Concept Modeling Semantics

In order to improve UML's suitability for modeling real-world concepts, the Concept Modeler interprets the UML standard to allow subproperties, existential quantification constraints, and universal quantification constraints. In addition to those interpretations, the Concept Modeler uses a small UML profile to add the capabilities of global properties, necessary and sufficient properties, and other future capabilities. Simply having or applying a «Concept Model» stereotype on a UML package causes anything within that package to have this interpretation, and allows these added capabilities.

The following subsections describe how the Concept Modeler interprets the UML standard and augments it to describe conceptualizations.

- Class

- Classes
- Anonymous union classes
- Advanced Modeling Patterns
- Equivalent classes
- Conditions
- Property ownership
- Global properties
- Equivalent properties
- Subproperty

- Property chain
- Property restrictions
- Cardinality restrictions
- Inverse properties
- Object properties
- Annotation and annotation properties
- Preferred Annotation Property
- Generalization
- Multiplicities

- IRI tagged value
- Complement Of
- Importing OWL
- Concept model export URI style
- OWL export folder
- UPCM library in CCM
- Equivalent classes in NLG
- Working with superclass intersection
- Intersection

Class

In the concept modeling interpretation of the UML standard, a class is a set or collection of individual things called *members*. The members of a class in a concept model are either things that exist in the real world around us, or things we can imagine to exist, such as unicorns. For example, depending on the stated scope of a concept model, the members of a Chair class would include the one you sit upon to do your work, or the one in a warehouse ready to be shipped to a customer.

UML Classes
The owner is no longer shown by default on a UML Class in Concept Modeling diagrams.

Anonymous Classes

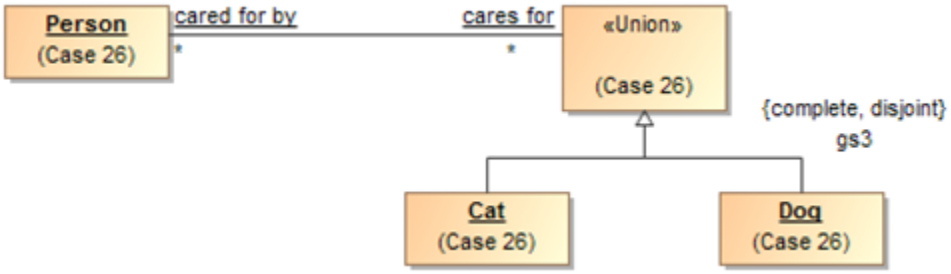
An unnamed UML Class represents an anonymous class expression, used for defining conditions, including domains and ranges.

Complex Chain
Importing a complex chain of restrictions no longer shares anonymous UML classes among multiple concept models.

Anonymous Union Classes

An anonymous union is an unnamed class used to represent a set of classes that can be used as a type of a property. An anonymous union class always implies a complete subclass generalization. (See Complete Subclasses.)

The following diagram states that an instance of a Person may have a value of type Cat or Dog for the *cares for* property. The diagram also states that an instance of a Cat or a Dog may have a value of type Person for the *cared for by* property.

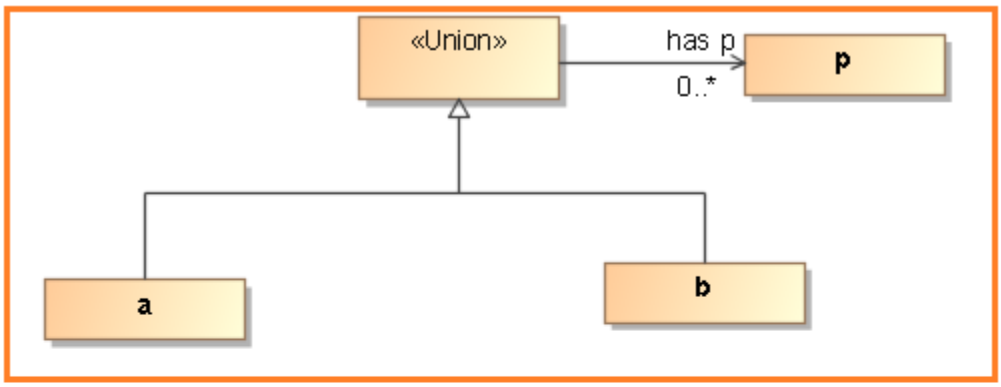


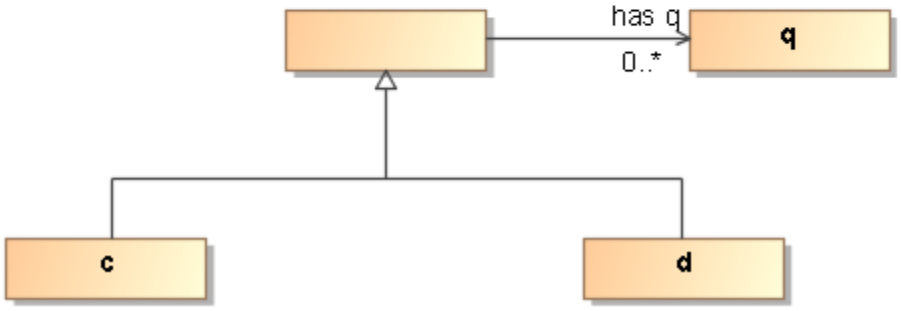
An anonymous union class.

In an ontology, if anonymous union, with same classes within the union, is used in multiple places, the Concept Modeler can distinguish it when importing the ontology. In other words, if the anonymous union has the same union members, the Concept Modeler will identify it as the same anonymous union.

AVAILABLE FROM 19.0 SP2

An anonymous Class with Subclasses but without an explicit «Union» is now treated the same as a union of its Subclasses. The following image show what that means.





The boxed diagram represents the usual relationship between an anonymous Union class. Notice the stereotype, *«Union»*. On the other hand, the diagram not boxed represents a similar structure but there's no anonymous Union class with the stereotype, *«Union»*. The modeling tool is programmed to consider the non-stereotyped class with the subclasses as a Union. Let's see that in the exported OWL of this model.

Reading the following code piece, notice the ObjectUnionOf between A and B, and the ObjectIUnionOf between C and D.

```
# Object Property: :hasP (has p)
ObjectPropertyDomain(:hasP ObjectUnionOf(:A :B))
ObjectPropertyRange(:hasP :P)
# Object Property: :hasQ (has q)
ObjectPropertyDomain(:hasQ ObjectUnionOf(:C :D))
ObjectPropertyRange(:hasQ :Q)
```

The full ontology is displayed below:

```
Ontology(<http://example.com/ontology/uniondemo>

Declaration(Class(:A))
Declaration(Class(:B))
Declaration(Class(:C))
Declaration(Class(:D))
Declaration(Class(:P))
Declaration(Class(:Q))
Declaration(ObjectProperty(:hasP))
Declaration(ObjectProperty(:hasQ))

#############################
#   Object Properties
#############################

# Object Property: :hasP (has p)

AnnotationAssertion(rdfs:label :hasP "has p"^^xsd:string)
ObjectPropertyDomain(:hasP ObjectUnionOf(:A :B))
ObjectPropertyRange(:hasP :P)

# Object Property: :hasQ (has q)

AnnotationAssertion(rdfs:label :hasQ "has q"^^xsd:string)
ObjectPropertyDomain(:hasQ ObjectUnionOf(:C :D))
ObjectPropertyRange(:hasQ :Q)



#############################
#   Classes
#############################

# Class: :A (a)

AnnotationAssertion(rdfs:label :A "a"^^xsd:string)

# Class: :B (b)

AnnotationAssertion(rdfs:label :B "b"^^xsd:string)

# Class: :C (c)

AnnotationAssertion(rdfs:label :C "c"^^xsd:string)

# Class: :D (d)

AnnotationAssertion(rdfs:label :D "d"^^xsd:string)

# Class: :P (p)

AnnotationAssertion(rdfs:label :P "p"^^xsd:string)

# Class: :Q (q)

AnnotationAssertion(rdfs:label :Q "q"^^xsd:string)


)
```