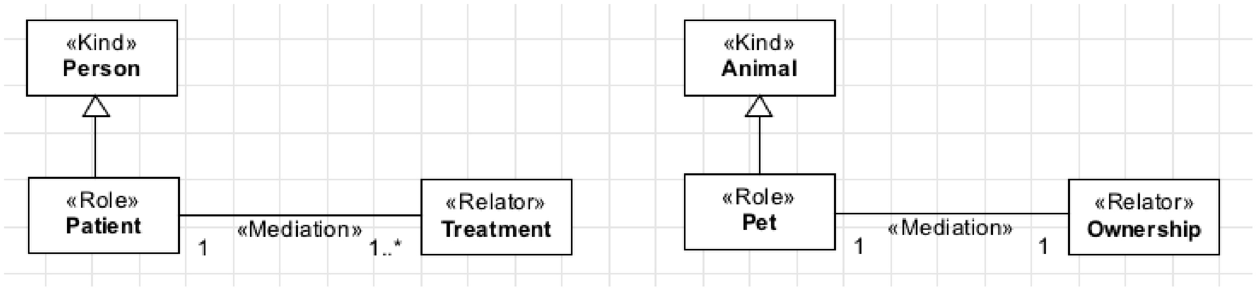


3.4.2 Constraints

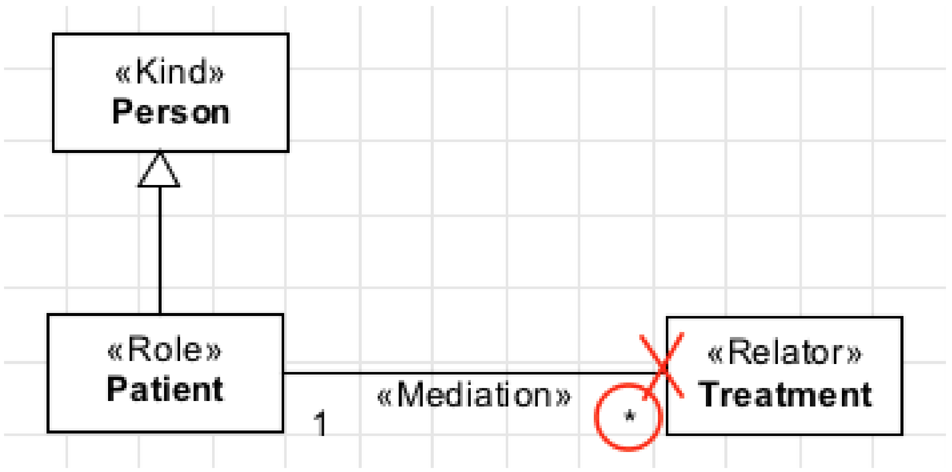
C1: A «*Role*» must always have **exactly one** *identity provider* («*Kind*», «*Collective*», «*Quantity*», «*Relator*», «*Mode*», «*Quantity*») as an ancestor (a direct or indirect super-type). To model our list of roles presented above, we should given them *identity providers*:



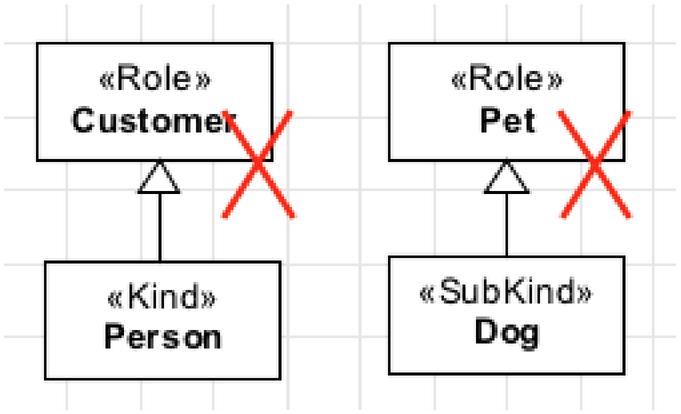
C2: Every «*Role*» must be connected, directly or indirectly, to a «*Mediation*» relation, since it is a *relationally dependent* construct. Continuing our example above, we should do the following:



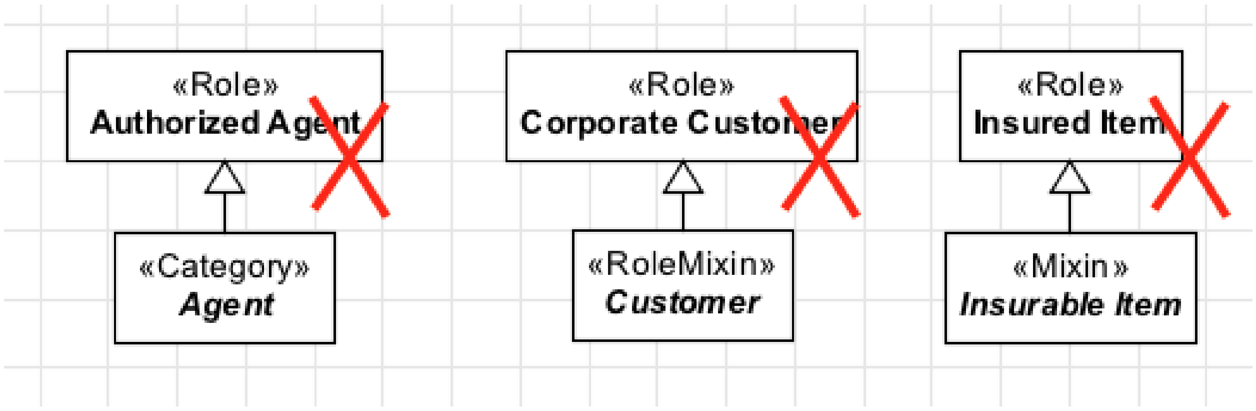
Remember that you can't defined a *relational dependency* with a minimum cardinality of zero. Therefore, the fragment below is wrong!



C3: A «*Role*» cannot be a supertype of a rigid type («*Kind*», «*Subkind*», «*Collective*», «*Quantity*», «*Relator*», «*Category*»).



C4: A «*Role*» cannot be a supertype of a mixin types («*Category*», «*RoleMixin*», «*Mixin*»).



3.4.3 Common questions

Q1: Can I define multiples *dependencies* for a «*Role*»?

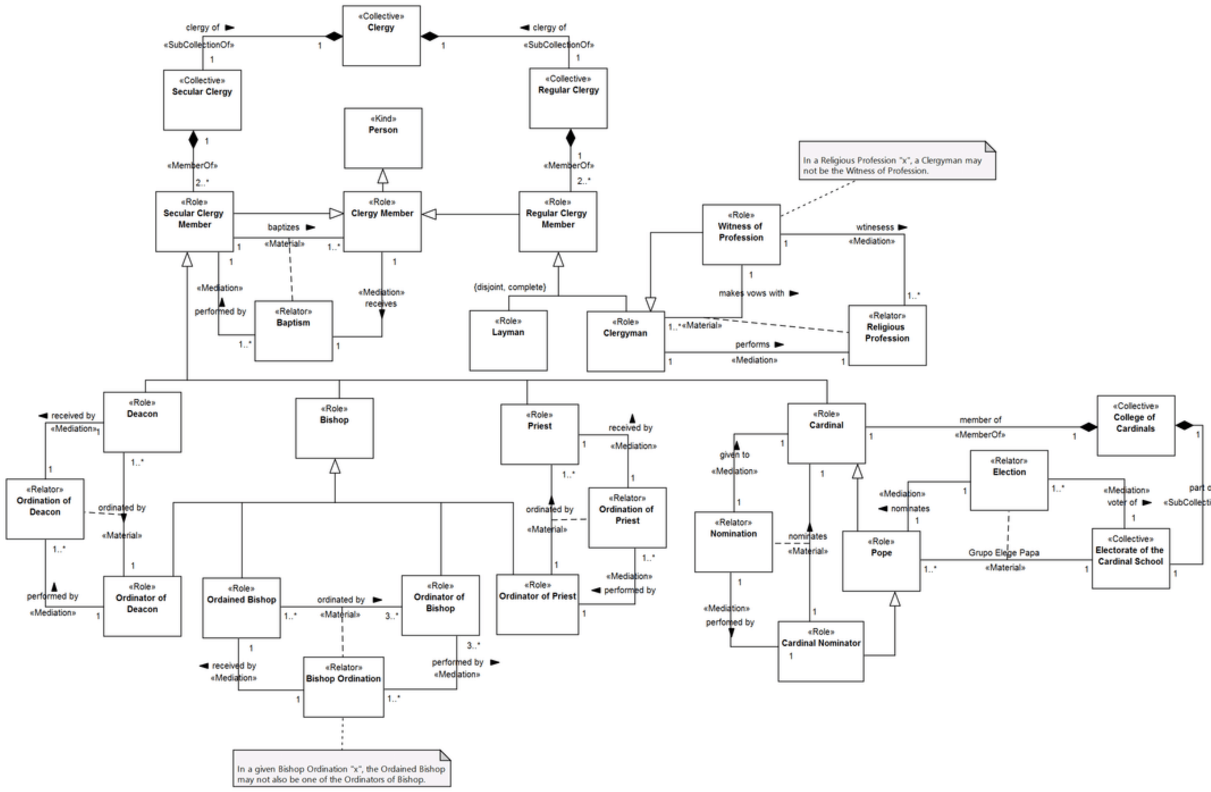
A1: Yes, there is no restriction in the number of *dependencies* one can define for a «*Role*». However, think carefully before doing so. You might be adding some unwanted instantiations to your ontology. This is an Ontological Anti-Pattern, called Multiple Dependency (read more about it here)

Q2: Can a «*Role*» be used to specialize another «*Role*»?

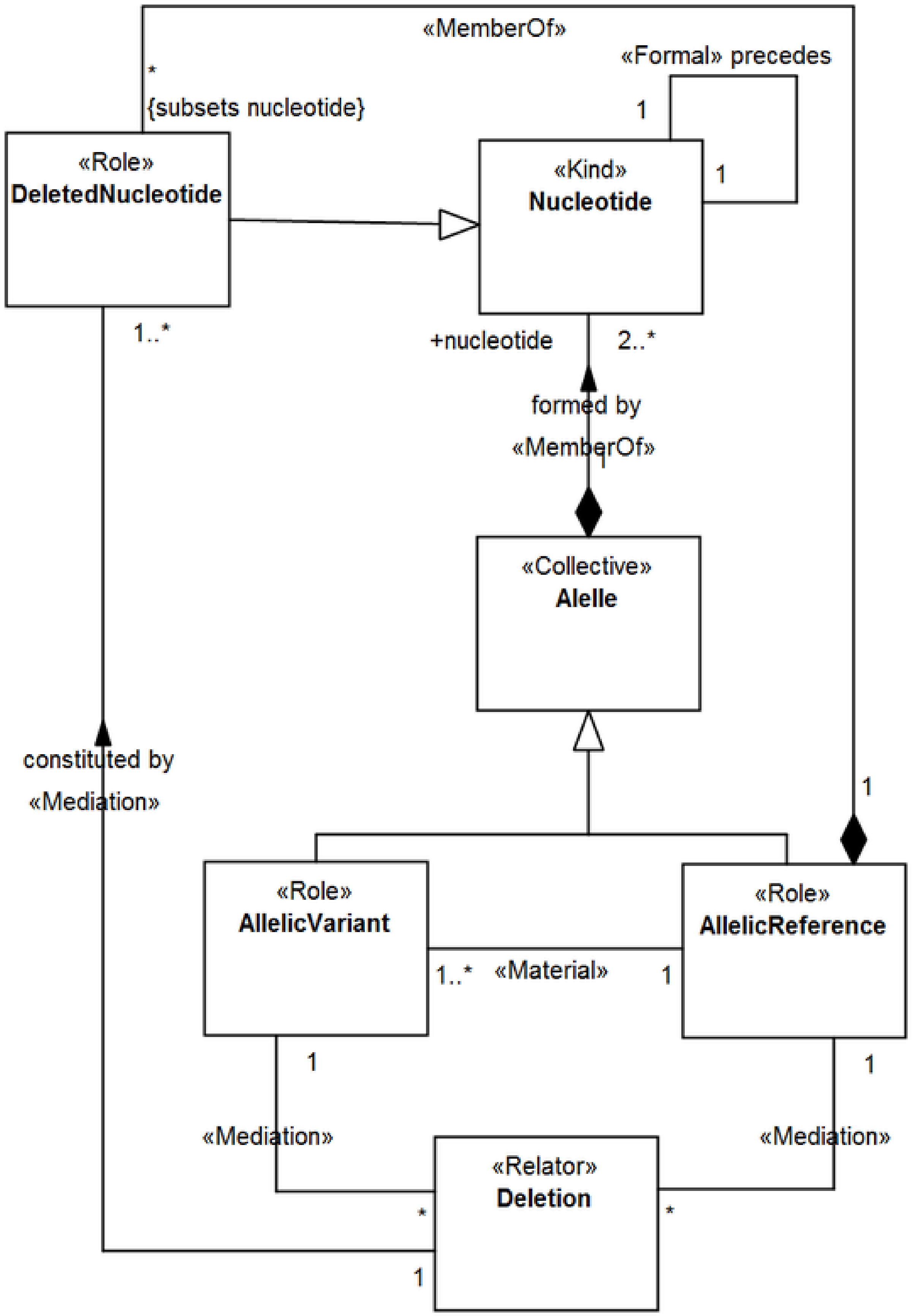
A2: Yes, there is no restriction regarding it. Again, take care when doing so. Since the language only require at least one *indirect dependency* for a «*Role*», you might forget to define additional *dependencies* for the sub-types.

3.4.4 Examples

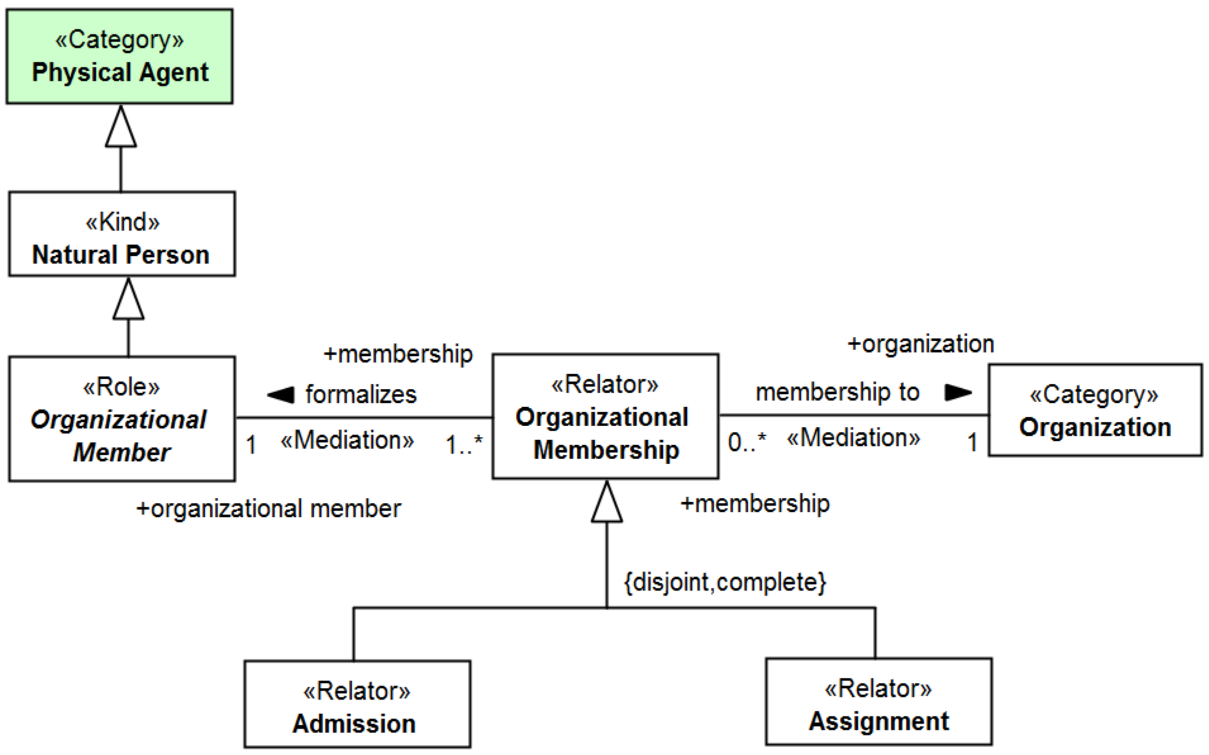
EX1: Conceptual model about roles in the Catholic clergy (see more):


«Collective» Clergy
clergy of
«SubCollectionOf»
«Collective» Secular Clergy
«Collective» Regular Clergy
«Kind» Person
«MemberOf»
«Role» Secular Clergy Member
«Role» Clergy Member
«Role» Regular Clergy Member
baptizes
«Material»
«Mediation»
performed by
«Relator» Baptism
receives
In a Religious Profession "x", a Clergyman may not be the Witness of Profession.
«Role» Witness of Profession
witnesess
makes vows with
{disjoint, complete}
«Role» Layman
«Role» Clergyman
«Relator» Religious Profession
performs
«Role» Deacon
received by
«Relator» Ordination of Deacon
ordinated by
«Role» Ordinator of Deacon
«Role» Bishop
«Role» Ordained Bishop
«Role» Ordinator of Bishop
«Relator» Bishop Ordination
In a given Bishop Ordination "x", the Ordained Bishop may not also be one of the Ordinators of Bishop.
«Role» Priest
«Relator» Ordination of Priest
«Role» Ordinator of Priest
«Role» Cardinal
member of
«Collective» College of Cardinals
given to
«Relator» Nomination
nominates
«Relator» Election
voter of
part of
«Role» Pope
Grupo Elege Papa
«Collective» Electorate of the Cardinal School
«Role» Cardinal Nominator

EX2: Fragment from an ontological analysis of a Human Genome scheme (see more):


«MemberOf»
*
{subsets nucleotide}
«Formal» precedes
1
1
«Role»
DeletedNucleotide
«Kind»
Nucleotide
1..*
+nucleotide
2..*
formed by
«MemberOf»
1
«Collective»
Alelle
constituted by
«Mediation»
1
«Role»
AllelicVariant
«Role»
AllelicReference
1..*
«Material»
1
1
1
«Mediation»
«Mediation»
«Relator»
Deletion
*
*
1

Errata: No material derivation, bad material multiplicity, bad memberOf multiplicity **EX3:** Fragment of the OntoUML Org Ontology (O3) (see more):

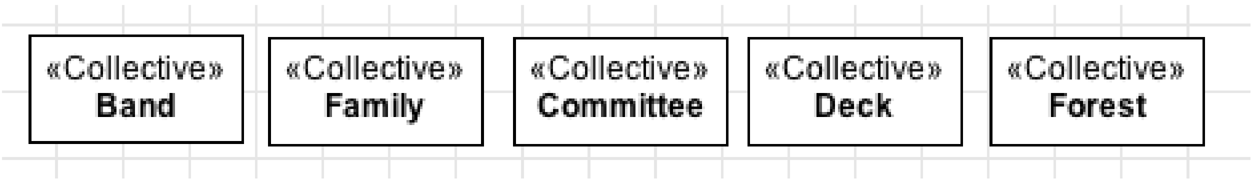


Errata: Relator cannot be subtype of Relator, Category not abstract and no subtypes (or just one), no material relation

3.5 Collective

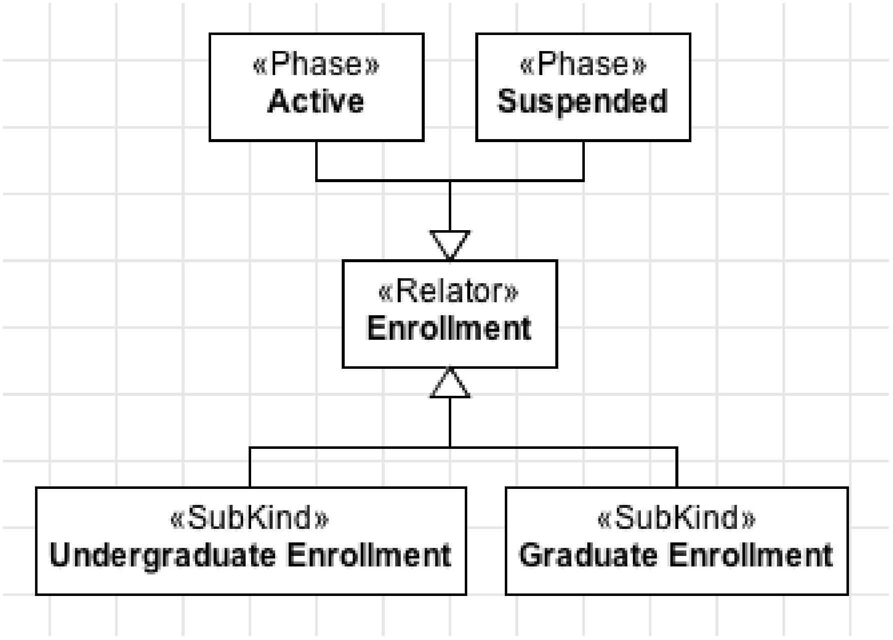
3.5.1 Definition

The «*Collective*» construct is used to represent *rigid* concepts that provide an *identity principle* for their instances. The main characteristic of a «*Collective*» is that it has an homogenous internal structure, i.e., all parts are perceived in the same way by the whole (see the «*MemberOf*» relation for more details about members of collections).



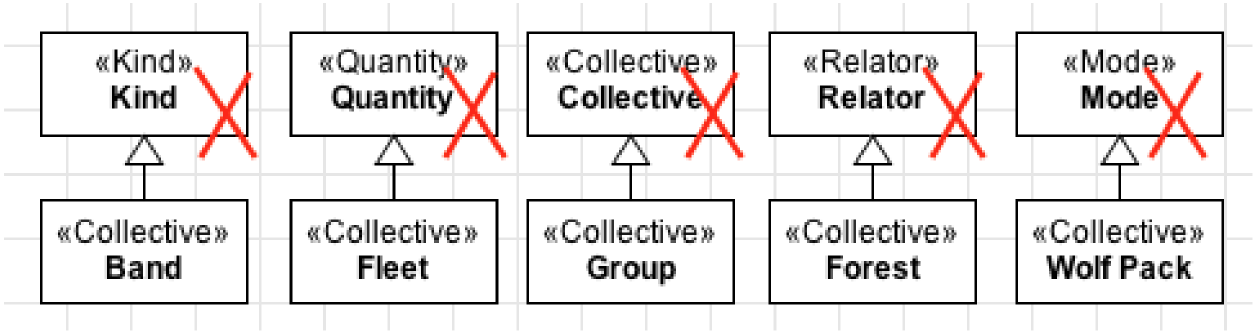
To decide whether or not to classify a concept as a collective, think about the relation between it has towards its parts (or members). Do all members are "equally perceived" by the whole (the collective)? In other words, do all members contribute in the same way to the functionality of the whole? If the answers are yes, you have a collective. It is important to keep in mind that some concepts, like Family or Fleet could be classified as both collectives and functional complexes. For instance, if we understand a family as a group of people with equal roles and responsibilities towards the family, we would say it is a collective. However, if we distinguish a person as the head of the family, and another as being responsible for the family's income, we would say that a family is a functional complex.

As the other identity provider stereotypes («*Kind*», «*Quality*», «*Relator*» and «*Mode*»), a «*Collective*» can be specialized by *subkinds*, *phases* and *roles*, as well as generalized by *mixins* and *categories*.

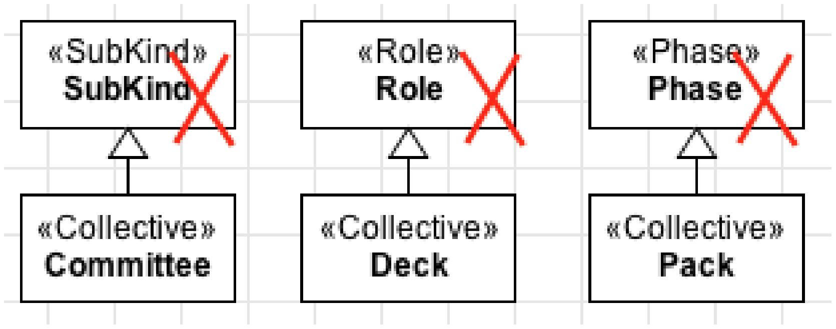


3.5.2 Constraints

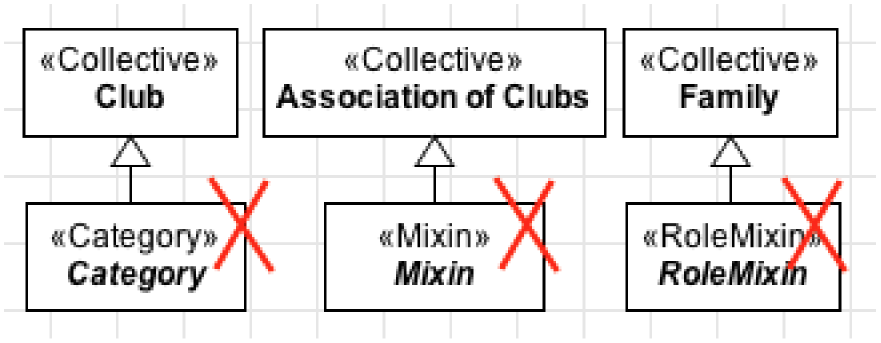
C1: A «*Collective*» cannot have an *identity* provider («*Kind*», «*Collective*», «*Quantity*», «*Relator*», «*Mode*» and «*Quantity*») as its direct or indirect super-type.



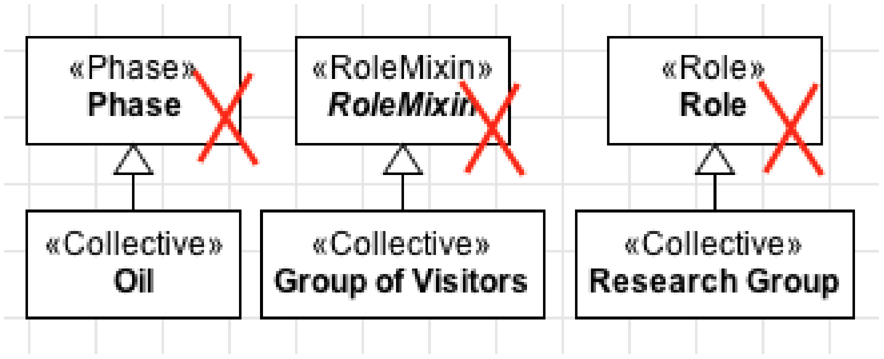
C2: A «*Collective*» cannot have types that inherit *identity* («*Subkind*», «*Role*» and «*Phase*») as its direct or indirect super-types.



C3: A «*Collective*» cannot have types that aggregate individuals with *different identity principles* («*Category*», «*RoleMixin*» and «*Mixin*») as its direct or indirect subtypes.



C4: As a *rigid* type, a «*Collective*» cannot have any *anti-rigid* type («*Role*», «*RoleMixin*» and «*Phase*») as its direct or indirect super-type.

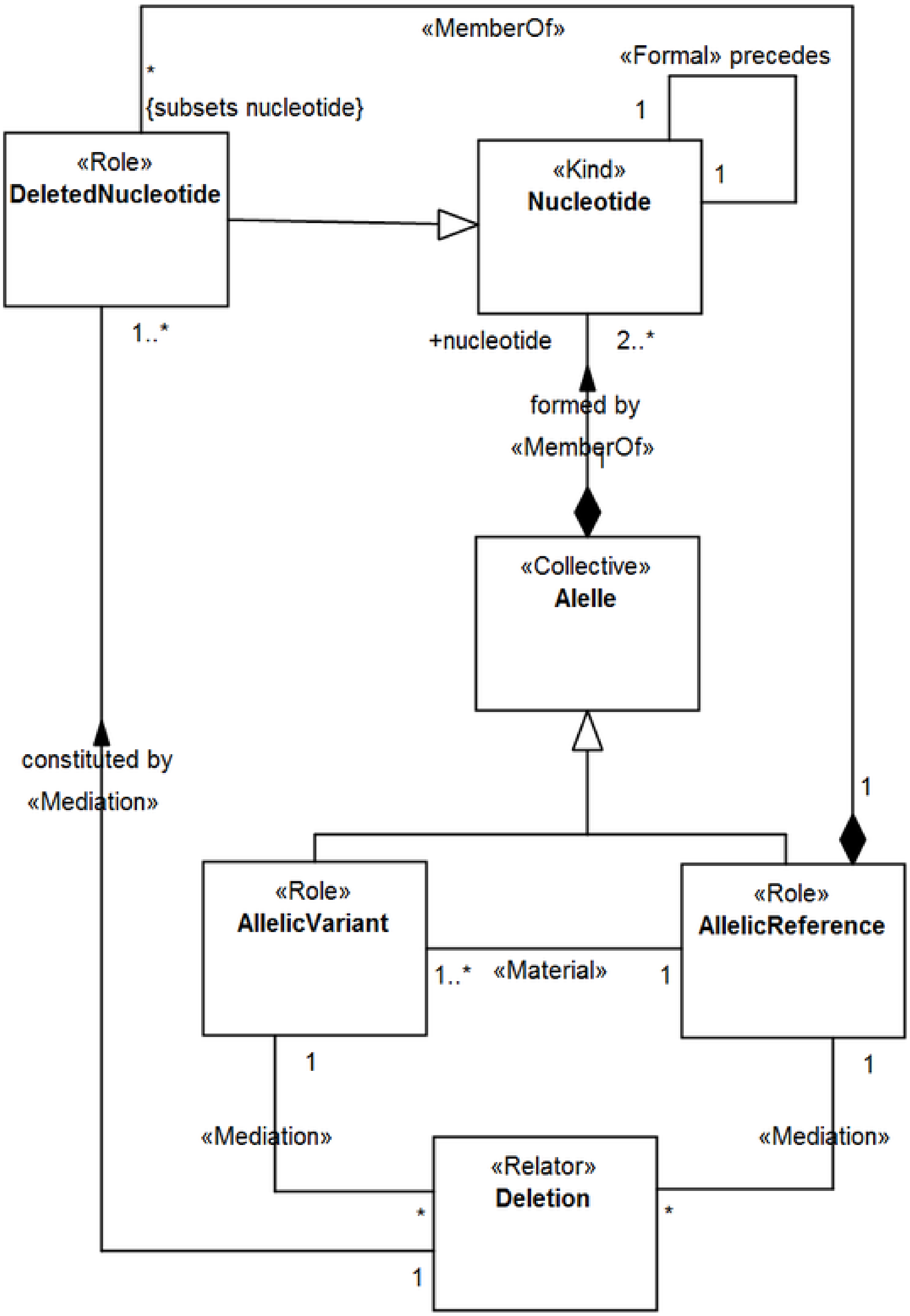


3.5.3 Common questions

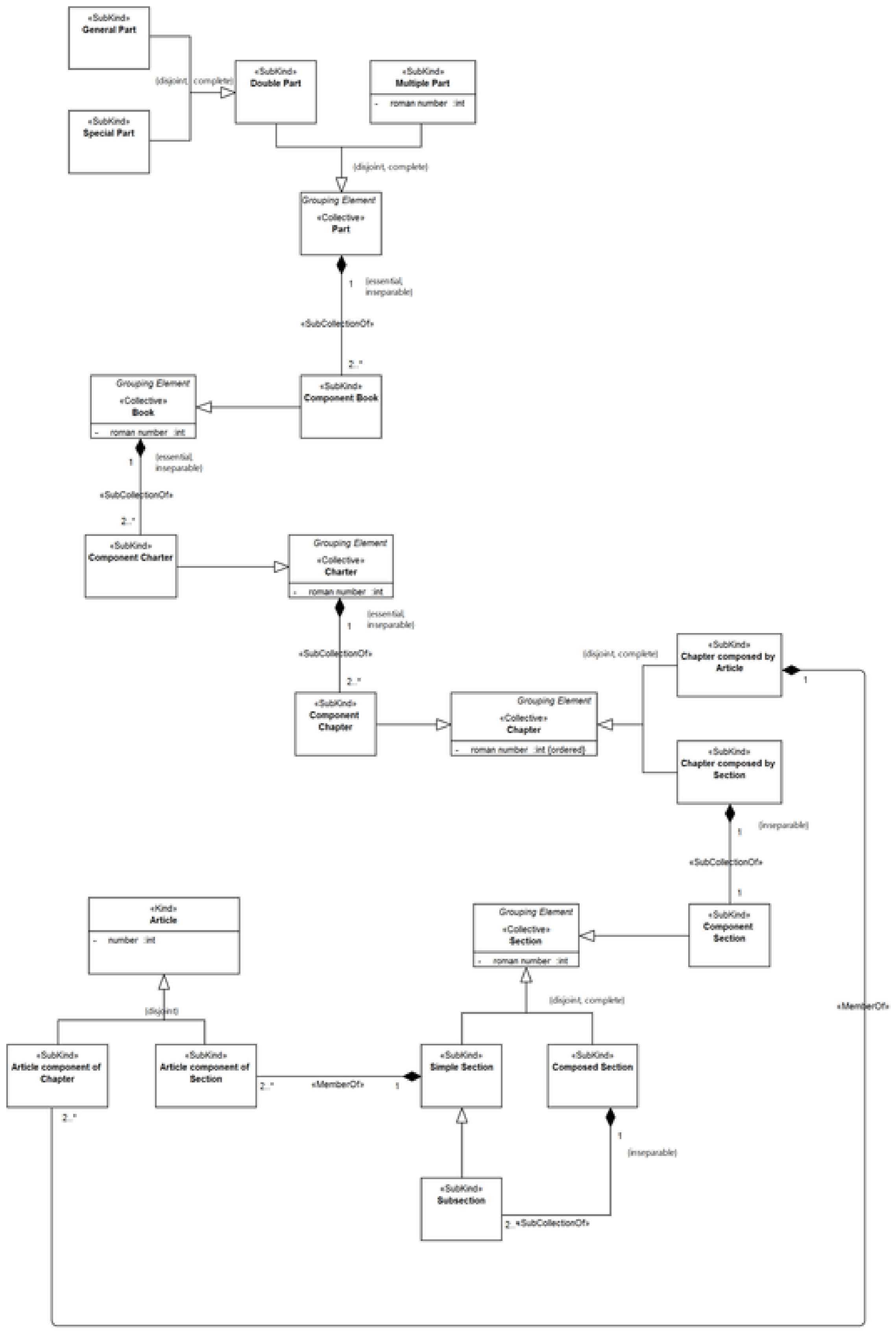
Ask us some question if something is not clear . . .

3.5.4 Examples

EX1: Fragment from the a conceptual model about the human genome (see more):



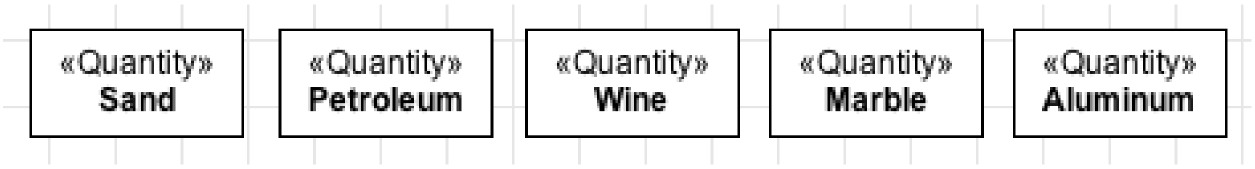
EX2: Fragment from the Normative Acts Ontology (see more):


«SubKind» General Part
«SubKind» Special Part
{disjoint, complete}
«SubKind» Double Part
«SubKind» Multiple Part
- roman number : int
{disjoint, complete}
Grouping Element «Collective» Part
1 {essential, inseparable}
«SubCollectionOf»
2..*
«SubKind» Component Book
Grouping Element «Collective» Book
- roman number : int
1 {essential, inseparable}
«SubCollectionOf»
2..*
«SubKind» Component Charter
Grouping Element «Collective» Charter
- roman number : int
1 {essential, inseparable}
«SubCollectionOf»
2..*
«SubKind» Component Chapter
Grouping Element «Collective» Chapter
- roman number : int {ordered}
{disjoint, complete}
«SubKind» Chapter composed by Article
«SubKind» Chapter composed by Section
1 {inseparable}
«SubCollectionOf»
1
«SubKind» Component Section
Grouping Element «Collective» Section
- roman number : int
{disjoint, complete}
«Kind» Article
- number : int
{disjoint}
«SubKind» Article component of Chapter
«SubKind» Article component of Section
2..* «MemberOf» 1
«SubKind» Simple Section
«SubKind» Composed Section
1 {inseparable}
«SubKind» Subsection
2..* «SubCollectionOf»
2..*
«MemberOf»
1

3.6 Quantity

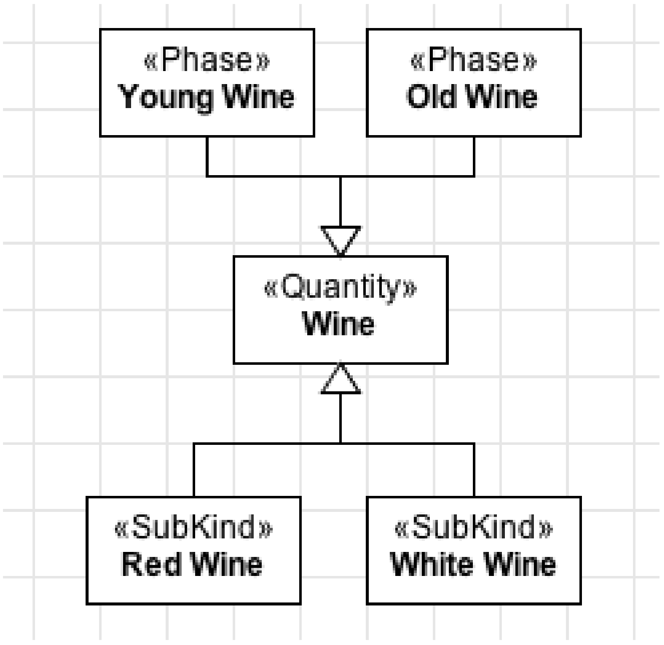
3.6.1 Definition

The «*Quantity*» construct is used to represent *rigid* concepts that provide an *identity principle* for their instances. A «*Quantity*» represent uncountable things, like Water, Clay, or Beer. It represents a maximally topologically connected amount of matter. Quantities only have other quantities as parts (see the «*SubQuantityOf*» relation for more details about members of collections). Here are some examples:



An easy way to decide whether a concept is a quantity or not, as yourself this: if you physically divide an instance of 'x' in two parts, are the resulting individuals two new instances of x? What if you divide another 5 or 10 times? If the answer is always yes, 'x' is a Quantity. To exemplify, let's think about an pile of sand. If you divide the pile in two, you now have to new piles of sand, right? What if you do that again for each remaining part? We would have 4 piles of sand.

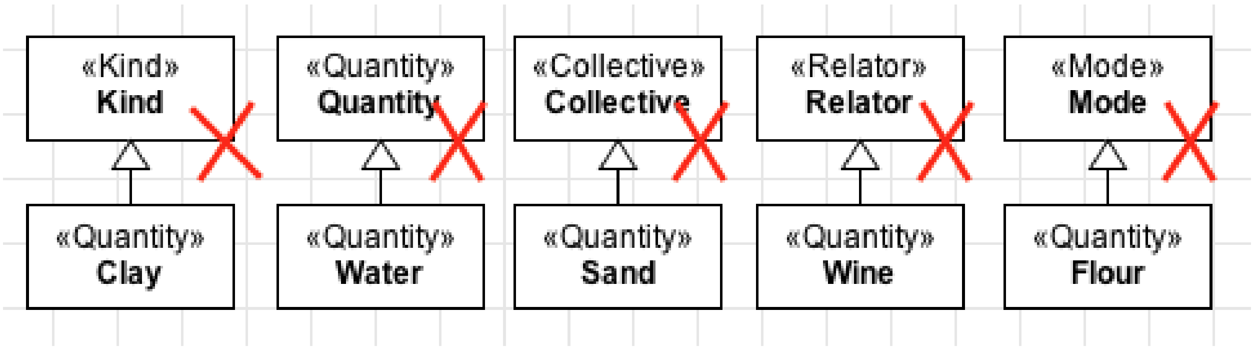
As the other identity provider stereotypes («*Kind*», «*Collective*», «*Relator*» and «*Mode*»), a Quality can be specialized by *subkinds*, *phases* and *roles*, as well as generalized by *mixins* and *categories*.



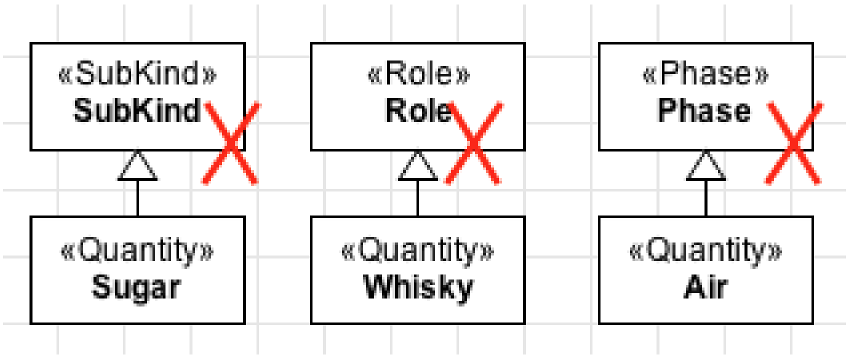
Be careful not to confuse «*Quantity*» and «*Quality*».

3.6.2 Constraints

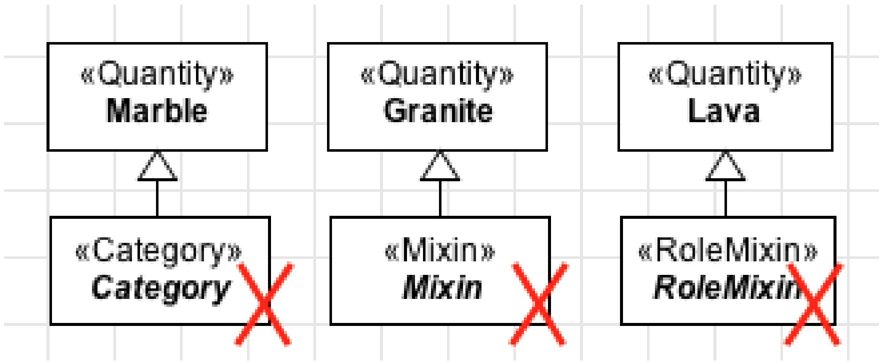
C1: A «*Quantity*» cannot have an *identity provider* («*Kind*», «*Collective*», «*Quantity*», «*Relator*», «*Mode*» and «*Quantity*») as its direct or indirect super-type.



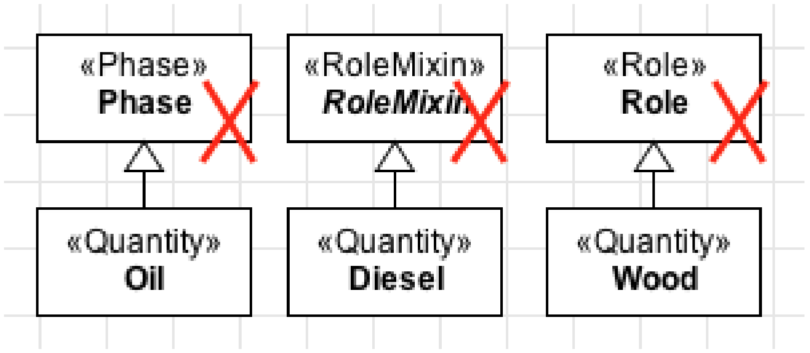
C2: A «*Quantity*» cannot have types that inherit *identity* («*Subkind*», «*Role*» and «*Phase*») as its direct or indirect super-types.



C3: A «*Quantity*» cannot have types that aggregate individuals with different *identity principles* («*Category*», «*RoleMixin*» and «*Mixin*») as its direct or indirect subtypes.



C4: As a *rigid* type, a «*Quantity*» cannot have any *anti-rigid* type («*Role*», «*RoleMixin*» and «*Phase*») as its direct or indirect super-type.



3.6.3 Common questions

Ask us some question if something is not clear …

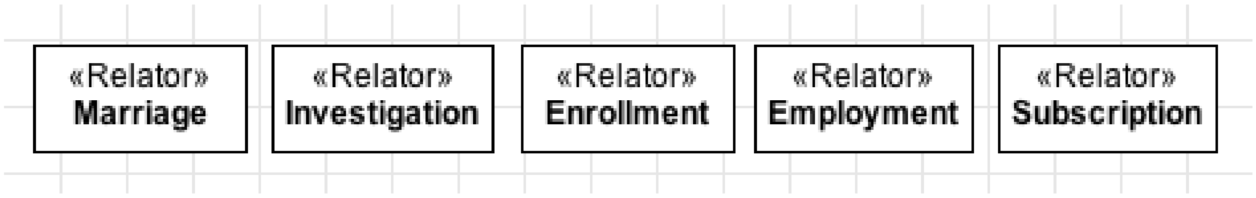
3.6.4 Examples

Todo: To be added …

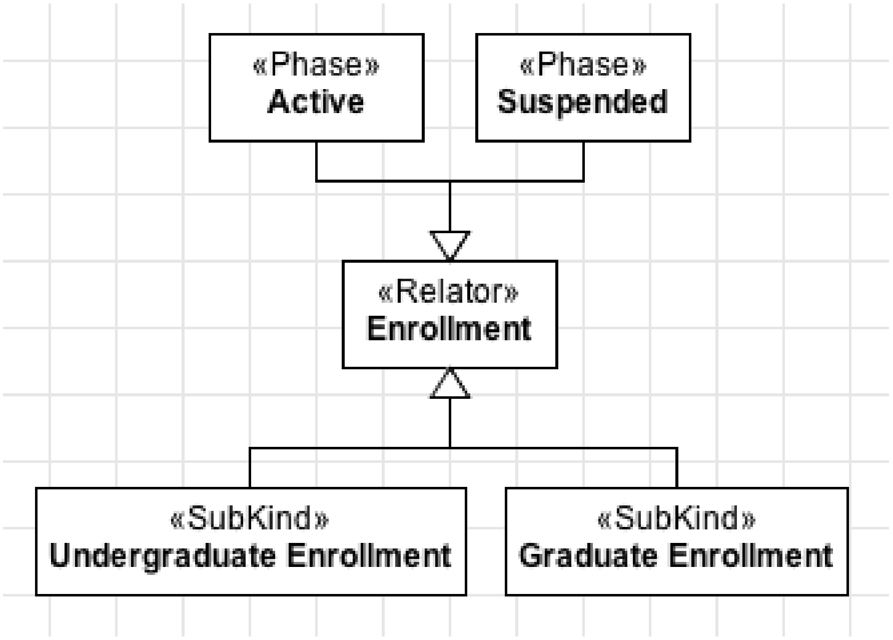
3.7 Relator

3.7.1 Definition

The «*Relator*» construct is used to represent **truth-makers** of *material relations*, i.e., the "things" that must exist in order for two or more individuals to be connected by *material relations*. Because of this nature, relators are always *dependent* on other individuals to exist. Here are some examples of concepts classified as relators:

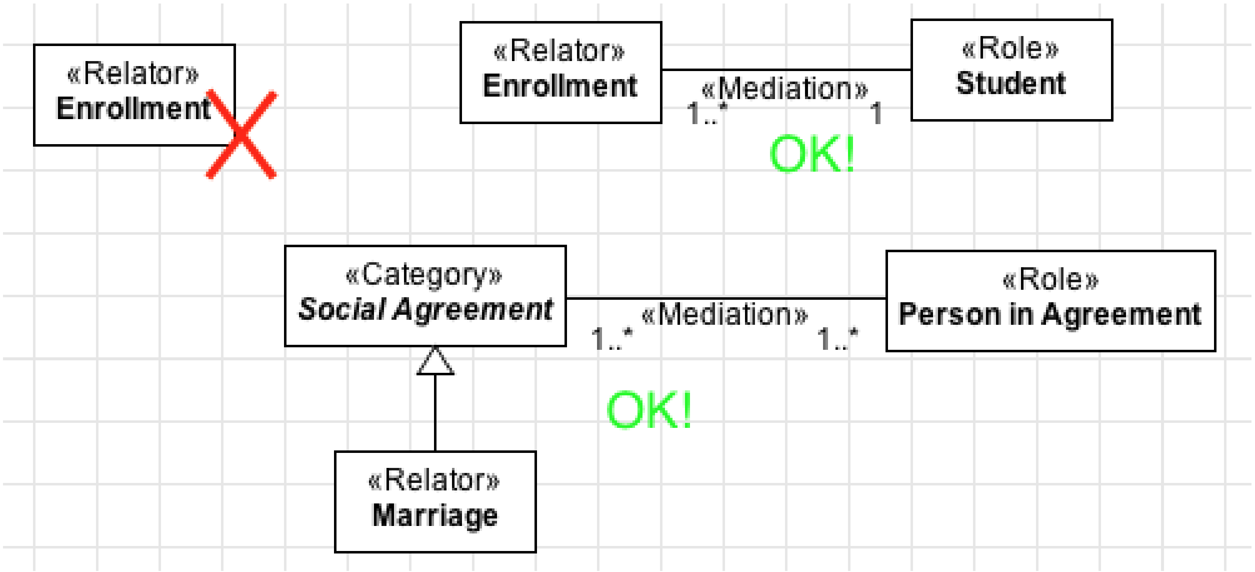


Note that the «*Relator*» meta-class is analogous to the «*Kind*», «*Collective*» and «*Quantity*» meta-classes, in the sense that it is *rigid* and provides an *identity principle* for its instances. The difference is that, instead of representing functional complexes, quantities or collections, a «*Relator*» represents the objectification of relational properties. The direct consequence is that relators can also be specialised by *subkinds*, *phases* and *roles*, and generalised by *categories* and *mixins*.

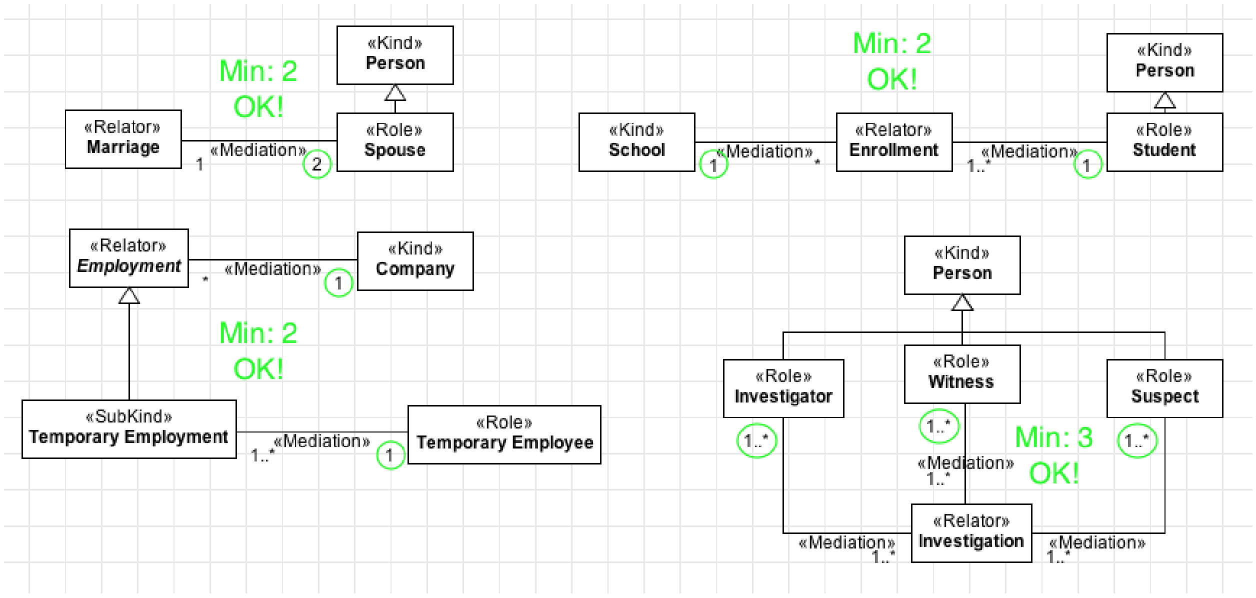

«Phase» Active
«Phase» Suspended
«Relator» Enrollment
«SubKind» Undergraduate Enrollment
«SubKind» Graduate Enrollment

3.7.2 Constraints

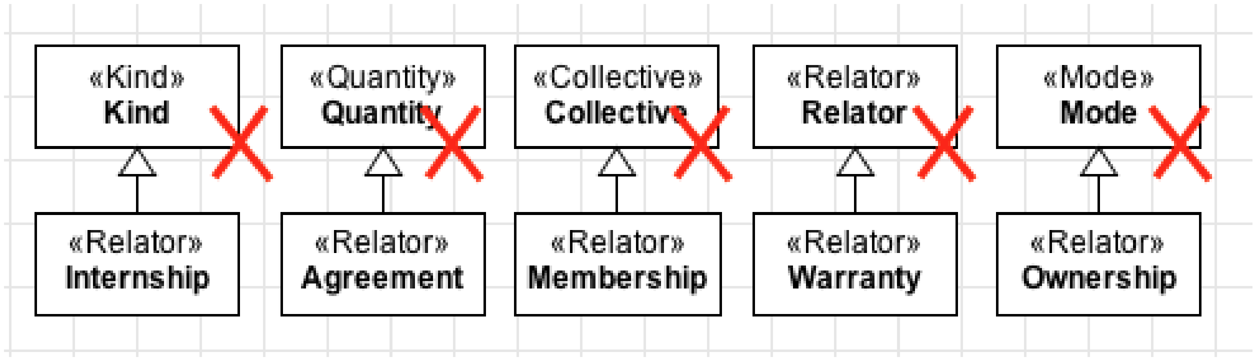
C1: A «*Relator*» must always be connected (directly or indirectly) to at least one relation stereotyped as «*Mediation*»



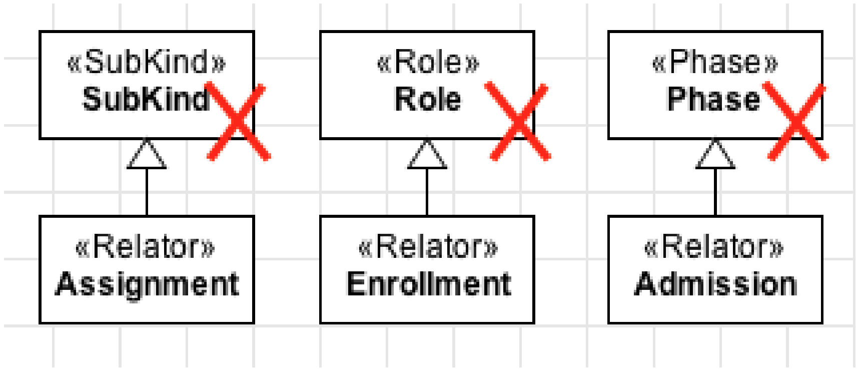
C2: The sum of the minimum cardinalities of the opposite ends of the *mediations* connected (directly or indirectly) to the «*Relator*» must be greater or equal to 2.



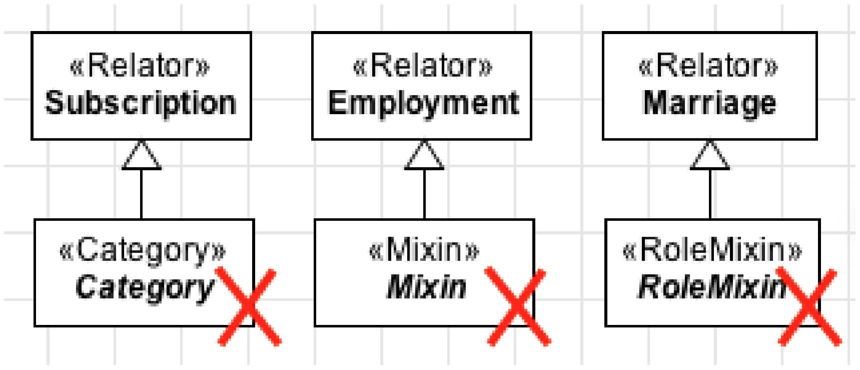
C3: A «*Relator*» cannot have an *identity provider* («*Kind*», «*Collective*», «*Quantity*», «*Relator*», «*Mode*» and «*Quantity*») as its direct or indirect super-type.



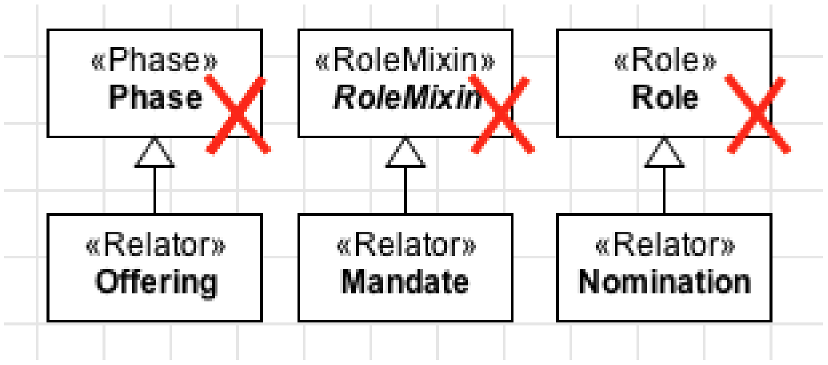
C4: A «*Relator*» cannot have types that inherit *identity* («*Subkind*», «*Role*» and «*Phase*») as its direct or indirect super-type.



C5: A «*Relator*» cannot have types that aggregate individuals with different *identity principles* («*Category*», «*RoleMixin*» and «*Mixin*») as its direct or indirect subtypes.



C6: As a *rigid* type, a «*Relator*» cannot have any *anti-rigid* type («*Role*», «*RoleMixin*» and «*Phase*») as its direct or indirect super-type.

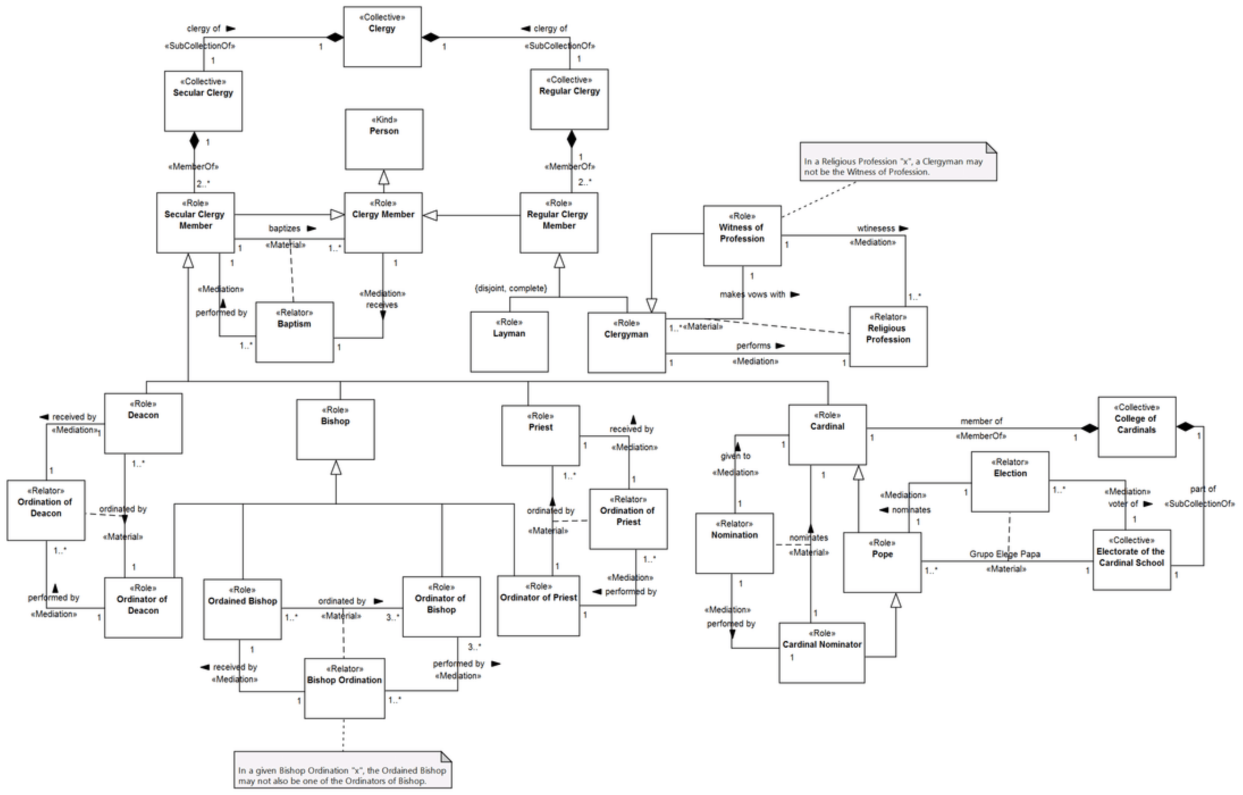


3.7.3 Common questions

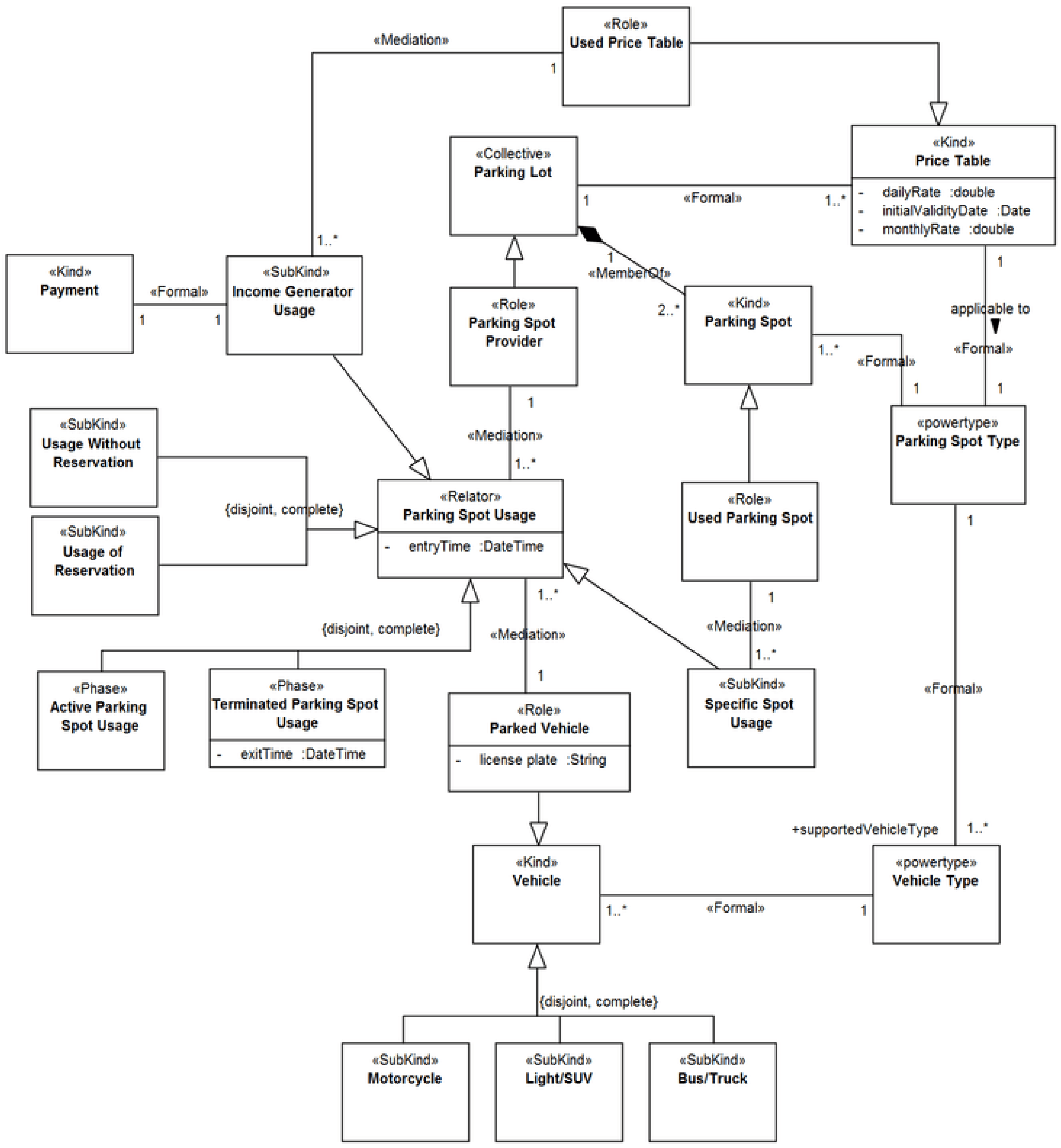
Ask us some question if something is not clear . . .

3.7.4 Examples

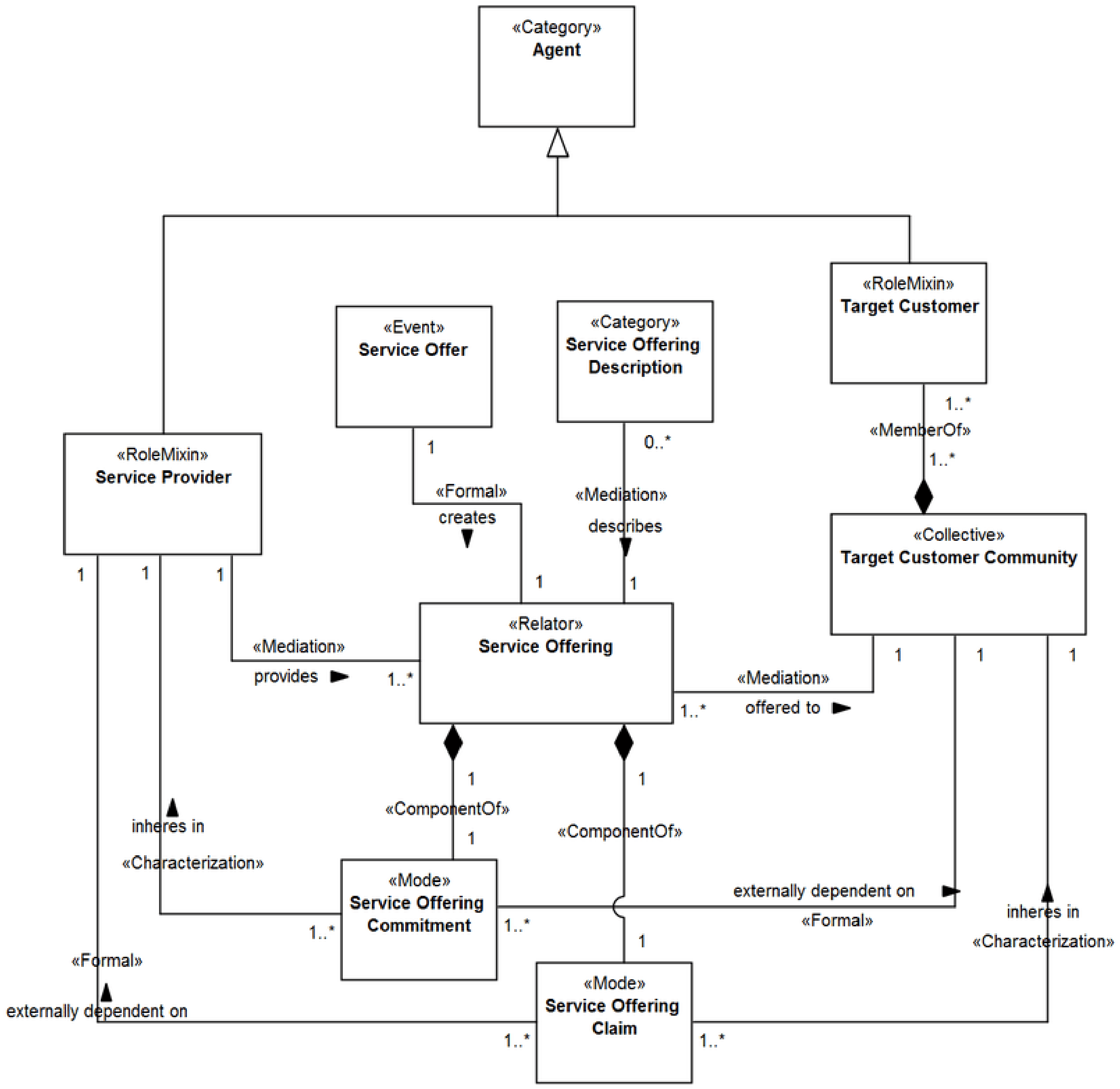
EX1: Conceptual model about the Catholic Clergy (see more):


«Collective» Clergy
clergy of
«SubCollectionOf»
«Collective» Secular Clergy
«Collective» Regular Clergy
«Kind» Person
«MemberOf»
2..*
«Role» Secular Clergy Member
«Role» Clergy Member
«Role» Regular Clergy Member
baptizes
«Material»
«Mediation»
performed by
receives
«Relator» Baptism
In a Religious Profession "x", a Clergyman may not be the Witness of Profession.
«Role» Witness of Profession
wtinesess
makes vows with
{disjoint, complete}
«Role» Layman
«Role» Clergyman
«Relator» Religious Profession
performs
«Role» Deacon
«Role» Bishop
«Role» Priest
«Role» Cardinal
«Collective» College of Cardinals
member of
received by
«Relator» Ordination of Deacon
ordinated by
«Relator» Ordination of Priest
given to
«Relator» Nomination
«Relator» Election
nominates
voter of
part of
«Role» Pope
Grupo Elege Papa
«Collective» Electorate of the Cardinal School
«Role» Ordinator of Deacon
«Role» Ordained Bishop
«Role» Ordinator of Bishop
«Role» Ordinator of Priest
«Role» Cardinal Nominator
«Relator» Bishop Ordination
In a given Bishop Ordination "x", the Ordained Bishop may not also be one of the Ordinators of Bishop.

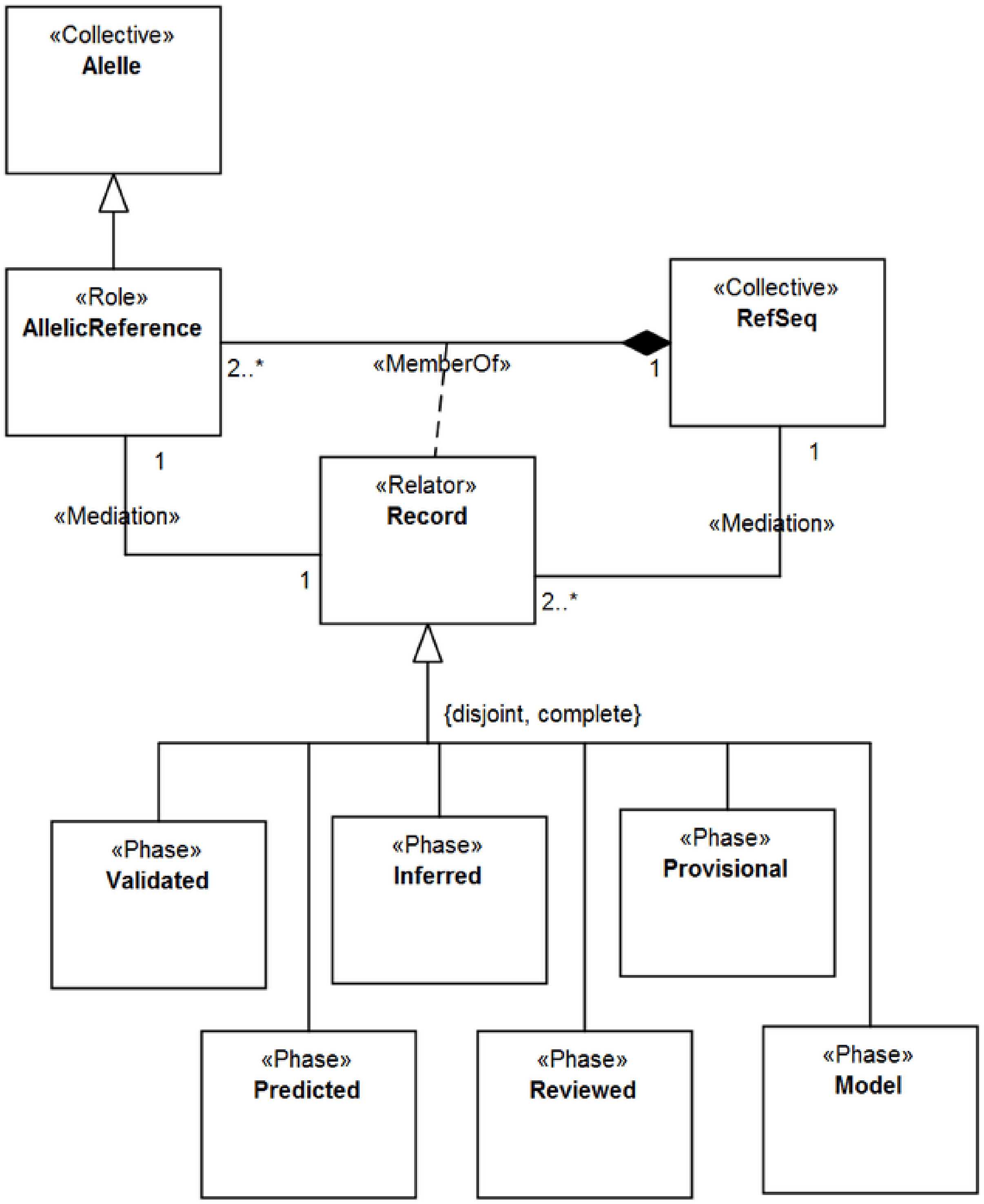
EX2: Fragment of a conceptual model representing the worldview of a possible parking lot management system (see more):



EX3: UFO-S fragment focused on service offering (see more):



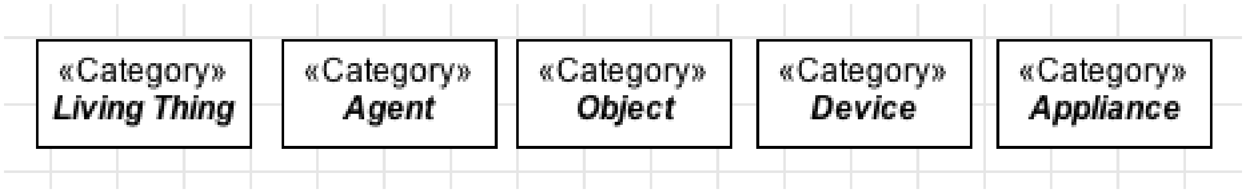
EX4: Fragment of a conceptual model about the human genome (see more):


«Collective»
Alelle
«Role»
AllelicReference
2..*
«MemberOf»
1
«Collective»
RefSeq
1
«Mediation»
«Relator»
Record
1
1
«Mediation»
2..*
{disjoint, complete}
«Phase»
Validated
«Phase»
Inferred
«Phase»
Provisional
«Phase»
Predicted
«Phase»
Reviewed
«Phase»
Model

3.8 Category

3.8.1 Definition

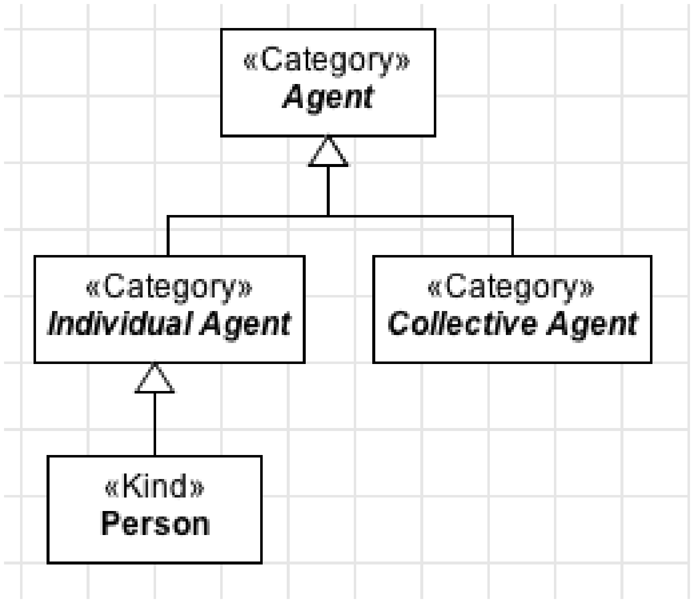
A «*Category*» is a *rigid* mixin that does not require a dependency to be specified. It is used to aggregate essential properties to individuals which following *different identity principles*. Let's see some examples:



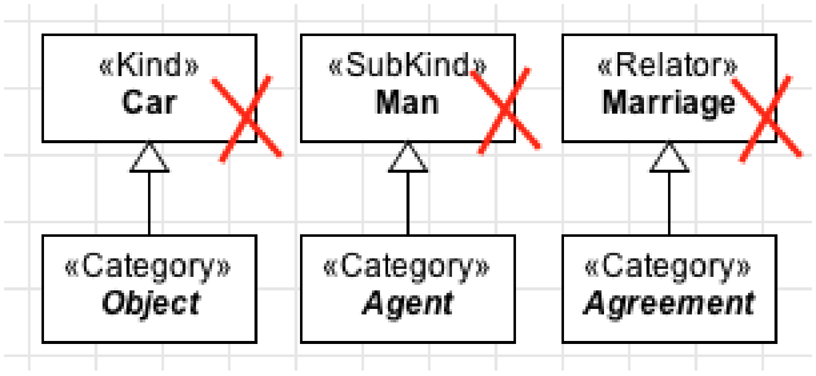
Categories are usually used in a refactoring process. For example, let's suppose that you defined two classes in your model, Person and Animal. Now you want to state that either people and animals have a weight. You than create a «*Category*», which has weight, and generalize the existing classes into it.

3.8.2 Constraints

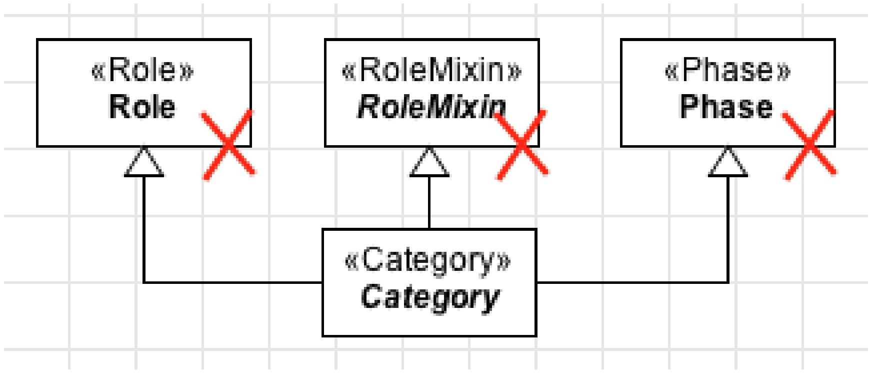
C1: A «*Category*» is always abstract. Notice that abstract classes are represented with an *italic* label.



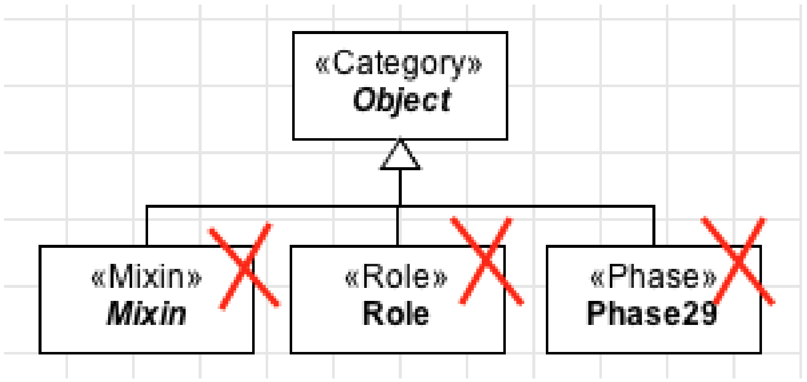
C2: A «*Category*» aggregate individuals that follow *different identity principles*, therefore it may not have as ancestor the following constructs: «*Kind*», «*Quantity*», «*Collective*», «*Subkind*», «*Role*», «*Phase*», «*Relator*», «*Mode*», «*Quality*».



C3: A «*Category*» is a *rigid* construct, therefore it cannot have as ancestor an *anti-rigid* type, as: «*Role*», «*RoleMixin*», «*Phase*».



C4: Categories cannot have as descendants the following types: «*Mixin*», «*Role*», «*Phase*».

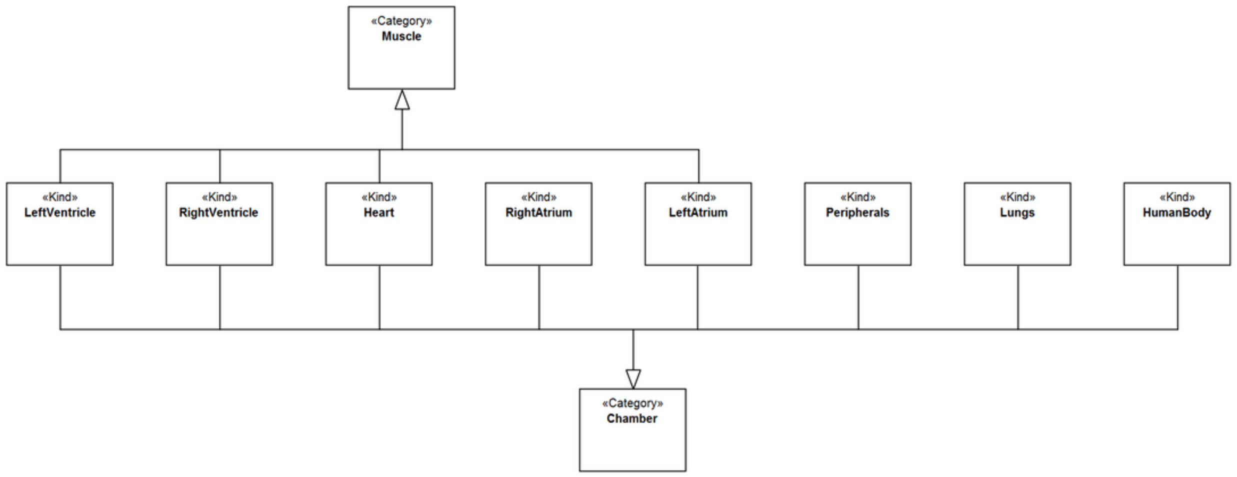


3.8.3 Common questions

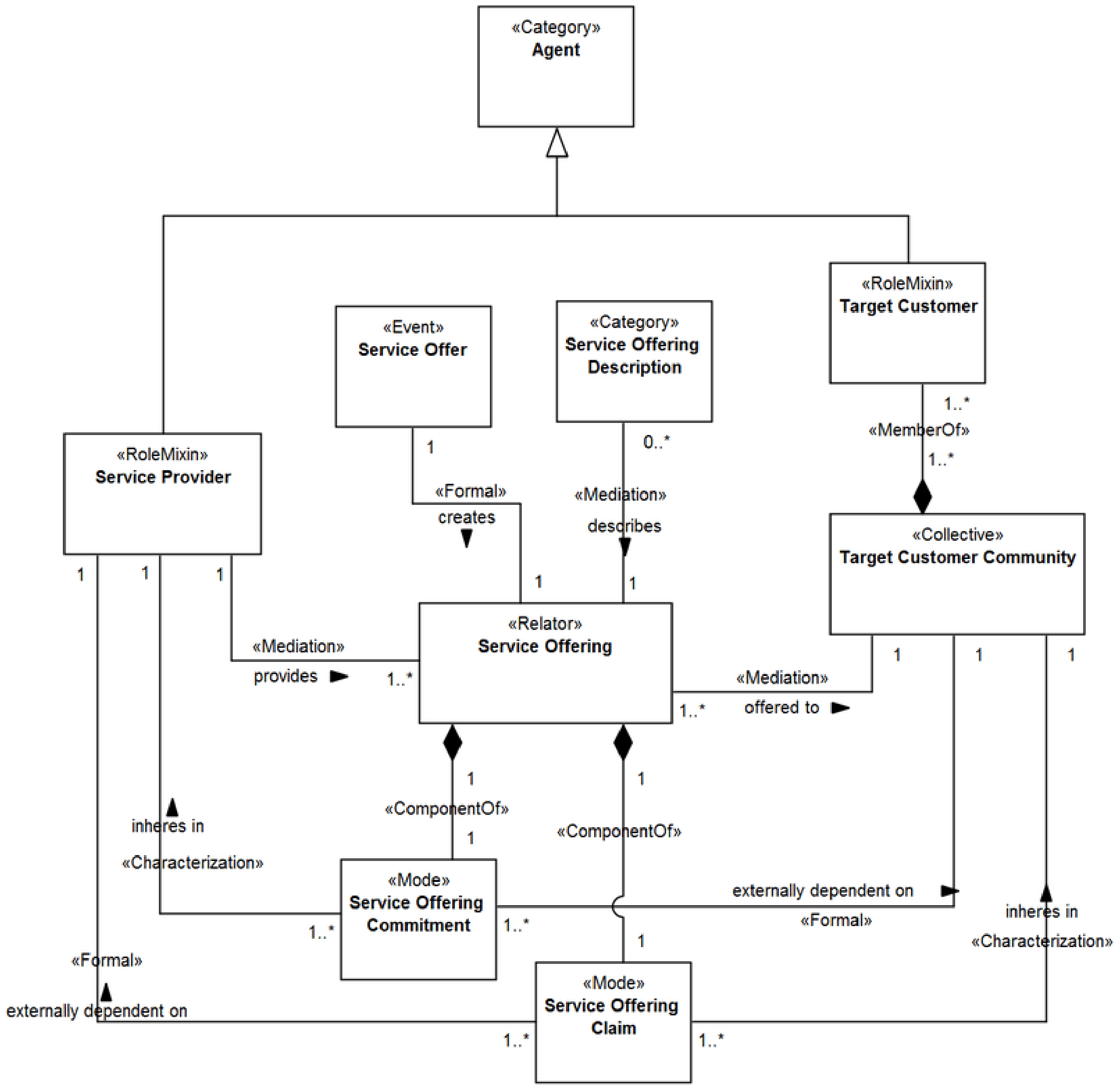
Ask us some question if something is not clear . . .

3.8.4 Examples

EX1: Fragment from the ECG Ontology (see more):



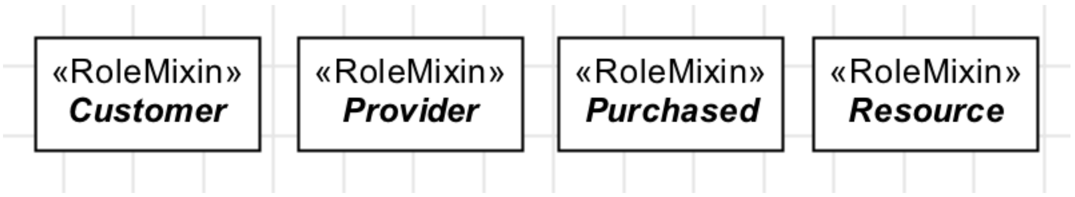
EX2: Fragment from UFO-S, a commitment-based service ontology (see more):


«Category» Agent
«RoleMixin» Service Provider
«Event» Service Offer
«Category» Service Offering Description
«RoleMixin» Target Customer
«Collective» Target Customer Community
«Relator» Service Offering
«Mode» Service Offering Commitment
«Mode» Service Offering Claim
«Formal» creates
«Mediation» describes
«MemberOf»
«Mediation» provides
«Mediation» offered to
«ComponentOf»
«ComponentOf»
inheres in «Characterization»
externally dependent on «Formal»
«Formal» externally dependent on
inheres in «Characterization»
1
0..*
1..*

3.9 RoleMixin

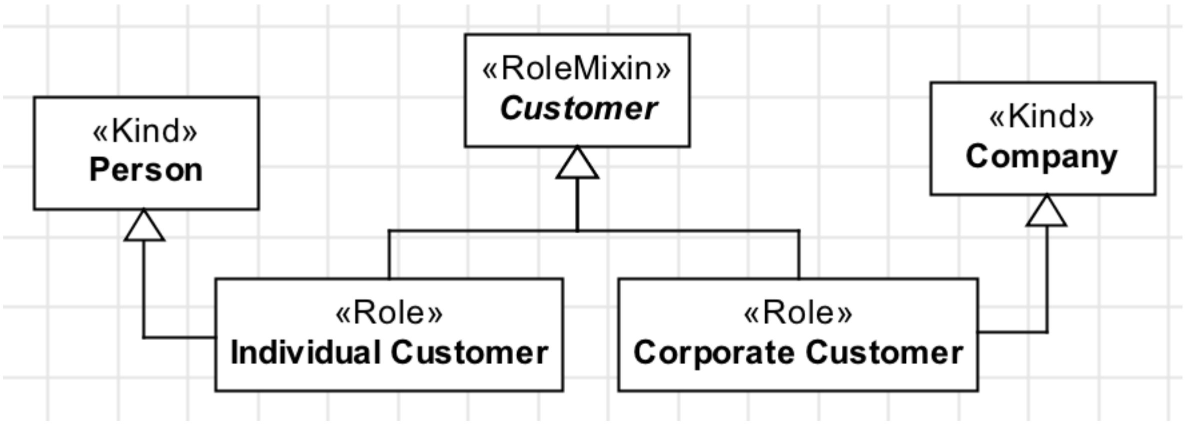
3.9.1 Definition

A «*RoleMixin*» is the equivalent of «*Role*» for types that aggregate instances with *different identity principles*. A class stereotyped as «*RoleMixin*» is also an *anti-rigid* type whose instantiation *depends on a relational property*. Here are some examples:

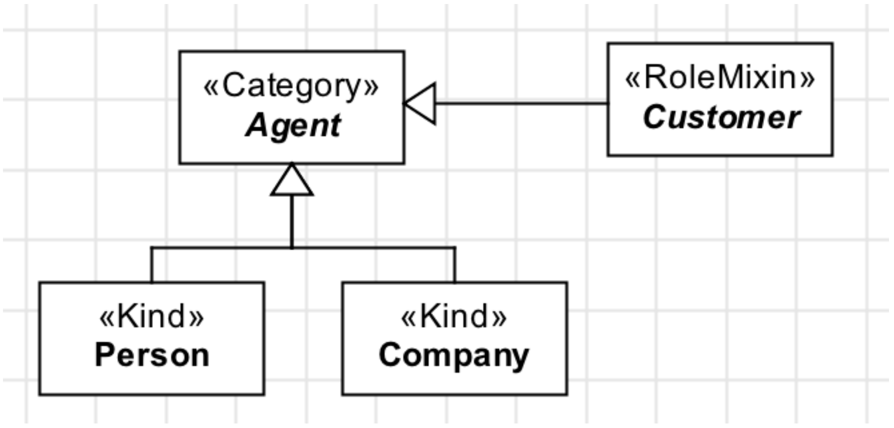


RoleMixins usually occur in one of the two patterns:

- **Pattern 1:** «*RoleMixin*» defined by roles



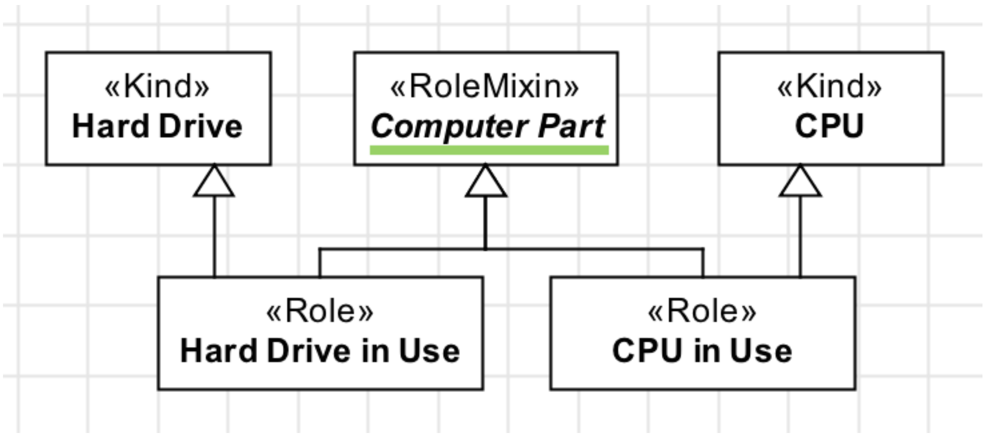
- **Pattern 2:** «*RoleMixin*» as a role of a «*Category*»



The second pattern is a more concise form of the first. They are semantically equivalent.

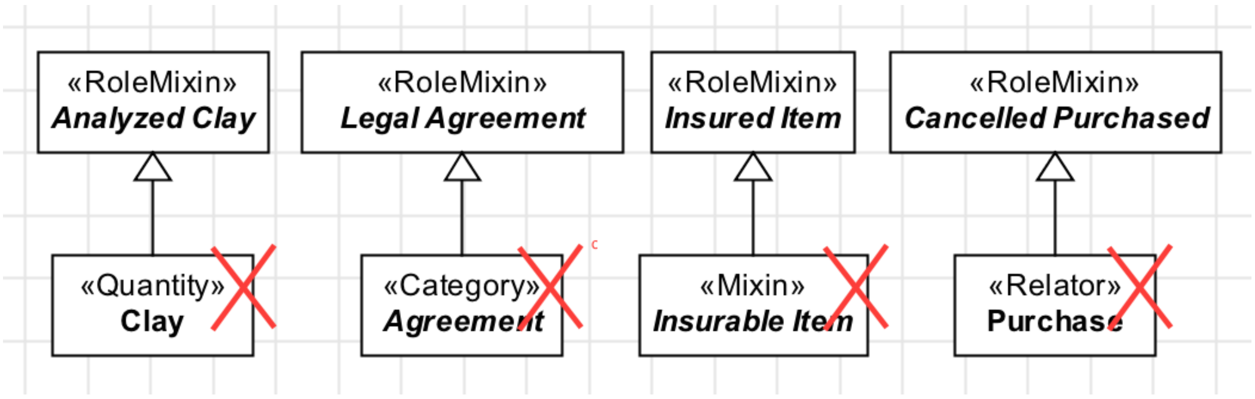
3.9.2 Constraints

C1: A «*RoleMixin*» is always abstract. Notice that abstract classes are represented with an *italic* label.



C2: A «*RoleMixin*» aggregate individuals that follow *different identity principles*, therefore it may not have as ancestor the following constructs: «*Kind*», «*Quantity*», «*Collective*», «*Subkind*», «*Role*», «*Phase*», «*Relator*», «*Mode*», «*Quality*».

C3: A «*RoleMixin*» is a *anti-rigid* construct, therefore it cannot have as descendent any *rigid* or *semi-rigid* type, as: «*Kind*», «*Quantity*», «*Collective*», «*Subkind*», «*Category*», «*Mixin*», «*Relator*», «*Mode*», «*Quality*».

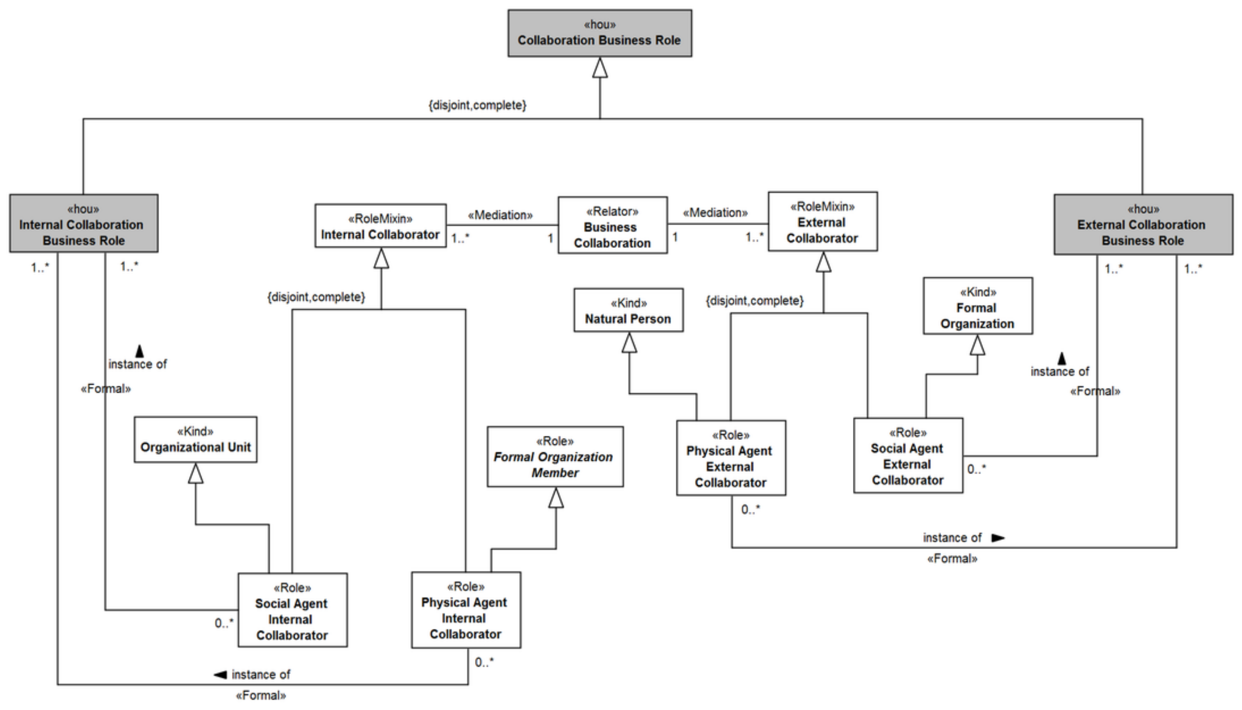


3.9.3 Common questions

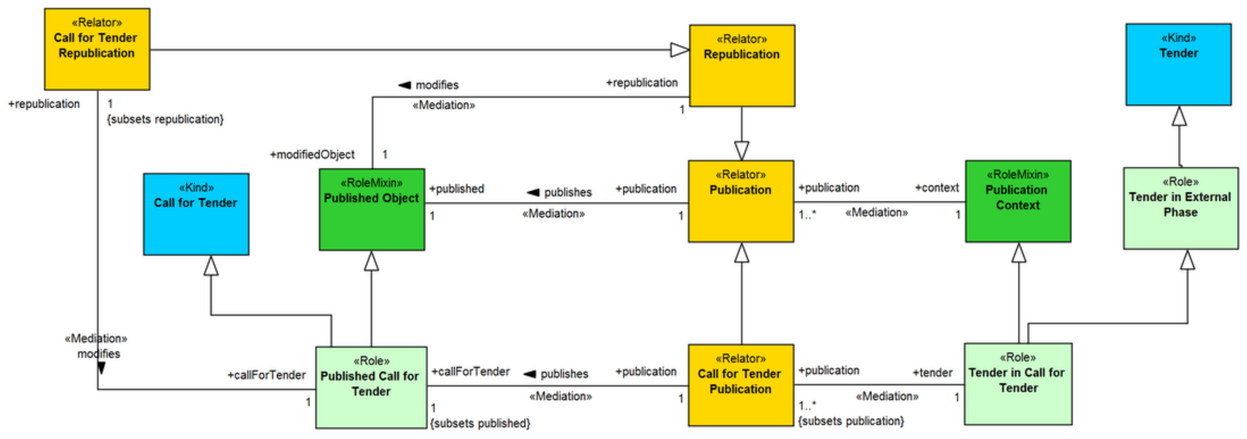
Ask us some question if something is not clear . . .

3.9.4 Examples

EX1: Fragment of the OntoUML Org Ontology (O3) (see more):


«hou» Collaboration Business Role
{disjoint,complete}
«hou» Internal Collaboration Business Role
«RoleMixin» Internal Collaborator
«Mediation»
«Relator» Business Collaboration
«RoleMixin» External Collaborator
«hou» External Collaboration Business Role
«Kind» Natural Person
«Kind» Formal Organization
«Kind» Organizational Unit
«Role» Formal Organization Member
«Role» Physical Agent External Collaborator
«Role» Social Agent External Collaborator
«Role» Social Agent Internal Collaborator
«Role» Physical Agent Internal Collaborator
instance of
«Formal»
1..*
0..*
1

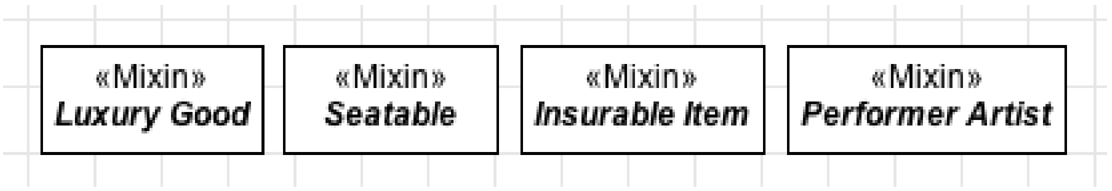
EX2: Fragment of a conceptual model about Brazilian Public Tenders (see more):



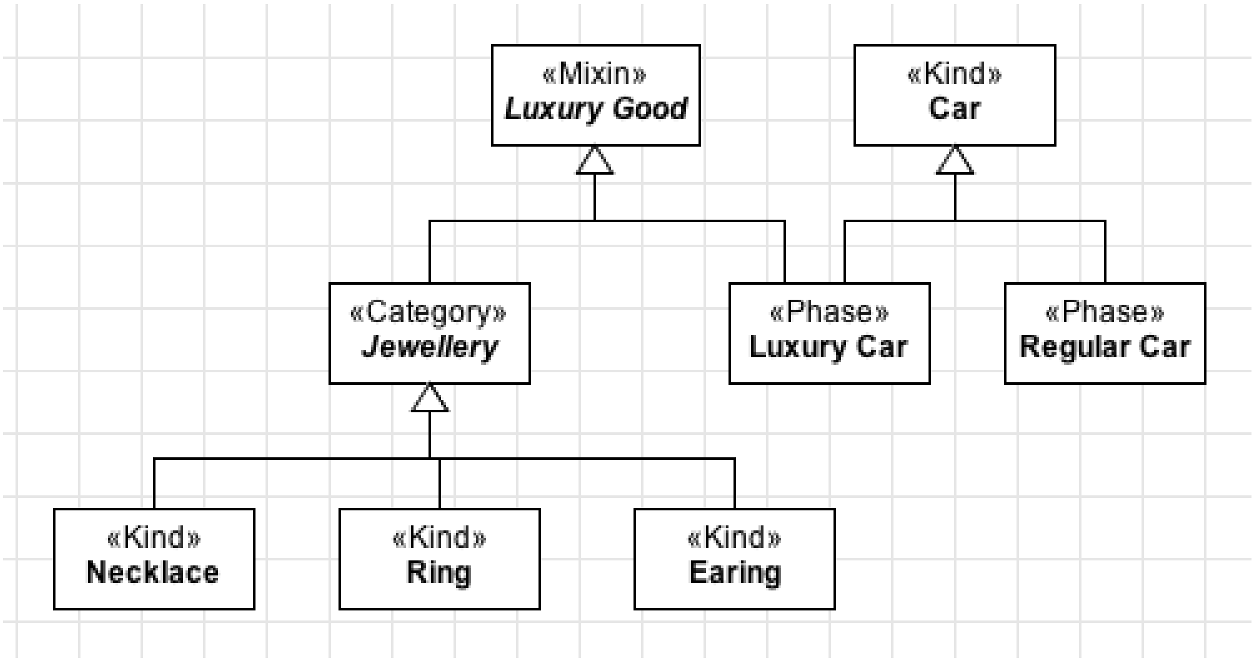
3.10 Mixin

3.10.1 Definition

A «*Mixin*» is a semi-rigid type, i.e., it "behaves" as a *rigid* type for some individuals and as an *anti-rigid* one for others (it's the only stereotype with such feature in OntoUML). As the «*Category*» and the «*RoleMixin*», the «*Mixin*» meta-class characterizes individuals that follow *different identity principles*. Here are some examples of types that could be classified as «*Mixin*»:

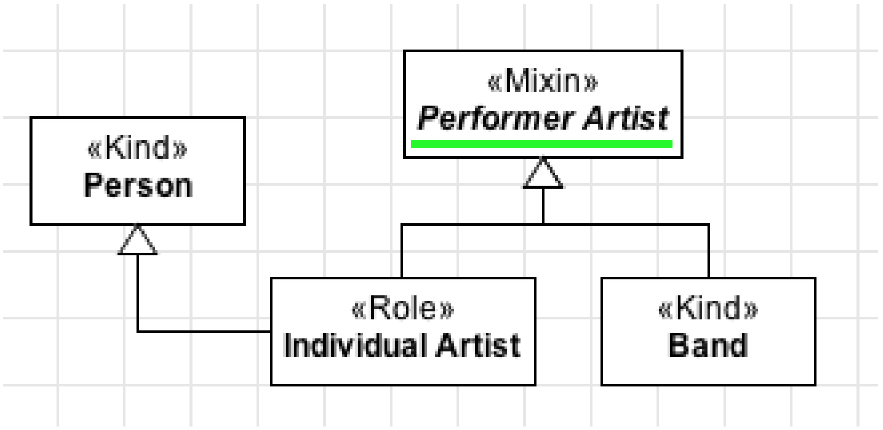


As *categories*, mixins are commonly applied during a refactoring process, in particular when we want to state that some properties are applied to both *rigid* and *anti-rigid* types. For instance, let's consider that we defined the following types in our model, Car and Jewellery, a general concept for Ring, Necklace, etc. Now we want to define the type Luxury Good. In our worldview, every jewellery is luxurious, but only cars that are worth more than 30k dollars are. Since the value of a car changes through the years, being a luxurious car is a temporary classification, whilst being a jewellery is a permanent one. The type Luxury Good, therefore, is *semi-rigid* or a «*Mixin*».

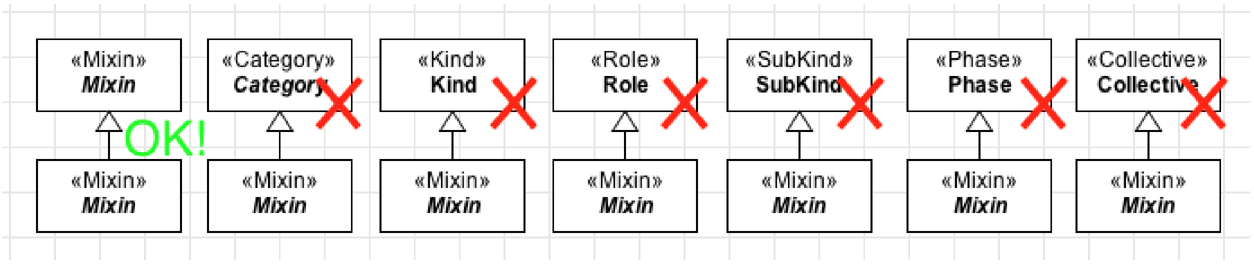


3.10.2 Constraints

C1: A «*Mixin*» is always abstract. Note that abstract classes are represented with *italic* labels.



C2: A «*Mixin*» is a *semi-rigid* construct and because of that, it cannot have as ancestor either a *rigid* or an *anti-rigid* type. Therefore, only mixins can be ancestor of other mixins.

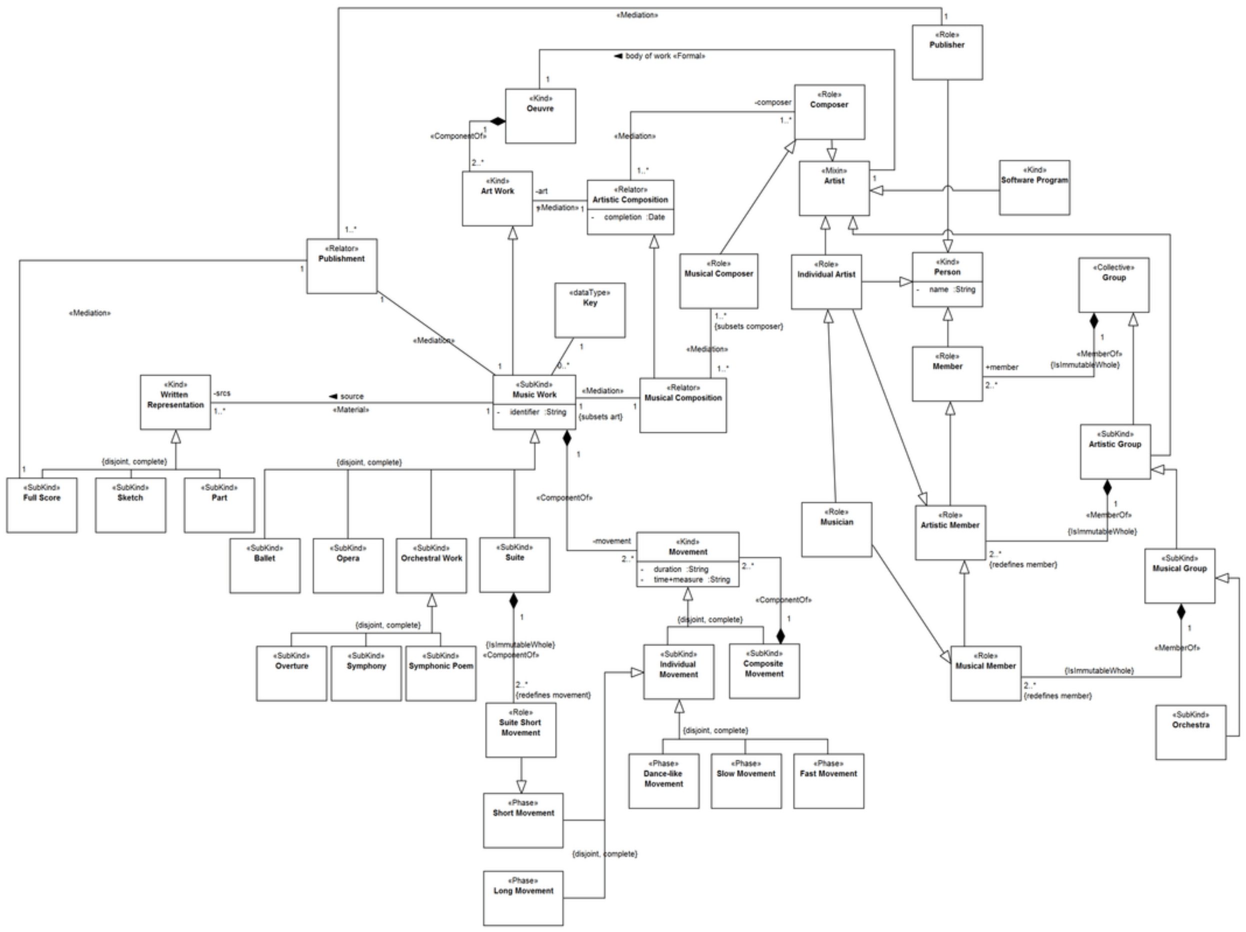


3.10.3 Common questions

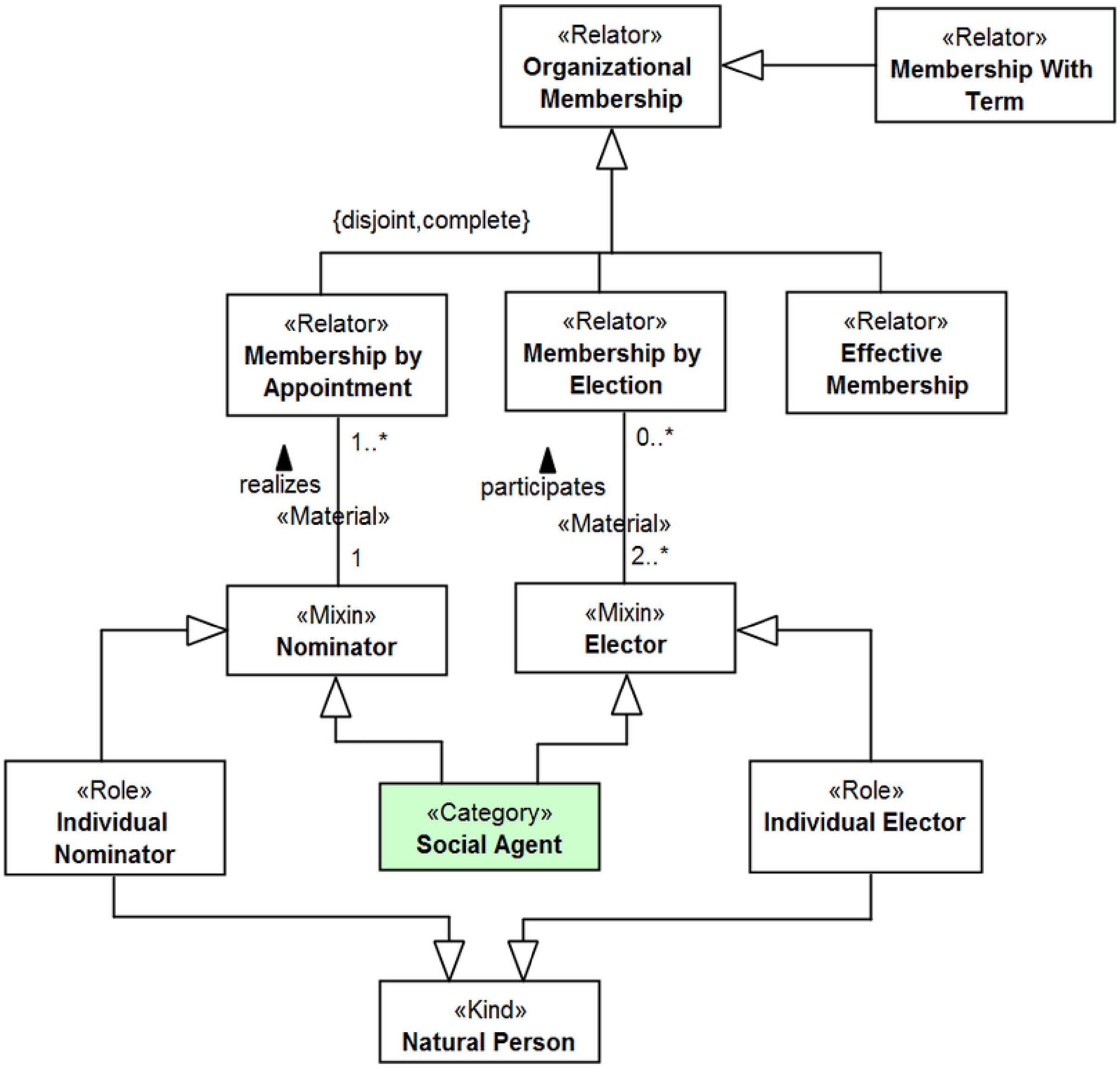
Ask us some question if something is not clear . . .

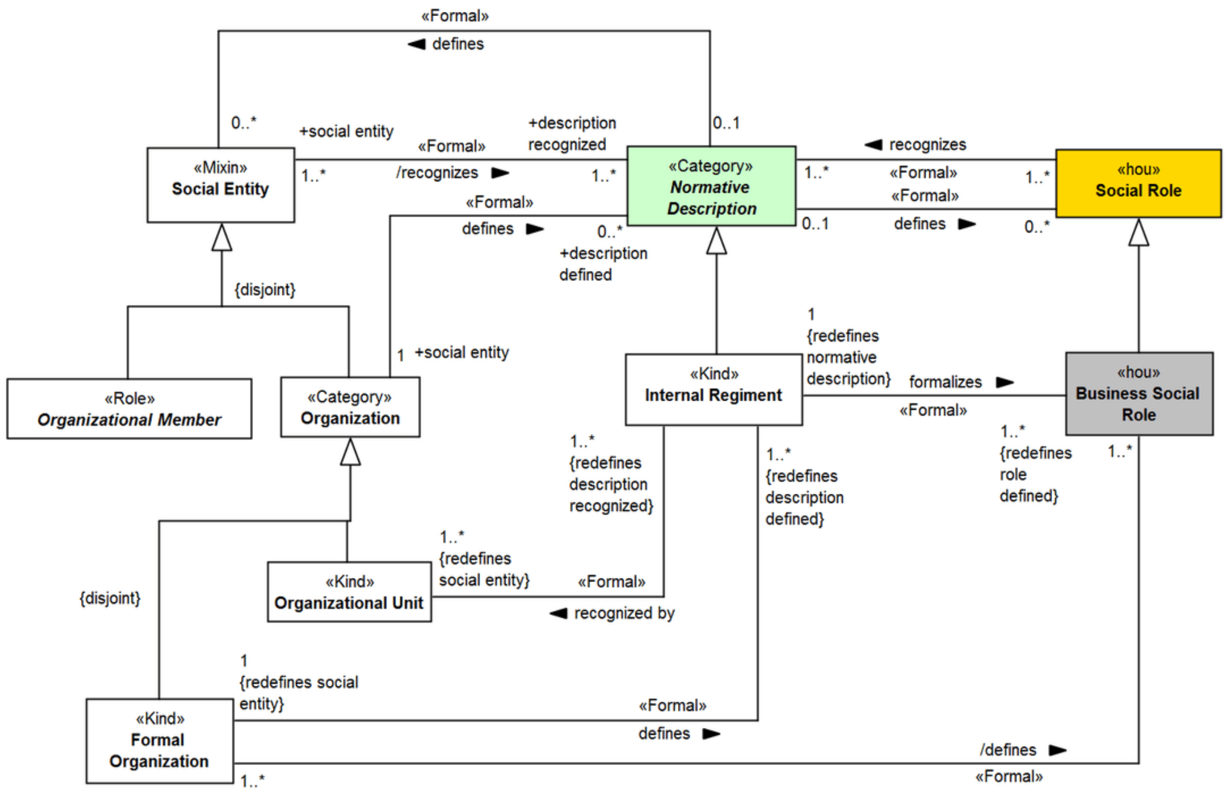
3.10.4 Examples

EX1: Conceptual model based on the Music Ontology (see more):



EX2: Fragments extracted from the OntoUML Org Ontology (O3), a model about the active structure of organisations (see more):

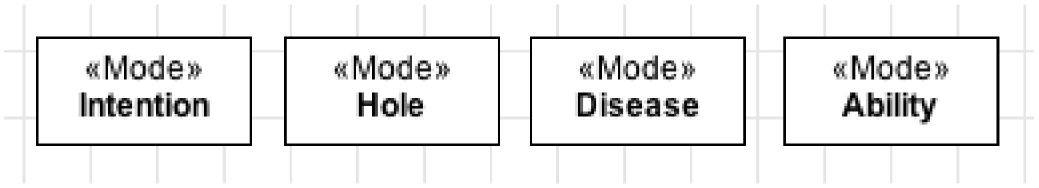

«Relator»
Organizational Membership
«Relator»
Membership With Term
{disjoint,complete}
«Relator»
Membership by Appointment
«Relator»
Membership by Election
«Relator»
Effective Membership
1..*
realizes
«Material»
1
0..*
participates
«Material»
2..*
«Mixin»
Nominator
«Mixin»
Elector
«Role»
Individual Nominator
«Category»
Social Agent
«Role»
Individual Elector
«Kind»
Natural Person



3.11 Mode

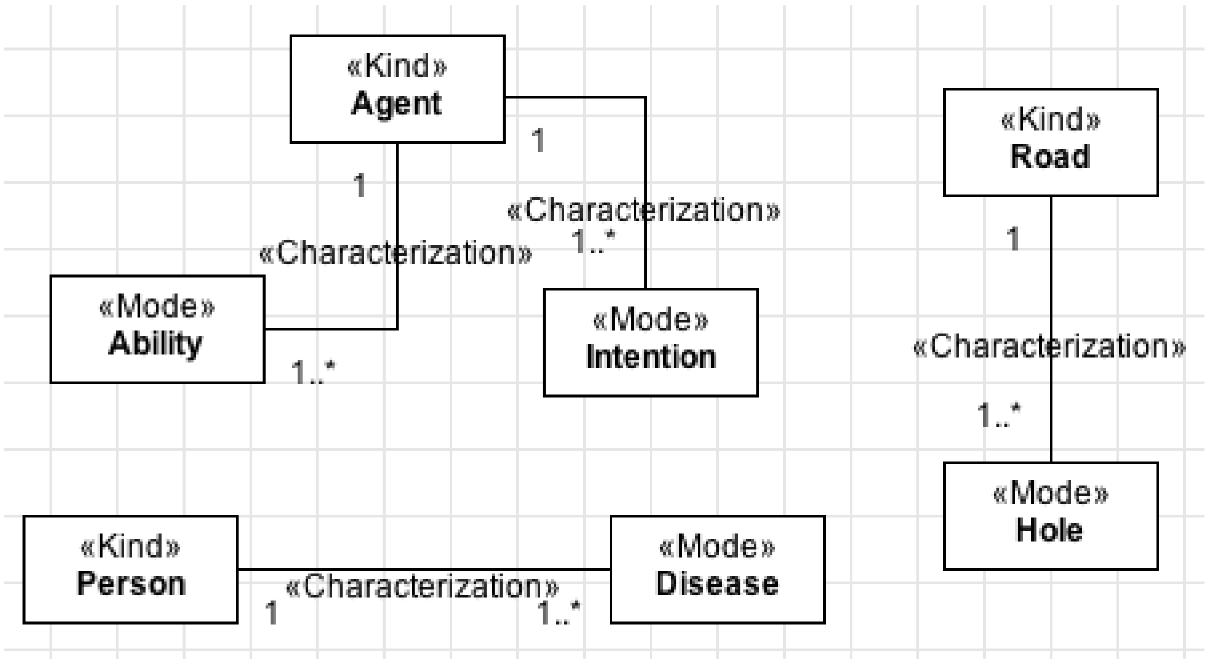
3.11.1 Definition

A «*Mode*» is a particular type of intrinsic property that has no structured value. Like *qualities*, modes are also individuals that *existentially depend* on their **bearers**. Types stereotyped as «*Mode*» are also *rigid*. You can find some examples of modes below:

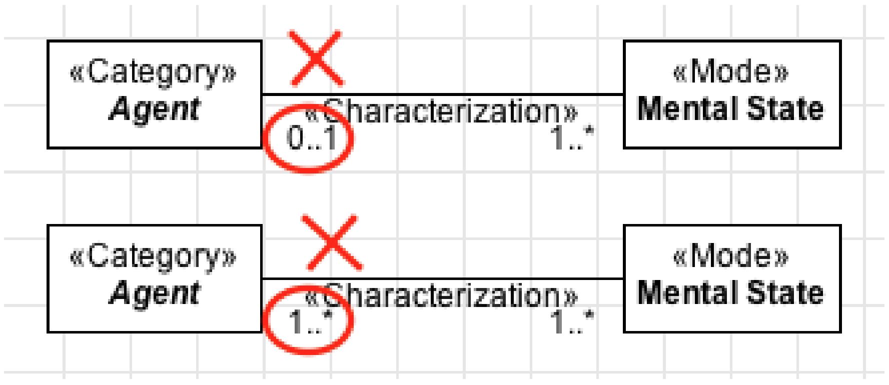


3.11.2 Constraints

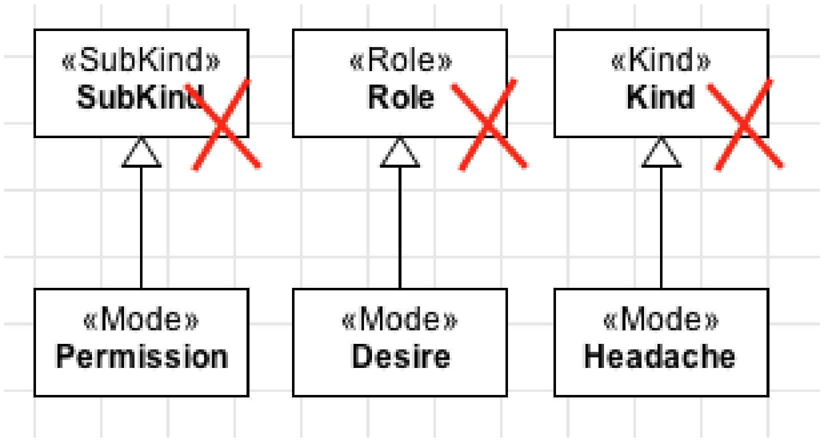
C1: Every «*Mode*» must be (directly or indirectly) connected to an association end of at least one «*Characterization*» relation.



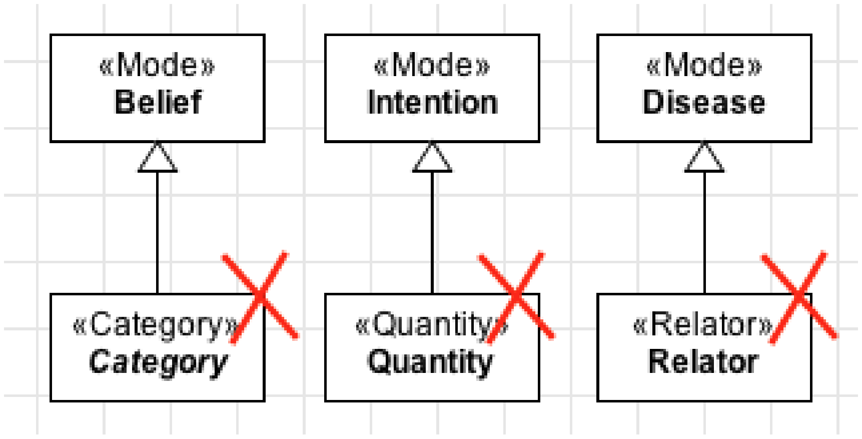
C2: The multiplicity of the characterized end (opposite to the «*Mode*») must be exactly one. Therefore, the following examples are forbidden.



C3: Modes cannot have as ancestors the following types: «*Kind*», «*Quantity*», «*Collective*», «*Subkind*», «*Role*», «*RoleMixin*», «*Phase*», «*Relator*», «*Quality*».



C4: Modes cannot have as descendants the following types: «*Kind*», «*Quantity*», «*Collective*», «*RoleMixin*», «*Category*», «*Mixin*», «*Relator*», «*Quality*».

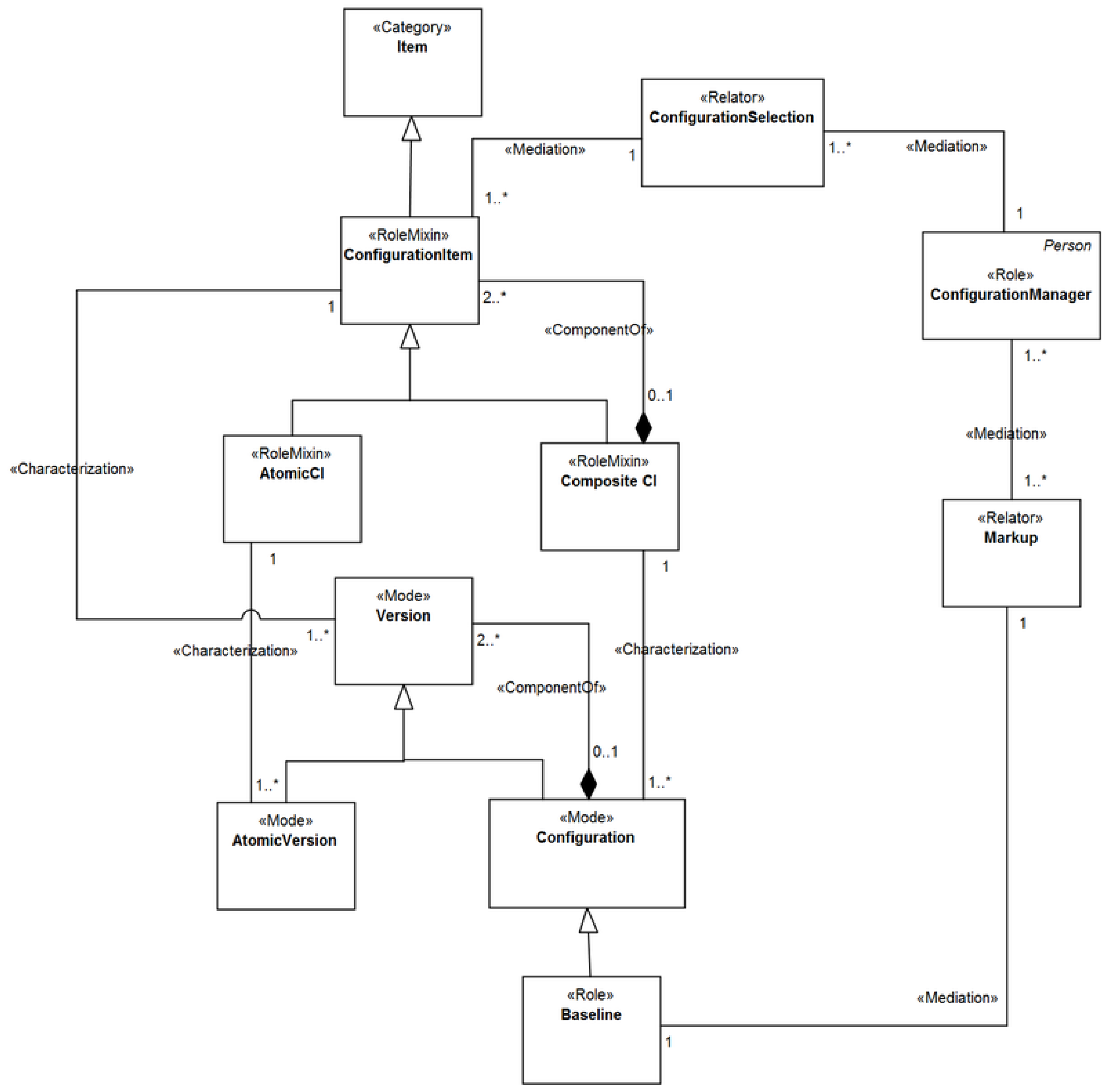


3.11.3 Common questions

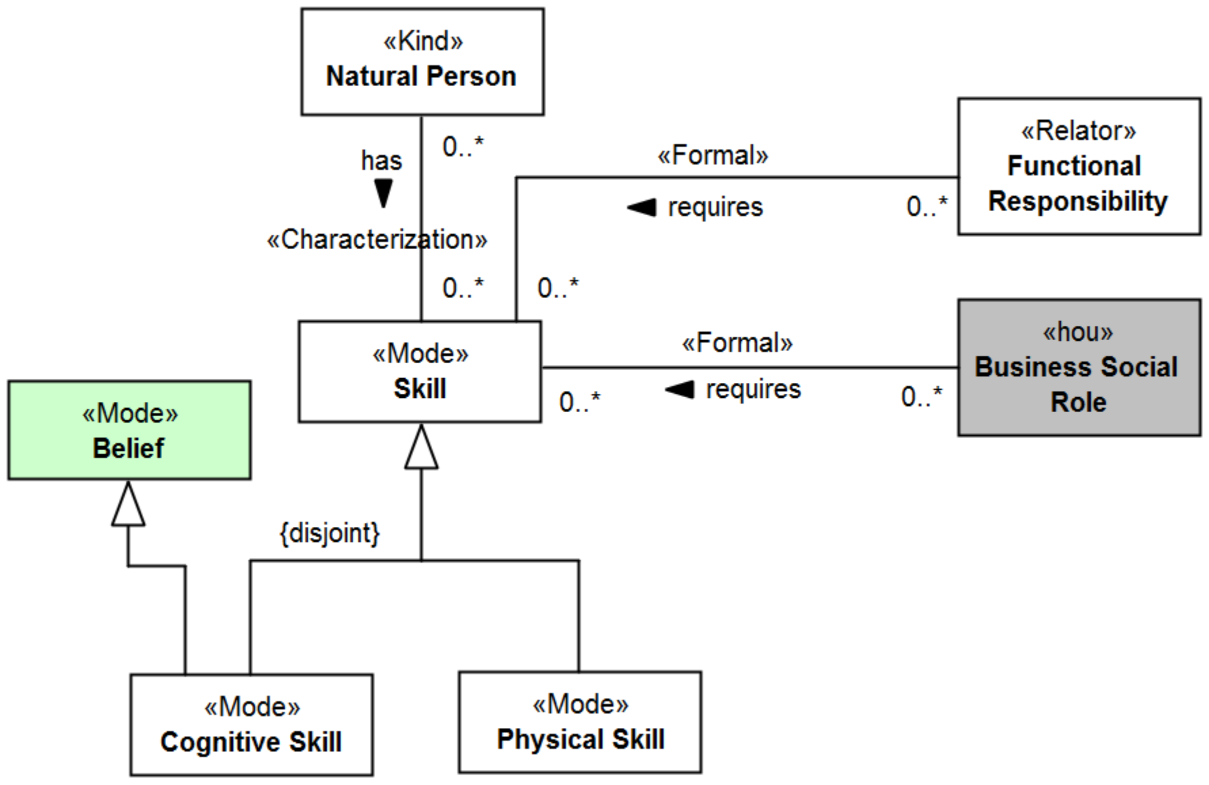
Ask us some question if something is not clear . . .

3.11.4 Examples

EX1: Fragment from the Configuration Management Task Ontology (see more):



EX2: Fragment from the OntoUML Org Ontology (O3) (see more):



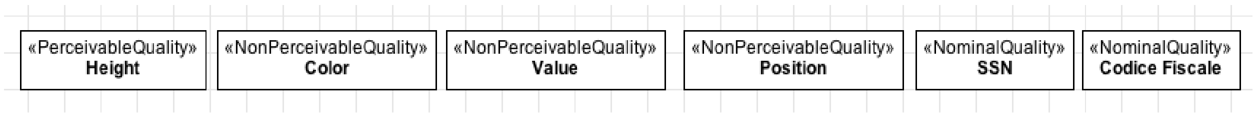
3.12 Quality

3.12.1 Definition

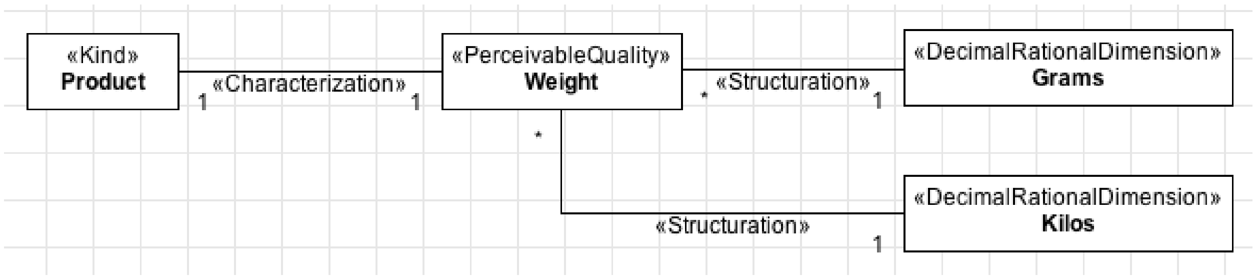
A «*Quality*» is a particular type of intrinsic property which has a structured value. Qualities are things that are *existentially dependent* on the things they characterize, called their **bearers**. Types stereotyped as «*Quality*» are also *rigid*. OntoUML differentiates between three types of qualities:

- **Perceivable**, which capture qualities that could be measured by an agent with the appropriate instrument, like weight, height, color and speed.
- **Non-Perceivable**, which represent properties which cannot be directly measured by an instrument, like currency.
- **Nominal**, which are used to make reference to an individual, like one's name, a book's ISBN or a credit card number.

Notice some examples of qualities in the next figure:

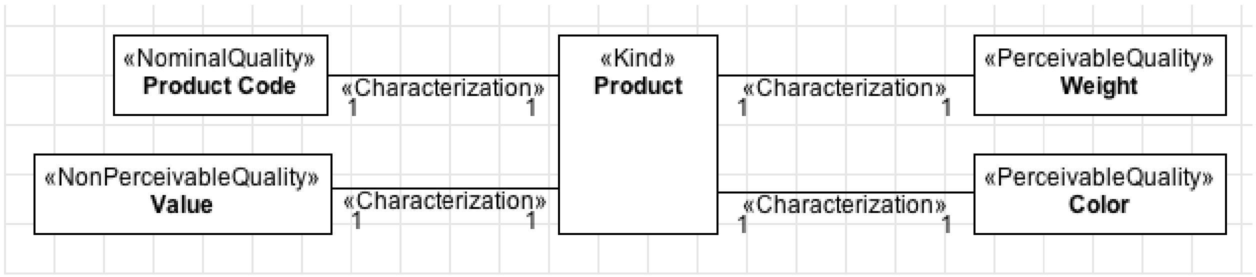


You can define different types of geometrical structures for a quality value using dimensions and domains. Here is an example:

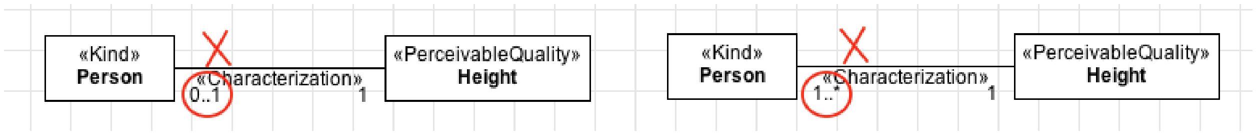


3.12.2 Constraints

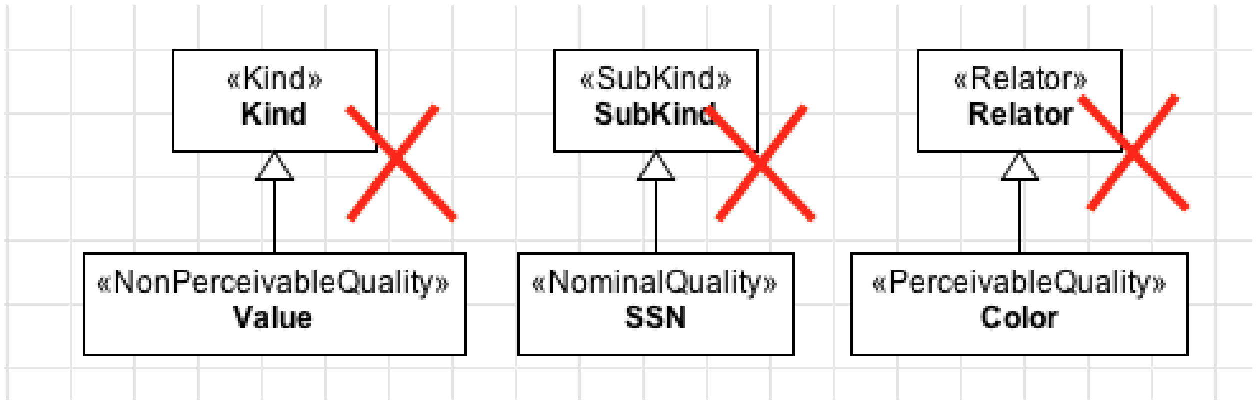
C1: A «*Quality*» must always be connected, through a «*Characterization*» to another type.



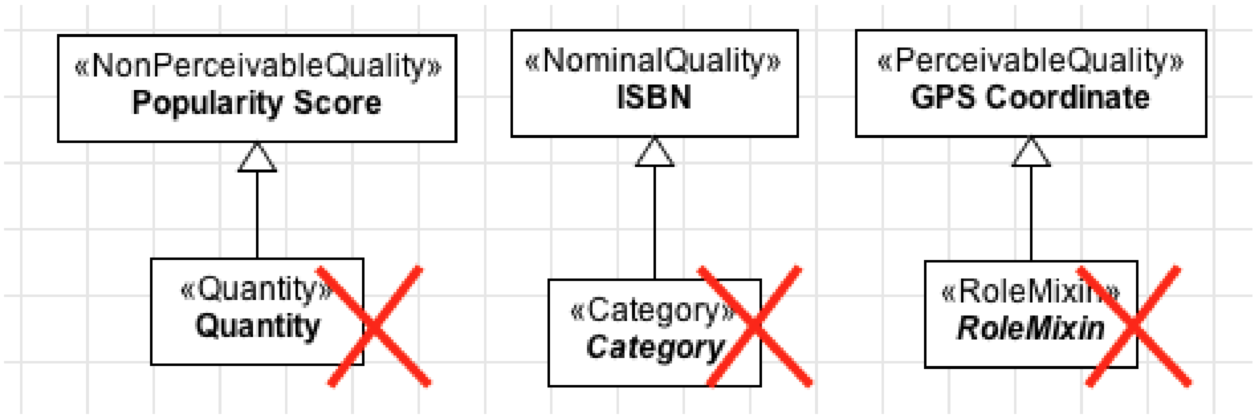
C2: The multiplicity of the characterized end (opposite to the quality) must be exactly one. Therefore, the following examples are forbidden.



C3: Qualtities cannot have as ancestors the following types: «*Kind*», «*Quantity*», «*Collective*», «*Subkind*», «*Role*», «*RoleMixin*», «*Phase*», «*Relator*», «*Mode*».



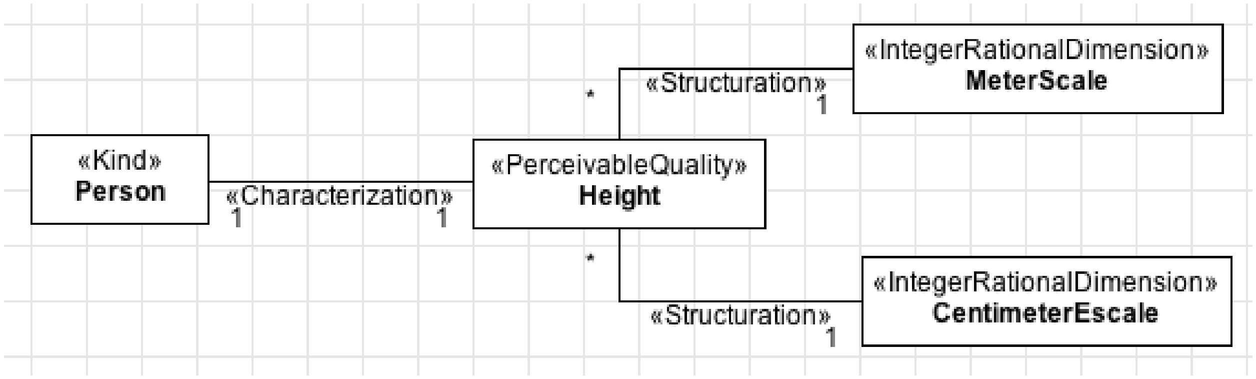
C4: Qualtities cannot have as descendants the following types: «*Kind*», «*Quantity*», «*Collective*», «*RoleMixin*», «*Category*», «*Mixin*», «*Relator*», «*Mode*».



3.12.3 Common questions

Q1: Can I represent the property “height” as an attribute instead of a «*Quality*»?

A1: Yes. The decision to represent attributes or qualities is entirely up to you. It is useful to represent properties as qualities when you want to define different escales for the same characteristic. For instance, if you want to model that a Person has a “height” property, which can be measured in meters or centimeters you should explicitly represent the Height quality.



3.12.4 Examples

Todo: To be added . . .

CHAPTER 4

Relationship stereotypes

4.1 Introduction

Relations are entities that glue together other entities. Every relation has a number of **relata** as arguments, which are connected or related by it. The number of a relation's arguments is called its arity. As much as an unary property such as *being Red,* properties of higher arities such as *being married-to, being heavier-than* are universals, since they can be predicated of a multitude of individuals. Relations can be classified according to the types of their relata. There are relations between sets, between individuals, and between universals, but there are also cross-categorical relations, for example, between urelements and sets or between sets and universals. We divide relations into two broad categories, called *Material* and *Formal* relations. *Formal relations* hold between two or more entities directly without any further intervening individual. Examples of formal relations are:

- 5 is *greater than* 3
- this day *is part of* this month
- N is *subset of* Q

but also the relations of instantiation, inherence, quale of a quality, association, existential dependence, among others – . . . relations that form the mathematical superstructure of our framework. *Material relations,* conversely, have material structure on their own and include examples such as:

- employments
- kisses
- enrollments
- flight connections
- commitments

The relata of a material relation are mediated by individuals that are called *relators*. Relators are individuals with the power of connecting entities:

- *a flight connection,* for example, founds a relator that connects airports
- *an enrollment* is a relator that connects a student with an educational institution

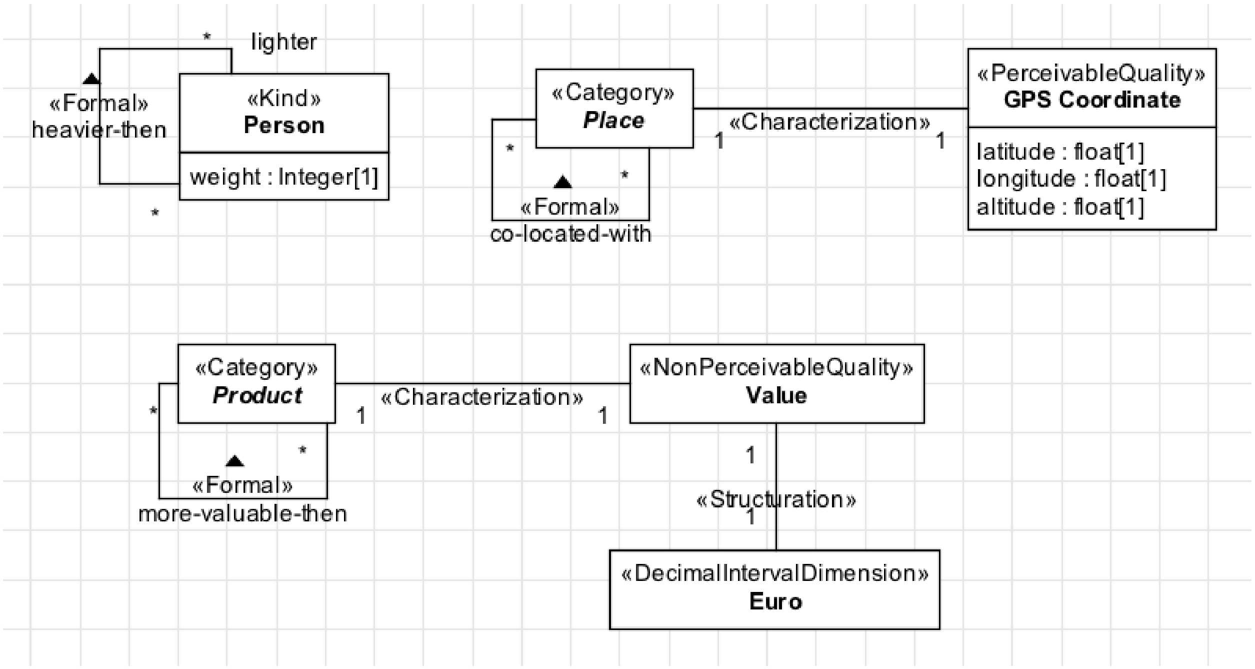
Quoted from:

GUIZZARDI, Giancarlo. *Ontological Foundations for Structural Conceptual Models.* Enschede: CTIT, Telematica Instituut, 2005.

4.2 Formal

4.2.1 Definition

The name «*Formal*» is short for *Domain Comparative Formal Relation*. This construct is used to represent relations that can be reduced to the comparison of the quality values that characterize the related individuals, like heavier-then, younger-then or cheaper-then. Here are some examples in OntoUML:



To specify how the relation can be reduced, use an OCL derivation rule:

```
context Person::lighter : Set(Person)
derive : Person.allInstances()->select(x | self.weight > x.weight)
```

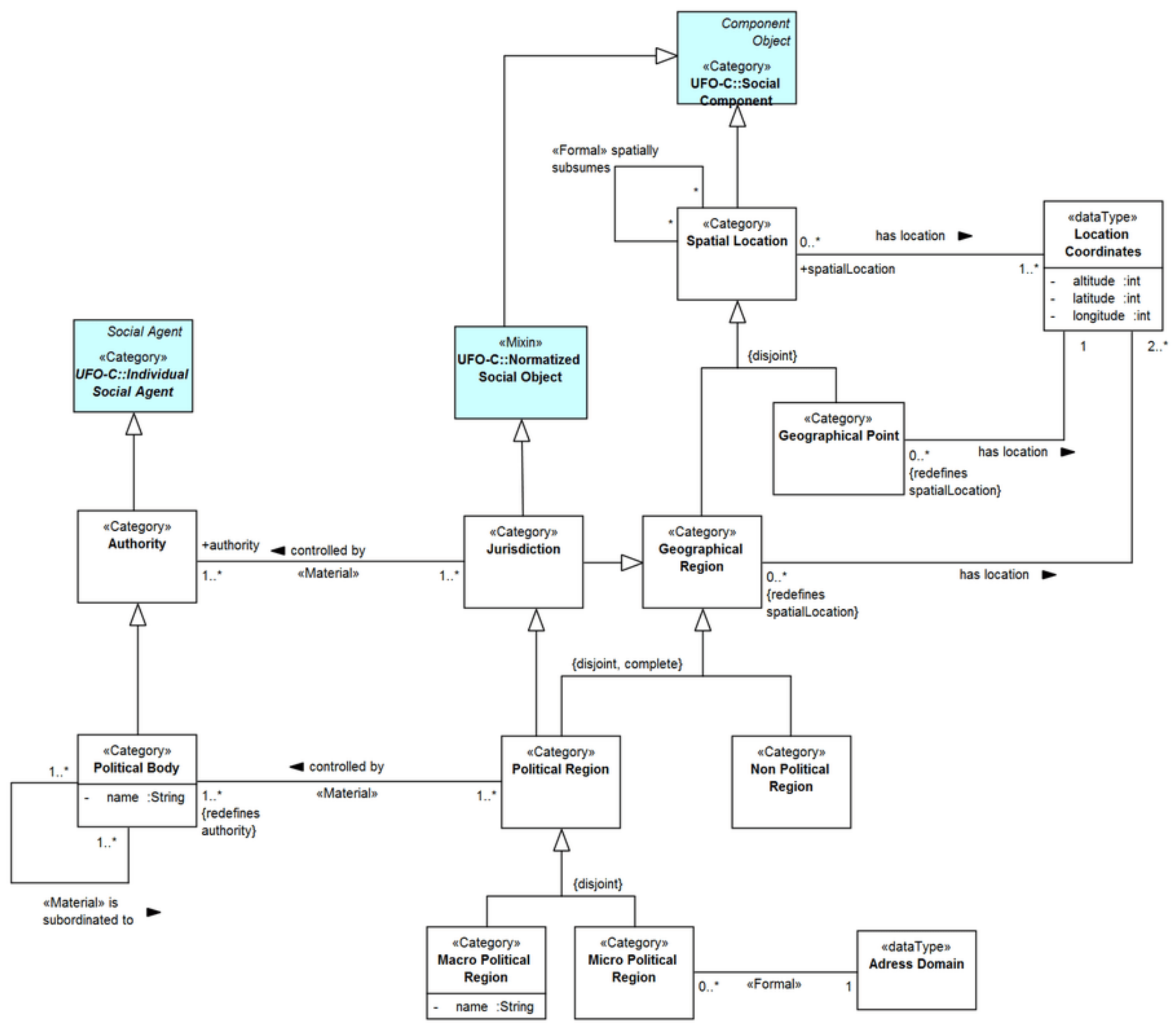
Tip: Due to its ontological, the «*Formal*» relations have no constraints in OntoUML. Nonetheless, make sure the relation you are modeling is indeed a comparative one. Think about how to reduce the relation to a comparison between values and represent the necessary properties.

4.2.2 Common questions

Ask us some question if something is not clear . . .

4.2.3 Examples

EX1: Fragment from OntoEmerge, an ontology about Emergency Plans (see more):



4.3 Material

4.3.1 Definition

«*Material*» relations have material structure on their own and include examples such as employments, kisses, enrollments, flight, connections and commitments. The **relata** of a material relation are mediated by individuals that are called relators. Relators («*Relator*») are individuals with the power of connecting entities; a flight connection, for example, founds a relator that connects airports, an enrollment is a relator that connects a student with an educational institution. Relators play an important role in answering questions of the sort: what does it mean to say that John is married to Mary? Why is it true to say that Bill works for Company X but not for Company Y?.

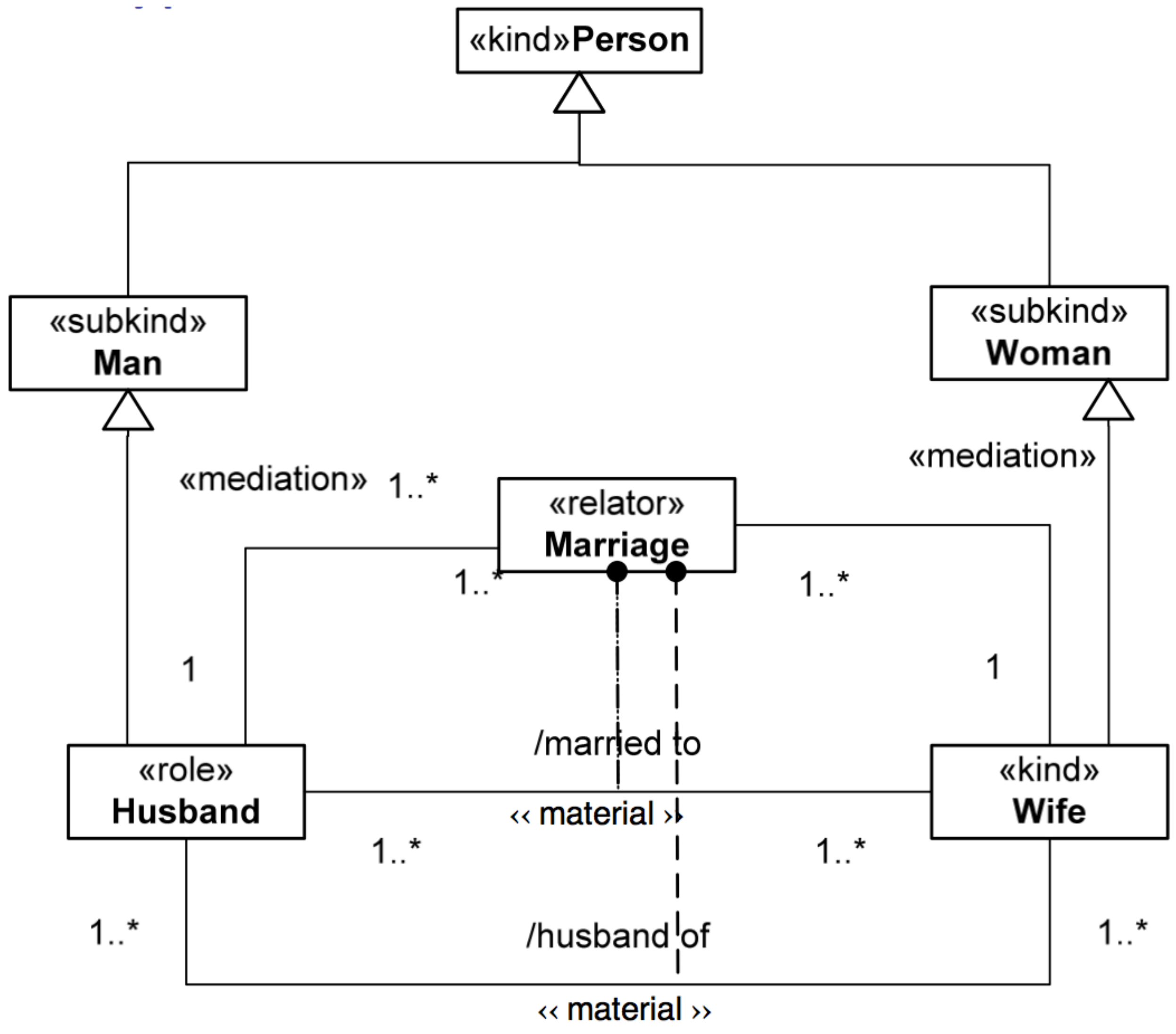
Material relations are derived (via «*Derivation*») from relators and the mediation relations that connect them to the corresponding relata. Cardinality constraints of mediation relations collapse by derivation. Material relations are always affected by collapsed cardinality). Also, several «*Material*» relations can be derived from a single «*Relator*» and «*Mediation*» relations.

4.3.2 Common questions

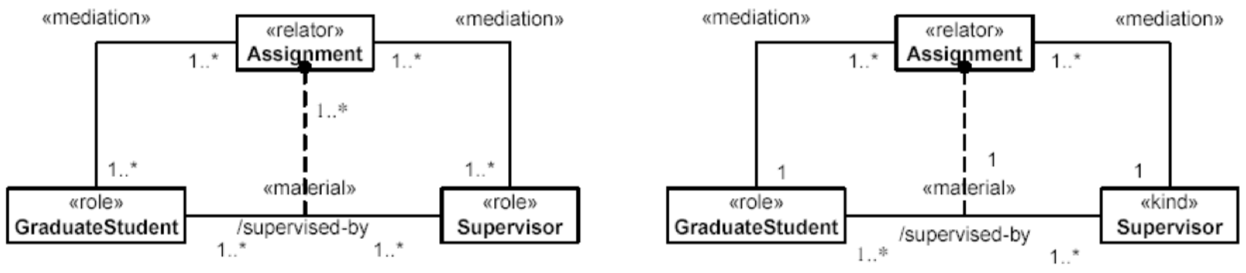
Ask us some question if something is not clear …

4.3.3 Examples

EX1:



EX2:



For more examples see «*Relator*», «*Derivation*», «*Mediation*», and «*Relator pattern*».

Quoted from:

GUIZZARDI, Giancarlo. *Ontological Foundations for Structural Conceptual Models.* Enschede: CTIT, Telematica Instituut, 2005. GUIZZARDI, Giancarlo. *Introduction to Ontological Engineering.* [presentation] Prague: Prague University of Economics, 2011.

4.4 Mediation

4.4.1 Definition

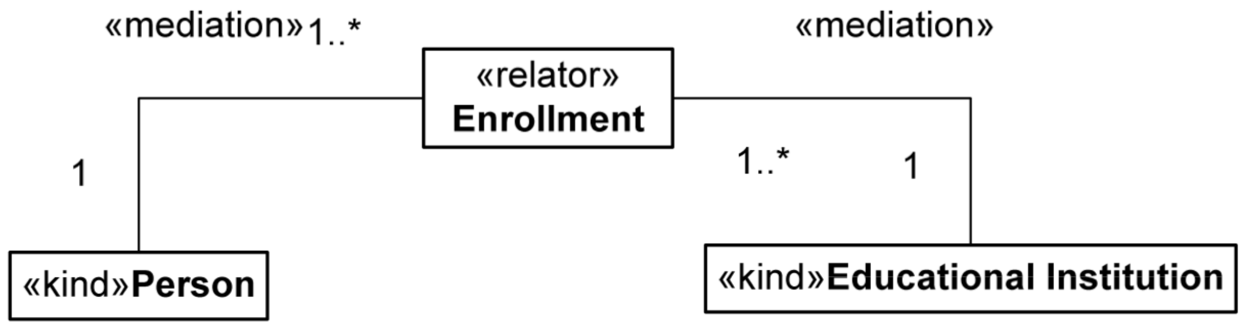
We define a relation of «*Mediation*» between a «*Relator*» and the entities it connects. Mediation is a type of *existential dependence* relation (a form of nonfunctional inherence). It can be derived from the relation between the **relata** and the *qua individiuals* that compose the relator and that inhere in the relata. A «*Relator*» must mediate at least two distinct individuals.

4.4.2 Common questions

Ask us some question if something is not clear . . .

4.4.3 Examples

EX1:



For more examples see «*Relator*», «*Material*», and «*Relator pattern*».

Quoted from:

GUIZZARDI, Giancarlo. *Introduction to Ontological Engineering.* [presentation] Prague: Prague University of Economics, 2011.

4.5 Characterization

4.5.1 Definition

«*Characterization*» is a relation between a *bearer type* and its *feature*. Feature is intrinsic (inherent) moment of its bearer type, and thus *existentially dependent* on the bearer. Feature may be stereotyped as «*Quality*» or «*Mode*». Feature *characterizes* a bearer type iff every instance of bearer exemplifies the feature.

4.5.2 Common questions

Ask us some question if something is not clear …

4.5.3 Examples

For examples see «*Quality*» and «*Mode*».

Source:

GUIZZARDI, Giancarlo. *Ontological Foundations for Structural Conceptual Models.* Enschede: CTIT, Telematica Instituut, 2005. GUIZZARDI, Giancarlo. *Introduction to Ontological Engineering.* [presentation] Prague: Prague University of Economics, 2011.

4.6 Derivation

4.6.1 Definition

«*Material*» relation can be completely **derived** (via «*Derivation*») from the «*Relator*» and the corresponding «*Mediation*» relations. Derivation makes the cardinality constraints of the mediation relations collapse (see «*Material*» relation, example 2).

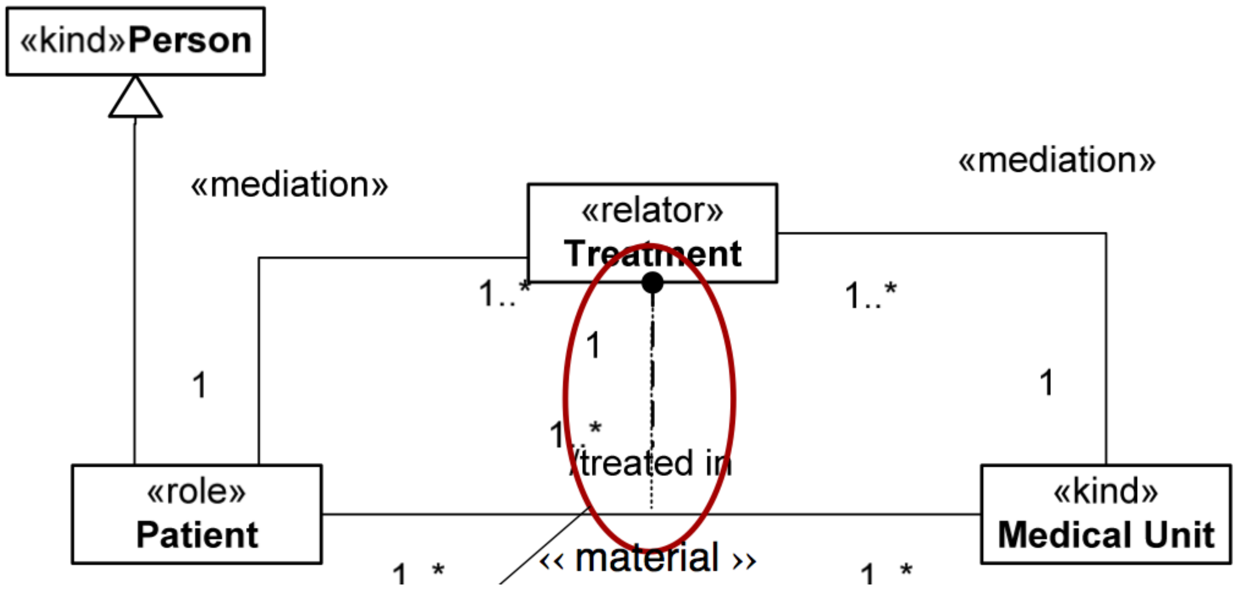
Also, several «*Material*» relations can be derived from a single «*Relator*» and «*Mediation*» relations (see «*Material*» relation, example 1).

4.6.2 Common questions

Ask us some question if something is not clear …

4.6.3 Examples

EX1:



For more examples see «*Relator*», «*Material*», and *Relator pattern*.

Quoted from:

GUIZZARDI, Giancarlo. *Introduction to Ontological Engineering.* [presentation] Prague: Prague University of Economics, 2011.

4.7 Structuration

4.7.1 Definition

«*Structuration*» relation allows structuring «*Quality*».

4.7.2 Common questions

Ask us some question if something is not clear . . .

4.7.3 Examples

For examples see «*Quality*».

Source:

GUIZZARDI, Giancarlo. *Introduction to Ontological Engineering.* [presentation] Prague: Prague University of Economics, 2011.

4.8 Part-Whole

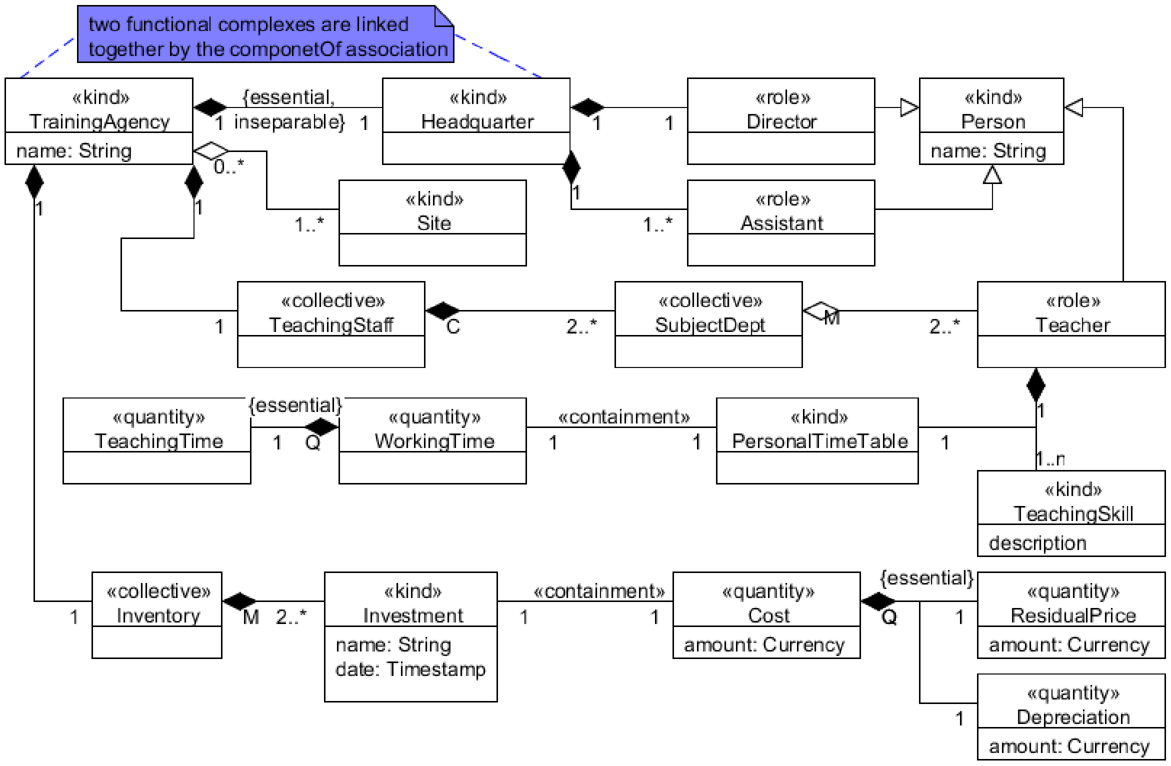
UML distinguishes between aggregation and composition only. OntoUML distinguishes among

- sharing
  - shared part (white)
  - exclusive part (black)
- multiplicity of relationship
  - mandatory part with respect to the whole
  - mandatory whole w.r.t. the part
  - mandatory non-rigid type (e.g. role, phase, mixin)

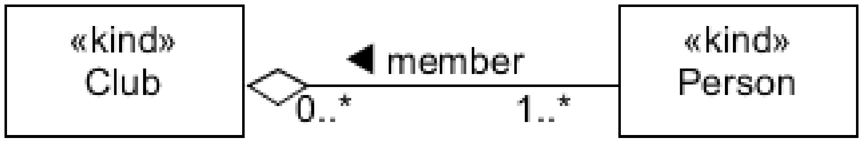
OntoUML also distinguishes among various types of wholes and their parts

- **functional whole** (and *ComponentOf* relation)
- *Collective* (and *SubCollectionOf* and *MemberOf* relations)
- *Quantity* (and *Containment* and *SubQuantityOf* relations)

4.8.1 Examples

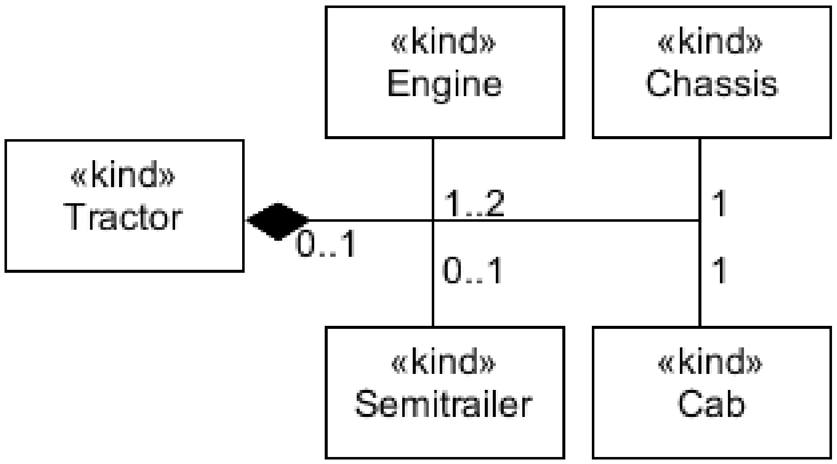


EX1:



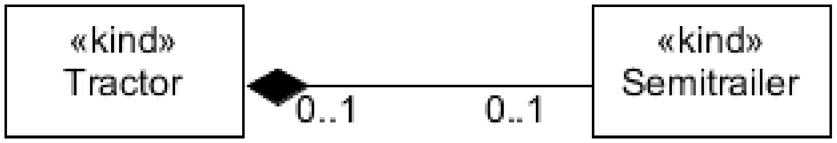
EX2:

Notice that maximum multiplicity of the whole is > 1.



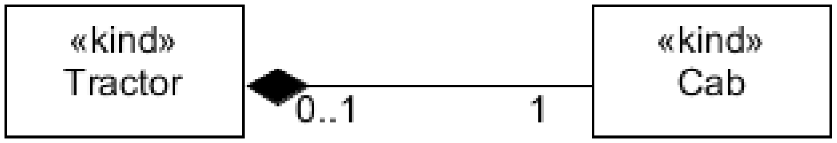
EX3:

Notice that maximum multiplicity of the whole is = 1.



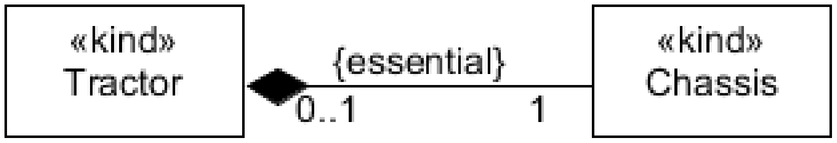
EX4:

Optional part w.r.t. the rigid whole. The whole doesn´t necessarily need any part.



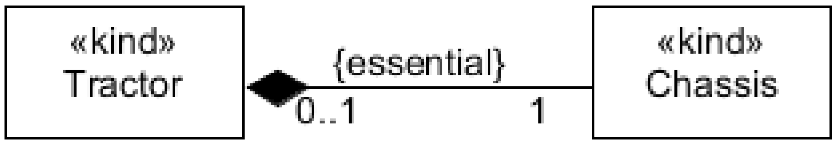
EX5:

Mandatory part w.r.t. the rigid whole. The whole does need a part, instances of the part may mute.



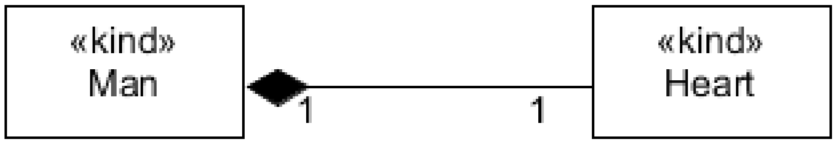
EX6:

Essential part w.r.t. the rigid whole. The whole does need a part, instances mustn´t mute.



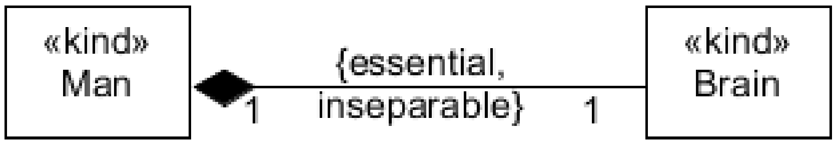
EX7:

Optional rigid whole w.r.t. the part. The part may exist alone, even without the whole.



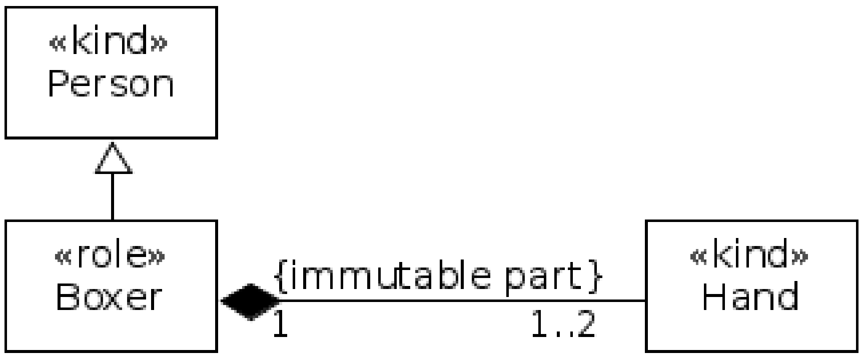
EX8:

Mandatory rigid whole w.r.t. the part. The part must belong to some whole, instances of the whole may mute.



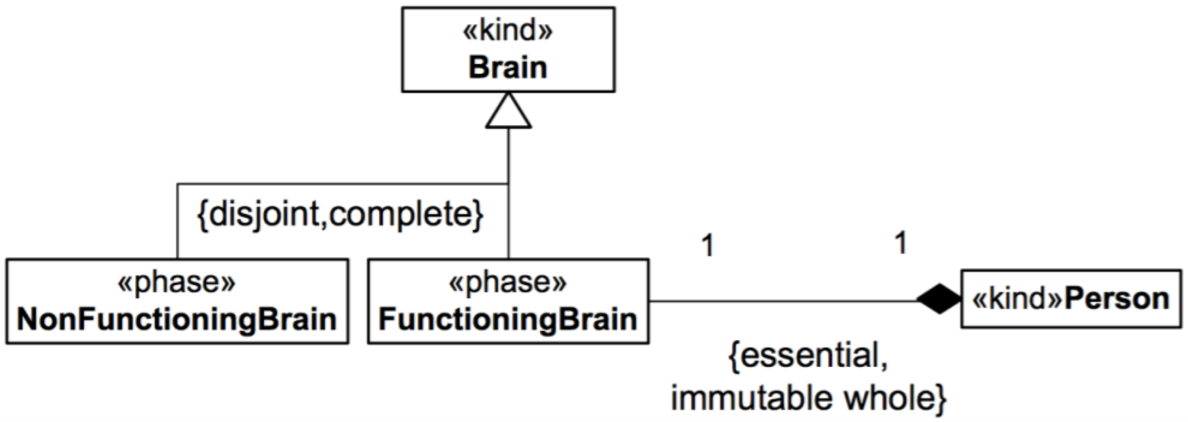
EX9:

Inseparable part of the rigid whole. The part must belong to the same whole, instances of the whole mustn´t mute.



EX10:

Immutable part of the antirigid whole. Whenever the whole exists in the particular role or phase, its parts must be still the same instances – they cannot not mute. Compare to *{essential}*.



EX11:

Immutable whole w.r.t. the antirigid part. Whenever the part exists in the particular role or phase, its wholes must be still the same instances – they cannot not mute. Instances of the whole may mute only as the part changes it´s role or phase.

References:

GUIZZARDI, Giancarlo. *Ontological Foundations for Structural Conceptual Models.* Enschede: CTIT, Telematica Instituut, 2005. GUIZZARDI, Giancarlo. *Introduction to Ontological Engineering.* [presentation] Prague: Prague University of Economics, 2011.

4.9 ComponentOf

4.9.1 Definition

«*ComponentOf*» is a parthood relation between two complexes. Examples include:

1. my hand is part of my arm;
2. a car engine is part of a car;
3. an Arithmetic and Logic Unit (ALU) is part of a Central Process Unit (CPU);
4. a heart is part of a circulatory system.

Transitivity holds for certain cases but not for others, it depends on context. «*ComponentOf*» relation obeys *weak supplementation principle* (at least 2 parts are required, may be of different types).

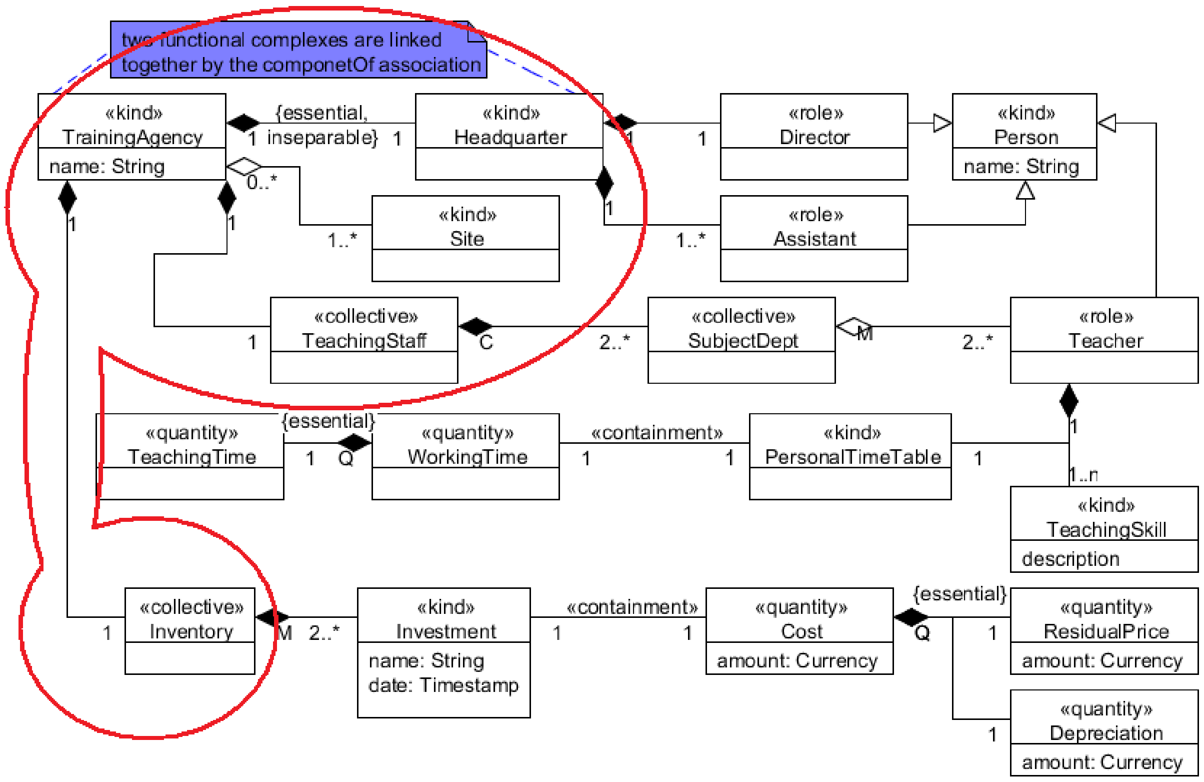
4.9.2 Constraints

C1: The classes connected to both association ends of this relation must represent universals whose instances are **functional complexes**.

4.9.3 Common questions

Ask us some question if something is not clear . . .

4.9.4 Examples



EX1:

See also *Part-Whole*.

References:

GUIZZARDI, Giancarlo. *Ontological Foundations for Structural Conceptual Models.* Enschede: CTIT, Telematica Instituut, 2005. GUIZZARDI, Giancarlo. *Introduction to Ontological Engineering.* [presentation] Prague: Prague University of Economics, 2011.

4.10 Containment

4.10.1 Definition

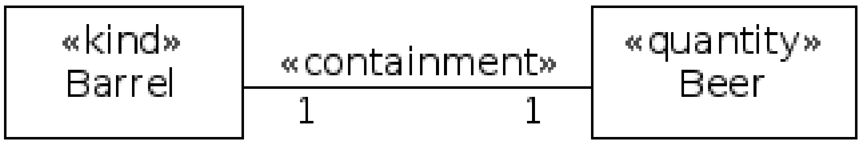
«*Containment*» is a relation between a container and its contents – a «*Quantity*», e.g., a barrel contains beer.

Multiplicities of the containment relation **must be exactly one** for the same reason as those of the «*SubQuantityOf*» relation.

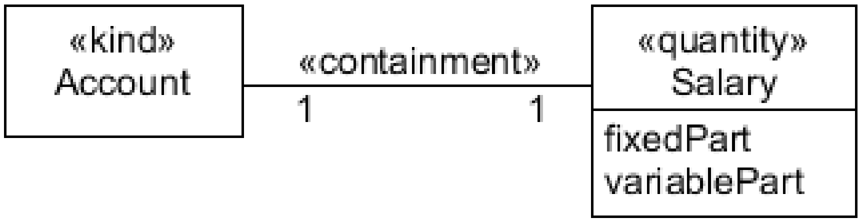
4.10.2 Common questions

Ask us some question if something is not clear . . .

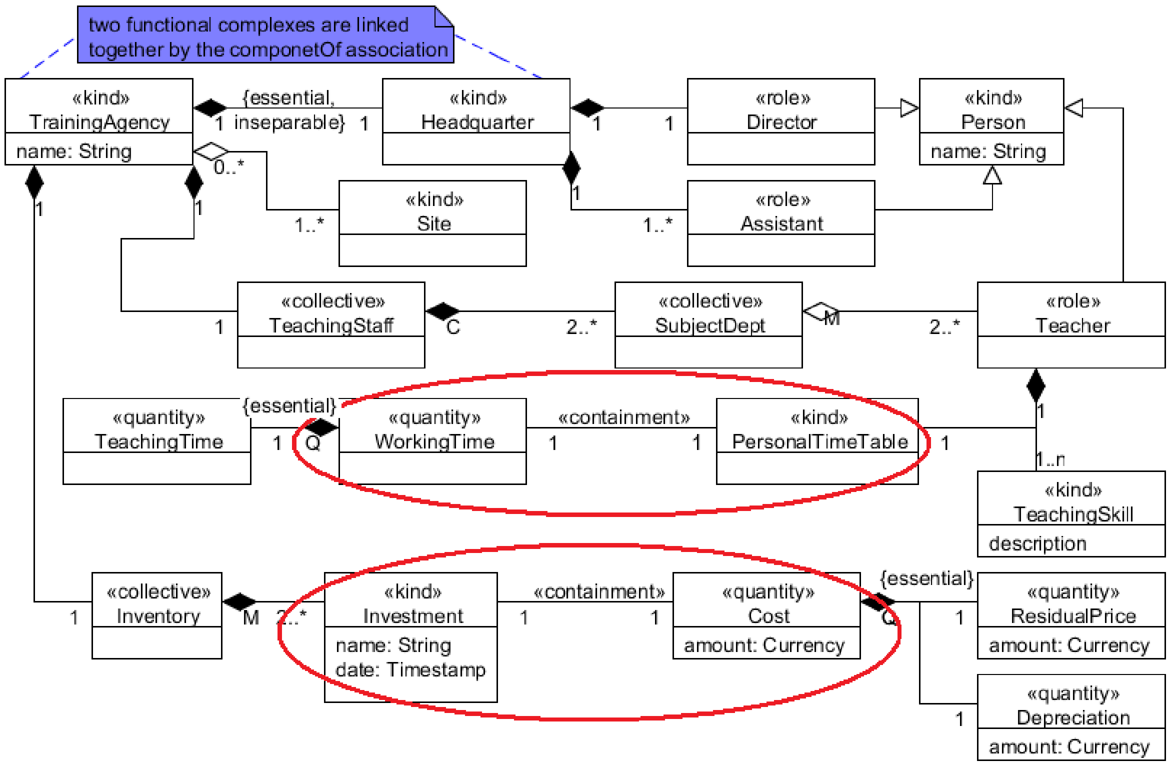
4.10.3 Examples



EX1:



EX2:



EX3:

See also

- *SubQuantityOf*
- *Part-Whole*

References:

GUIZZARDI, Giancarlo. *Ontological Foundations for Structural Conceptual Models.* Enschede: CTIT, Telematica Instituut, 2005. GUIZZARDI, Giancarlo. *Introduction to Ontological Engineering.* [presentation] Prague: Prague University of Economics, 2011.

4.11 MemberOf

4.11.1 Definition

«*MemberOf*» is a parthood relation between a **functional complex** or a «*Collective*» (as a part) and a «*Collective*» (as a whole).

Examples include:

1. a tree is part of forest;
2. a card is part of a deck of cards;
3. a fork is part of cutlery set;
4. a club member is part of a club.

«*MemberOf*» relation obeys *weak supplementation principle* (at least 2 parts are required, may be of different types). The memberOf relation is **intransitive.**

For example, Kazi, Bobek, Nemo and others are members of the *TJ Sokol Zizkov* Youth Tourist Club. The TJ Sokol Zizkov Youth Tourist Club is the member of the Association of the Youth Tourist Clubs. But Kazi, Bobek, Nemo and others are not members of the Association of the Youth Tourist Clubs, since not persons but only clubs may be members of the association. Although transitivity does not hold across two «*MemberOf*» relations, a «*MemberOf*» relation followed by «*SubCollectionOf*» is transitive.

4.11.2 Constraints

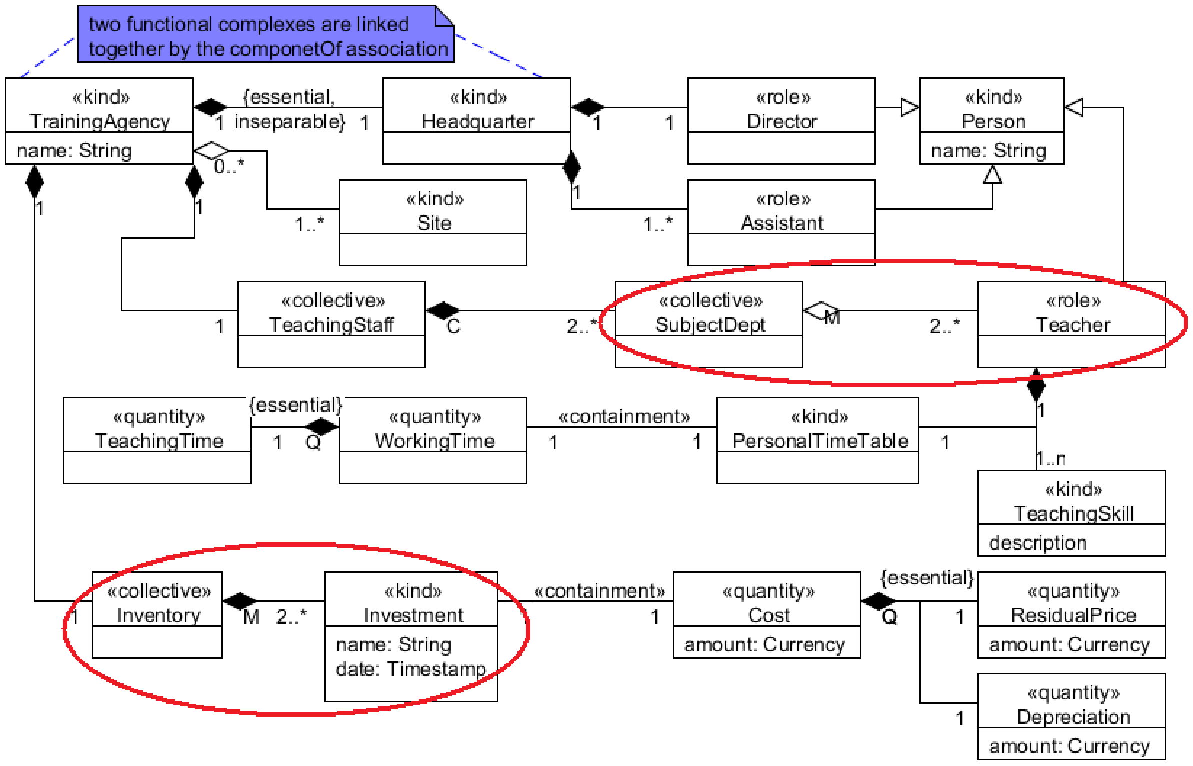
C1: This relation can only represent **essential** parthood if the object representing the whole is **extensional** (i.e. provided that adding or removing of any member changes the identity of the collective). In this case, all parthood relations in which the whole is extensional are constrained as **{essential}** parthood relations.

C2: The classifier connected to the whole end must be a «*Collective*». The classifier connected to the part end can be either a «*Collective*» or **functional complex**.

4.11.3 Common questions

Ask us some question if something is not clear . . .

4.11.4 Examples



EX1:

See also *Part-Whole*.

References:

GUIZZARDI, Giancarlo. *Ontological Foundations for Structural Conceptual Models.* Enschede: CTIT, Telematica Instituut, 2005. GUIZZARDI, Giancarlo. *Introduction to Ontological Engineering.* [presentation] Prague: Prague University of Economics, 2011.

4.12 SubCollectionOf

4.12.1 Definition

«*SubCollectionOf*» is a parthood relation between two collectives. Examples include:

1. the north part of the Black Forest is part of the Black Forest;
2. the collection of Jokers in a deck of cards is part of that deck of cards;
3. the collection of forks in cutlery set is part of that cutlery set;
4. the collection of male individuals in a crowd is part of that crowd.

The subCollectionOf relation can be **shareable** in some cases while **non-shareable** in others. For example, the Kulik siblings is a collection of three members: Marie, Vaclav, and Karel. The same Kulik siblings are sub-collection of the Kulik family, as well as a sub-collection of the FC Bilsko football team, as well as a sub-collection of the Voluntary Firefighter Unit in Bilsko. On contrary, the local organization of the Agrarian Party in Borovno is a sub-collection of the Agrarian Party, but must not be a sub-collection of any other political party, because the statutes prohibit it.

«*Collective*» is a type of collections (and collections are instances of collectives). Collection is an *integral whole,* or closure defined by a *unifying relation.* Closure means that no more parts or members can be added to the collection by its unifying relation.

Unlike «*Quantity*», «*Collective*» have members and their members may not be placed together (or connected *topologically*), but unified *intentionally* e.g. by the common role, or purpose, or social connection. Closure of the unifying relation makes the collective *maximal,* e.g. the football team is made up of all its members and no subset of its members can make up the same team. For this reason, the «*SubQuantityOf*» relation is **irreflexive**. Moreover, for the same reason, any super-collective can have at maximum one sub-collective of a given type. Finally, since every sub-collective of a super-collective is obtained by refining the unifying relation of the latter, the subCollectionOf relation is always **transitive**. Since collections are *maximal*, the «*SubCollectionOf*» parthood must have a **cardinality constraint of one and exactly one** in the sub-collection side. Addition or removal of a sub-collection (or even a member) of a collection may or may not change identity of the collection. E.g. new firefighter units are taken in the National Rescue System and some of the existing units cease to exist without changing identity of the National Rescue System. Similarly, the Voluntary Firefighter Unit in Karlik consists of three members: Velebil, Strasirybka, and Jech. Then Veselik applies for membership and is taken in the firefighter unit. It is still the same unit, its identity does not change. On contrary, imagine: Jarmila and Jaroslav are spouses. If Jaroslav died, the spousal will cease to exist. And the unifying relation of spousal does not even admit changing Jaroslav for Karel – such a change would change the identity of the spousal, as well. This means that collectives are not *extensional* (but *intentional*). That is why only the **weak supplementation axiom** holds for the subCollectionOf relation (unlike the «*SubQuantityOf*» relation, where the *strong supplementation axiom* holds). This axiom means among others that every super-collection must have at least two different types of sub-collections.

4.12.2 Constraints

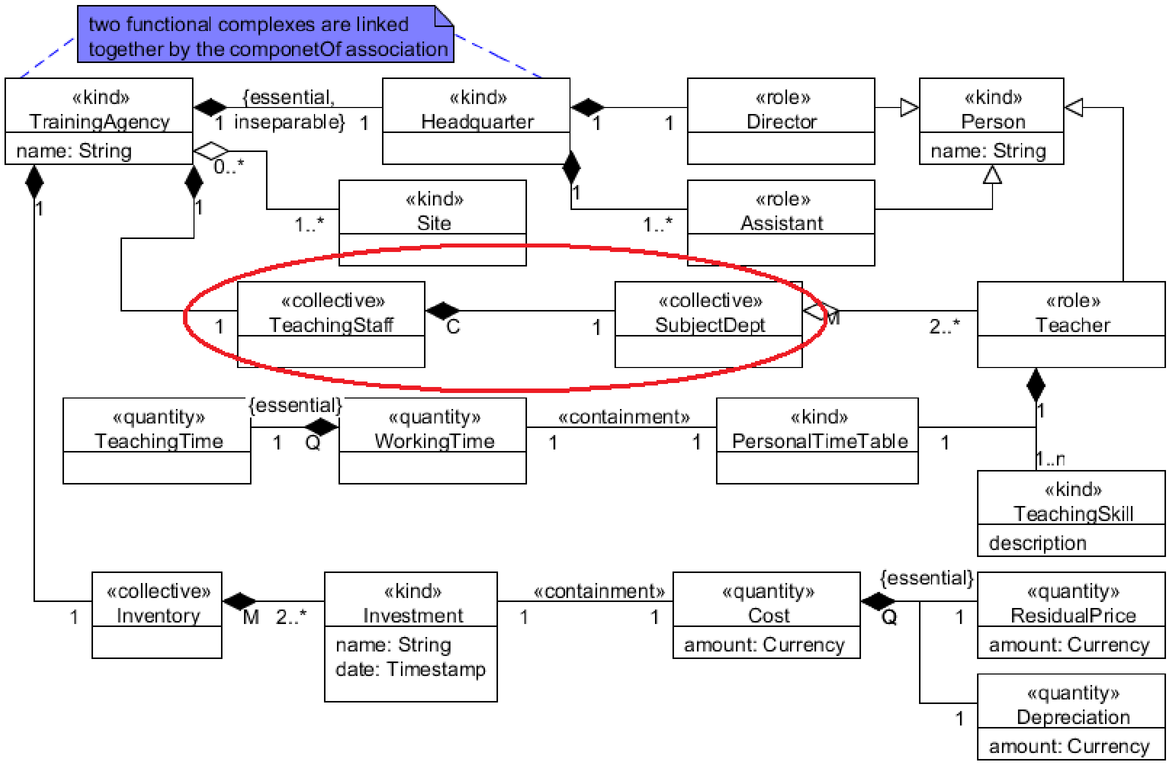
C1: The classes connected to both association ends of this relation must represent universals whose instances are *collectives*. *Collectives* are types as defined in the overview table above.

C2: The maximum cardinality constraint in the association end connected to the part must be one.

4.12.3 Common questions

Ask us some question if something is not clear . . .

4.12.4 Examples



EX1:

See also

- *Part-Whole*
- «*MemberOf*»

Quoted from:

GUIZZARDI, Giancarlo. *Ontological Foundations for Structural Conceptual Models.* Enschede: CTIT, Telematica Instituut, 2005. GUIZZARDI, Giancarlo. *Introduction to Ontological Engineering.* [presentation] Prague: Prague University of Economics, 2011.

4.13 SubQuantityOf

4.13.1 Definition

«*SubQuantityOf*» is a parthood relation between two quantities, e.g.:

1. alcohol is part of wine;
2. plasma is part of blood;
3. sugar is part of ice cream.

Quantities have not elements (or members). Since their members cannot be enumerated, they must be defined by a relation that unifies them into a connected whole (self-connectedness). Quantities are connected *topologically* (unlike e.g. collectives, which parts and members may not be placed together). *Topological connection* is characteristic for quantities and because of *topological connection,* sub-quantities cannot be shared among several super-quantities. For this reason, a subQuantityOf relation is always **non-sharable.** Since quantities do not have elements, they can be arbitrarily divided, like e.g. water. That´s why any quantity is defined to be *maximal portion* and can not be part of itself (water cannot be part of water). Since every part of a quantity is maximal (and self-connected), the SubQuantityOf parthood must have a **cardinality constraint of one and exactly one** in the sub-quantity side. E.g. since alcohol is a quantity (and, hence, maximal), there is exactly one quantity of alcohol which is part of a specific quantity of wine. Since quantity is *maximal,* it cannot have a quantity of the same kind as its part – i.e. the «*SubQuantityOf*» relation is **irreflexive**.

Nevertheless, a quantity can be part of another quantity (like glucose in wine) using the «*SubQuantityOf*» relation. The change of any of parts of the quantity changes the identity of the whole (i.e. quantities are *extensional* entities). That is why **the strong supplementation axiom** holds for the the «*SubQuantityOf*» relations (unlike «*SubCollectionOf*» relation, which on contrary holds only weaker axiom). For the same reason, all parts of a quantity are **essential** and «*SubQuantityOf*» relations are essential parthood relations. Further, since essential parthood relations are always transitive, «*SubQuantityOf*» is always **transitive**.

4.13.2 Constraints

C1: The «*SubQuantityOf*» relation is always non-shareable.

C2: A sub-quantity is always an **essential part** of its super-quantity (marked with {essential} constraint).

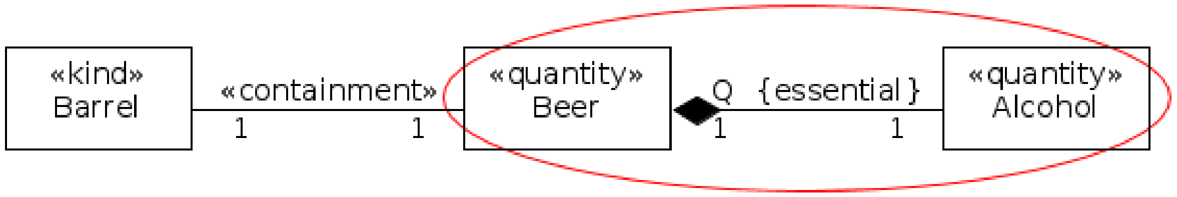
C3: The cardinality in the part-end must be exactly one.

C4: The «*SubQuantityOf*» **quantities** at its both ends. Quantities are types as defined in the overview table above.

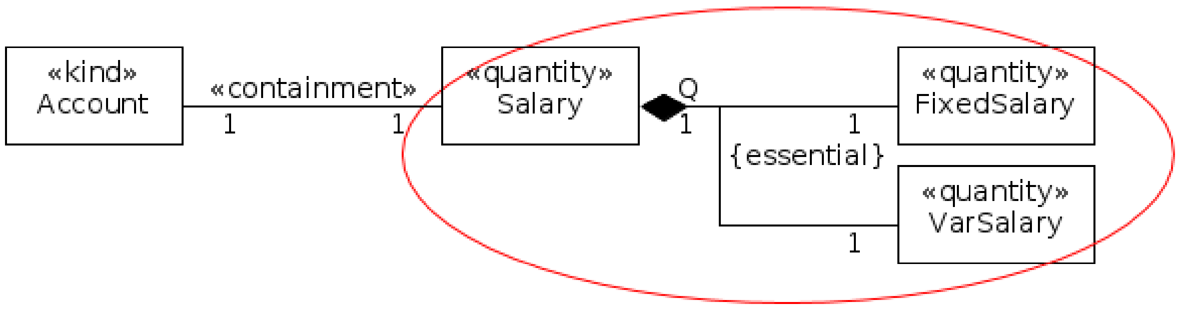
4.13.3 Common questions

Ask us some question if something is not clear . . .

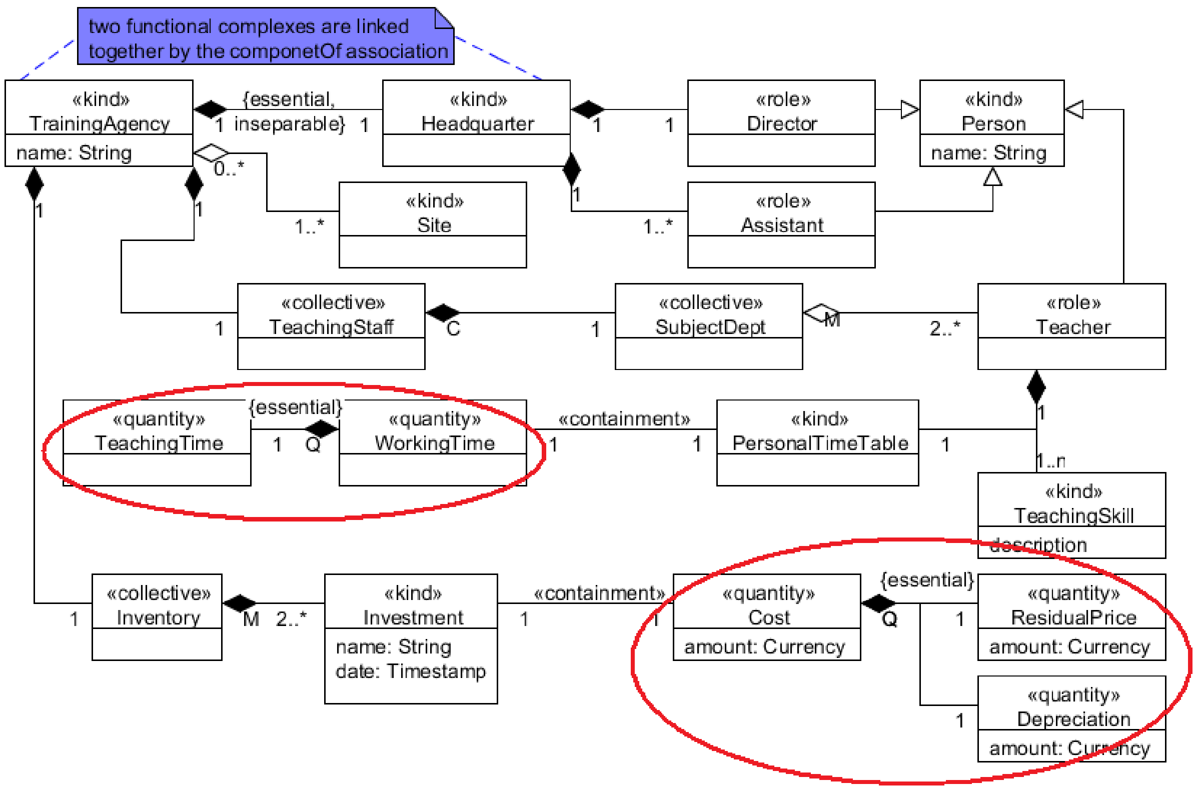
4.13.4 Examples



EX1:



EX2:



EX3:

See also

- :ref:'part-whole
- «*Containment*»

References:

GUIZZARDI, Giancarlo. *Ontological Foundations for Structural Conceptual Models.* Enschede: CTIT, Telematica Instituut, 2005. GUIZZARDI, Giancarlo. *Introduction to Ontological Engineering.* [presentation] Prague: Prague University of Economics, 2011.

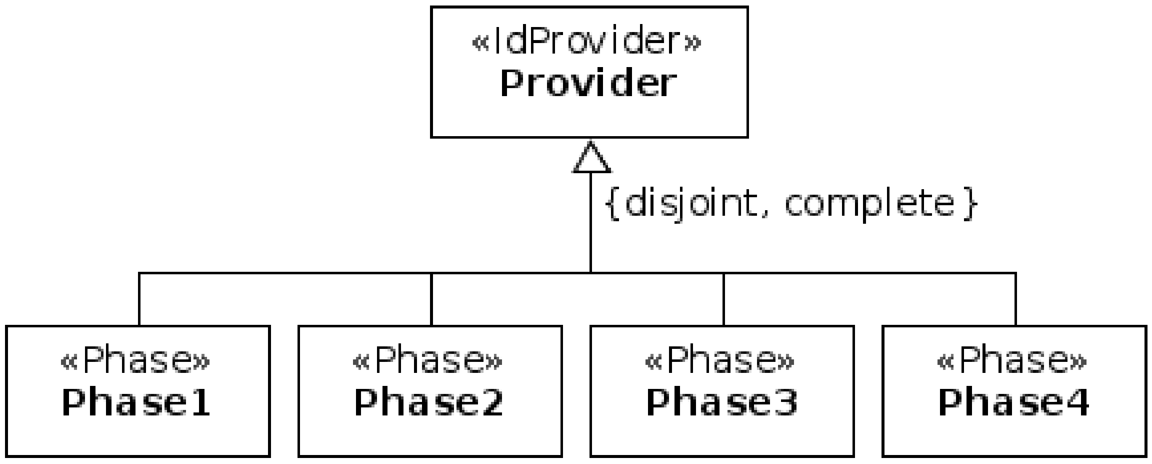
CHAPTER 5

OntoUML Pattern Catalogue

To help you build your OntoUML models faster, we are assembling a list of known patterns. Please notice that this list is still under construction, so some patterns might still be missing.

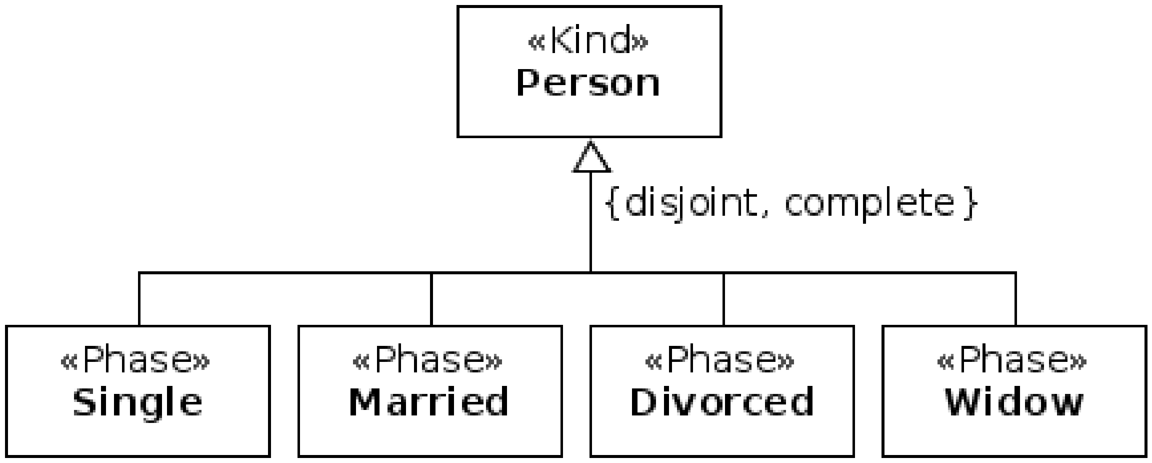
5.1 Phase Partition pattern

5.1.1 Generic pattern



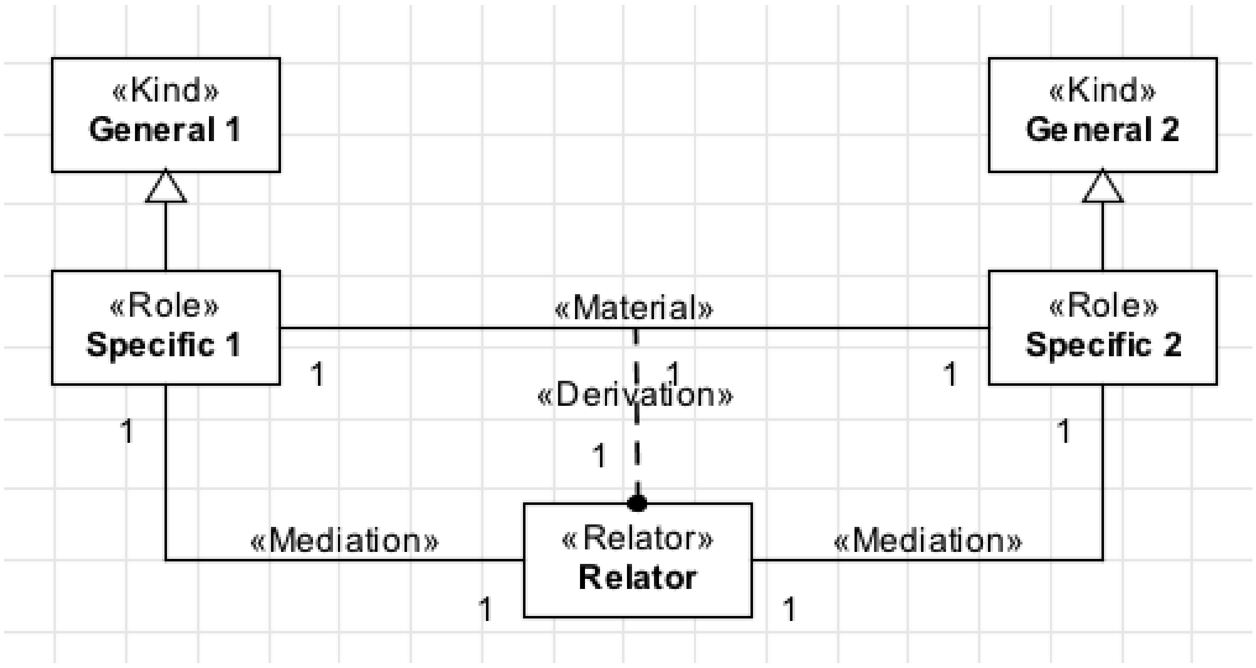
5.1.2 Examples

EX1:



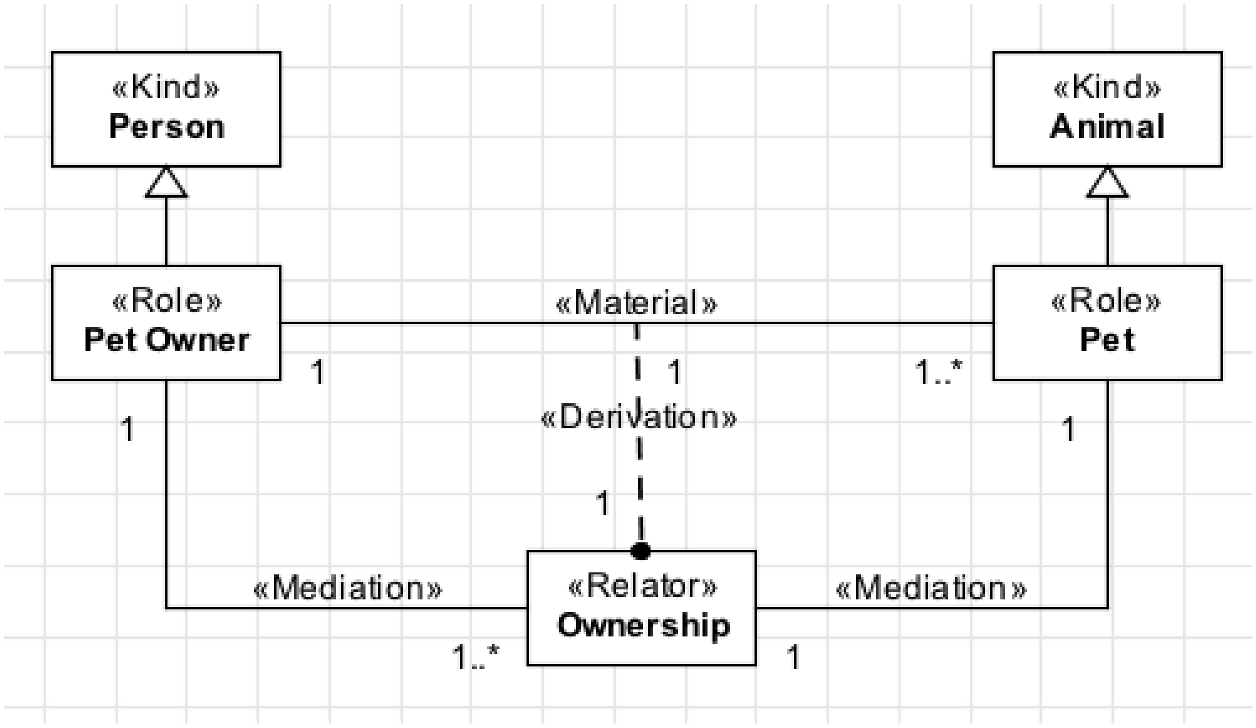
5.2 Relator pattern

5.2.1 Generic pattern

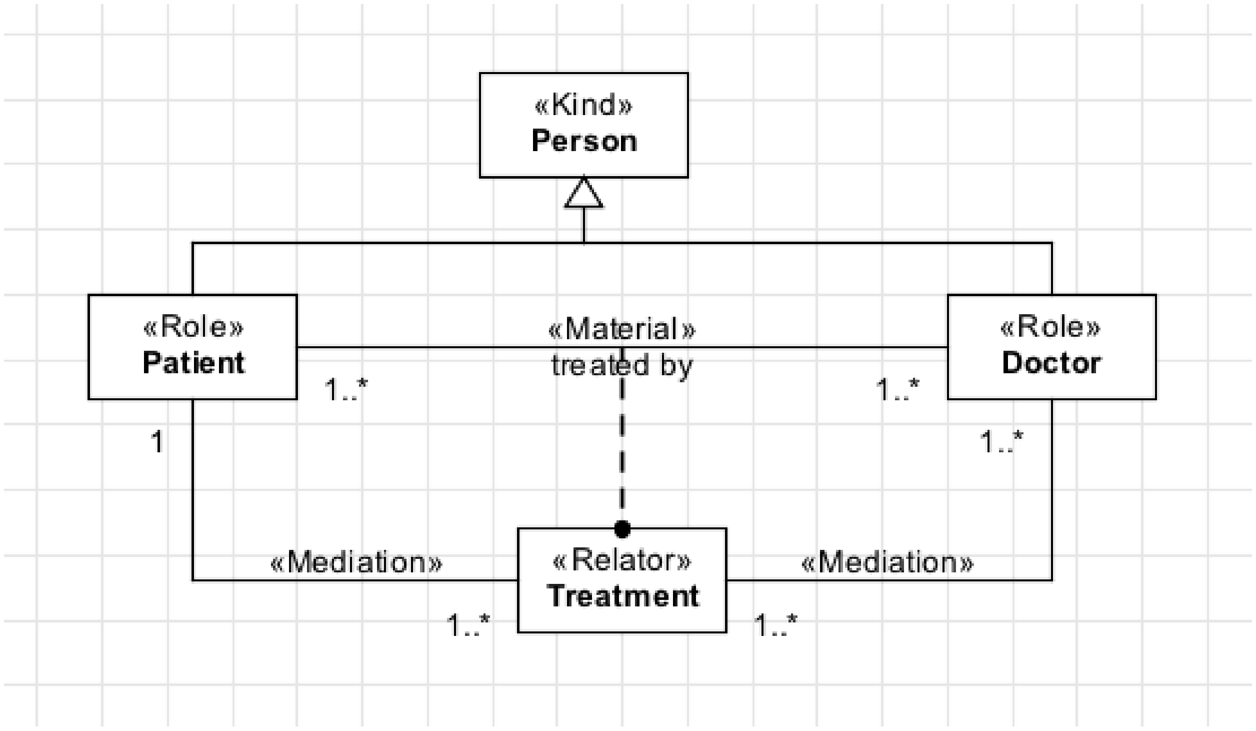


5.2.2 Examples

EX1:

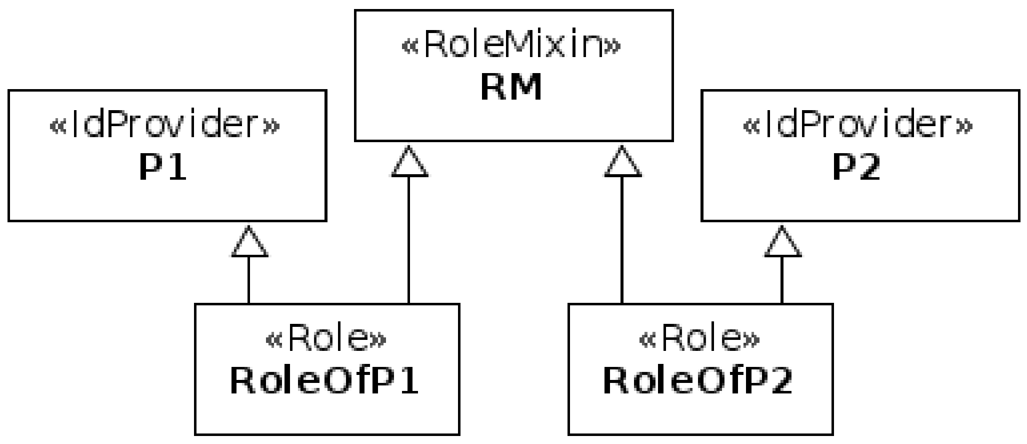

«Kind»
Person
«Kind»
Animal
«Role»
Pet Owner
«Material»
«Role»
Pet
1
1
1..*
«Derivation»
1
1
1
«Mediation»
«Relator»
Ownership
«Mediation»
1..*
1

EX2:


«Kind»
Person
«Role»
Patient
«Material»
treated by
«Role»
Doctor
1..*
1..*
1
1..*
«Mediation»
«Relator»
Treatment
«Mediation»
1..*
1..*

5.3 RoleMixin pattern

5.3.1 Generic pattern

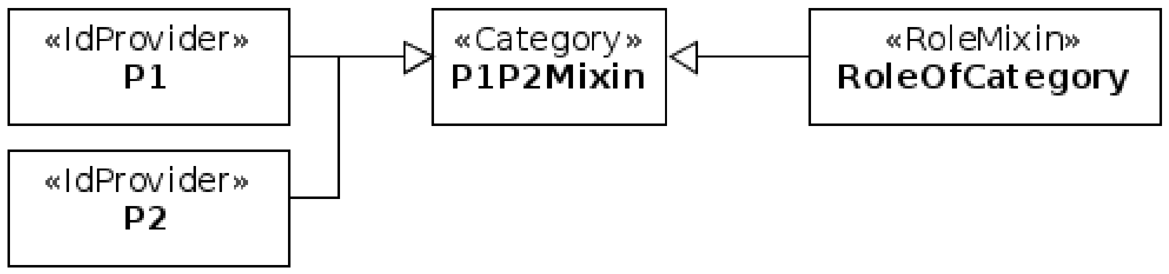


5.3.2 Examples

See RoleMixin

5.4 RoleMixin Alternative pattern

5.4.1 Generic pattern



5.4.2 Examples

See RoleMixin

CHAPTER 6

Contributing

This project is community-driven. Are you OntoUML enthusiast? We would like to invite you to cooperate on this documentation.

6.1 Reporting issues

Found a problem? Any uncertainty? Please create an issue on our GitHub repository github.com/OntoUML/OntoUML.

6.2 Solving issues

Feel free to solve any issue by yourself. You need just a GitHub account, you will fix the problem in your fork of the repository and then submit a pull request to the original one. Also, you can fork the repository and try to propose your OntoUML changes for the future version.

6.3 Documentation guidelines

- Keep the file structure, if you want to propose some big changes, please create an issue where we can discuss such big change.
- Do not use line breaks unless ending paragraph. In 21st century all human-usable editors and IDEs have functionality called "word wrap" that is configurable per user. Why should someone with wide screen see only 80 characters per line if want more?
- Try to be consistent, maximize readers understanding (do not expect any IT or Ontology expertise), interlink with other related pages and also label your pages.
- Take a look at Sphinx docs and reStructuredText Markup Specification.

CHAPTER 7

TODOs

Todo: To be added . . .

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ontouml/checkouts/init-ontouml/classes/aspects/quality/examples.rst, line 4.)

Todo: To be added . . .

(The original entry is located in classes/aspects/quality/examples.rst, line 4.)

Todo: To be added . . .

(The *original entry* is located in /home/docs/checkouts/readthedocs.org/user_builds/ontouml/checkouts/init-ontouml/classes/sortals/quantity/examples.rst, line 4.)

Todo: To be added . . .

(The original entry is located in classes/sortals/quantity/examples.rst, line 4.)

Todo: This topic will be covered soon. . .

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/ontouml/checkouts/init-ontouml/theory/dependency.rst, line 6.)

Todo: This topic will be covered soon. . .

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/ontouml/checkouts/init-ontouml/theory/objects_events.rst, line 6.)

CHAPTER 8

Indices and tables

- genindex
- modindex
- search