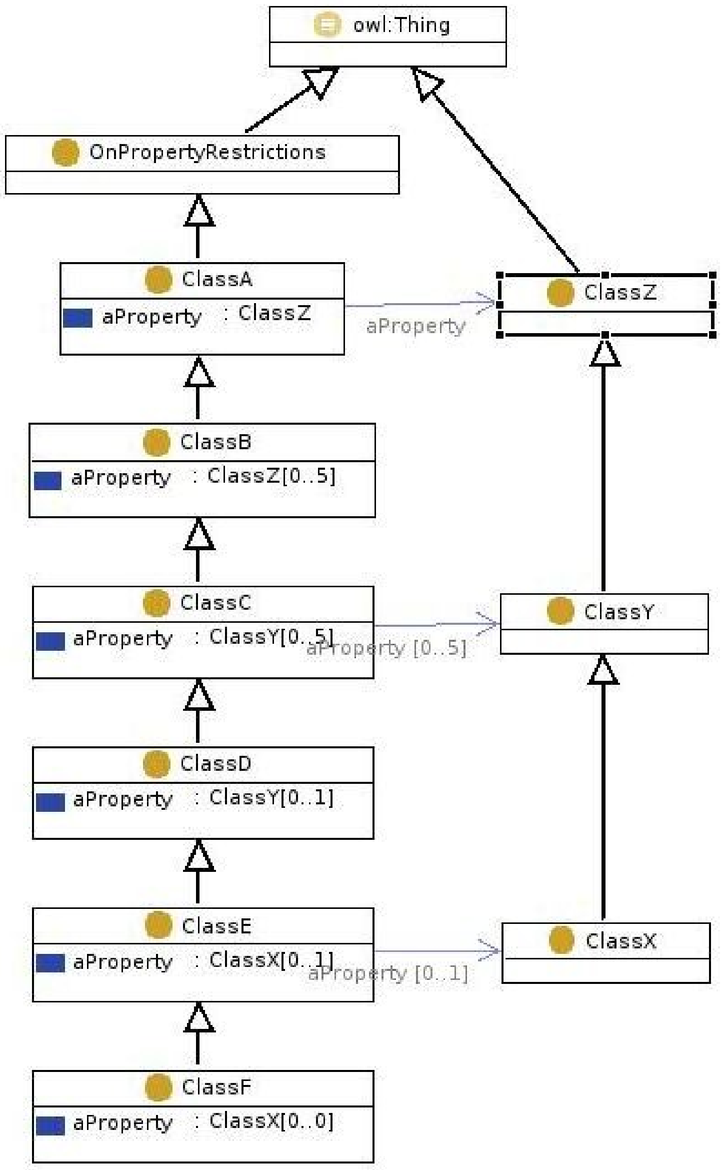
Property Restrictions, Ranges and Accessor Methods

A short document about possible use cases of restrictions, property ranges and the effects on the creation of accessor methods for a property

Ex 1: Nested Restrictions

This example demonstrates the implications related to multiple nested restrictions.

A class hierarchy of nested subclasses ClassA to ClassF successively add restrictions to the property aProperty. A class tree of helper classes consisting of the classes ClassZ to ClassX is used as range (allValuesFrom) restriction.



For the classes *Class* to *ClassF* this leads to the generation of accessor methods as presented in the following sections. *AddAll* and *removeAll* are not shown. Deprecated properties are shown ~~strike through~~.

Property Restrictions, Ranges and Accessor Methods

ClassA

- existsAProperty(),
- ~~Iterator<Object> listAProperty()~~, Iterator<ClassZ> listAProperty()
  >>>> Name clash! <<<<
- ~~hasAProperty(Object value)~~, hasAProperty(ClassZ value)
- ~~addAProperty(Object value)~~, addAProperty(ClassZ value)
- ~~removeAProperty (Object value)~~, removeAProperty(ClassZ value)

ClassB

- existsAProperty(),
- ~~Iterator<Object> listAProperty()~~, Iterator<ClassZ> listAProperty(),
  >>>> Name clash! <<<<
- ~~hasAProperty(Object value)~~, hasAProperty(ClassZ value)
- ~~addAProperty(Object value)~~, addAProperty(ClassZ value)
- ~~removeAProperty (Object value)~~, ~~removeAProperty(ClassZ value)~~

ClassC

- existsAProperty(),
- ~~Iterator<Object> listAProperty()~~, ~~Iterator<ClassZ> listAProperty()~~,
  Iterator<ClassY> listAProperty() **>>>> Name clash! <<<<**
- ~~hasAProperty(Object value)~~, ~~hasAProperty(ClassZ value)~~, hasAProperty(ClassY value)
- ~~addAProperty(Object value)~~, ~~addAProperty(ClassZ value)~~, addAProperty(ClassY value)
- ~~removeAProperty (Object value)~~, ~~removeAProperty(ClassZ value)~~,
  removeAProperty(ClassY value)

ClassD

- existsAProperty(),
- ClassY getAProperty() setAProperty(ClassY value), removeAProperty()
- ~~Iterator<Object> listAProperty()~~, ~~Iterator<ClassZ> listAProperty()~~,
  ~~Iterator<ClassY> listAProperty~~() **>>>> Name clash! <<<<**
- ~~hasAProperty(Object value)~~, ~~hasAProperty(ClassZ value)~~, ~~hasAProperty(ClassY value)~~
- ~~addAProperty(Object value)~~, ~~addAProperty(ClassZ value)~~, addAProperty(ClassY value)

- ~~removeAProperty (Object value)~~, ~~removeAProperty(ClassZ value)~~, ~~removeAProperty(ClassY value)~~

ClassE

- existsAProperty(),
- ~~ClassY getAProperty() setAProperty(ClassY value)~~, removeAProperty()
- ClassZ getAProperty() setAProperty(ClassZ value)
- ~~Iterator<Object> listAProperty(), Iterator<ClassZ> listAProperty(), Iterator<ClassY> listAProperty()~~ **>>>> Name clash! <<<<**
- ~~hasAProperty(Object value), hasAProperty(ClassZ value), hasAProperty(ClassY value)~~
- ~~addAProperty(Object value)~~, ~~addAProperty(ClassZ value)~~, addAProperty(ClassY value)
- ~~removeAProperty (Object value)~~, ~~removeAProperty(ClassZ value)~~, ~~removeAProperty(ClassY value)~~

ClassF

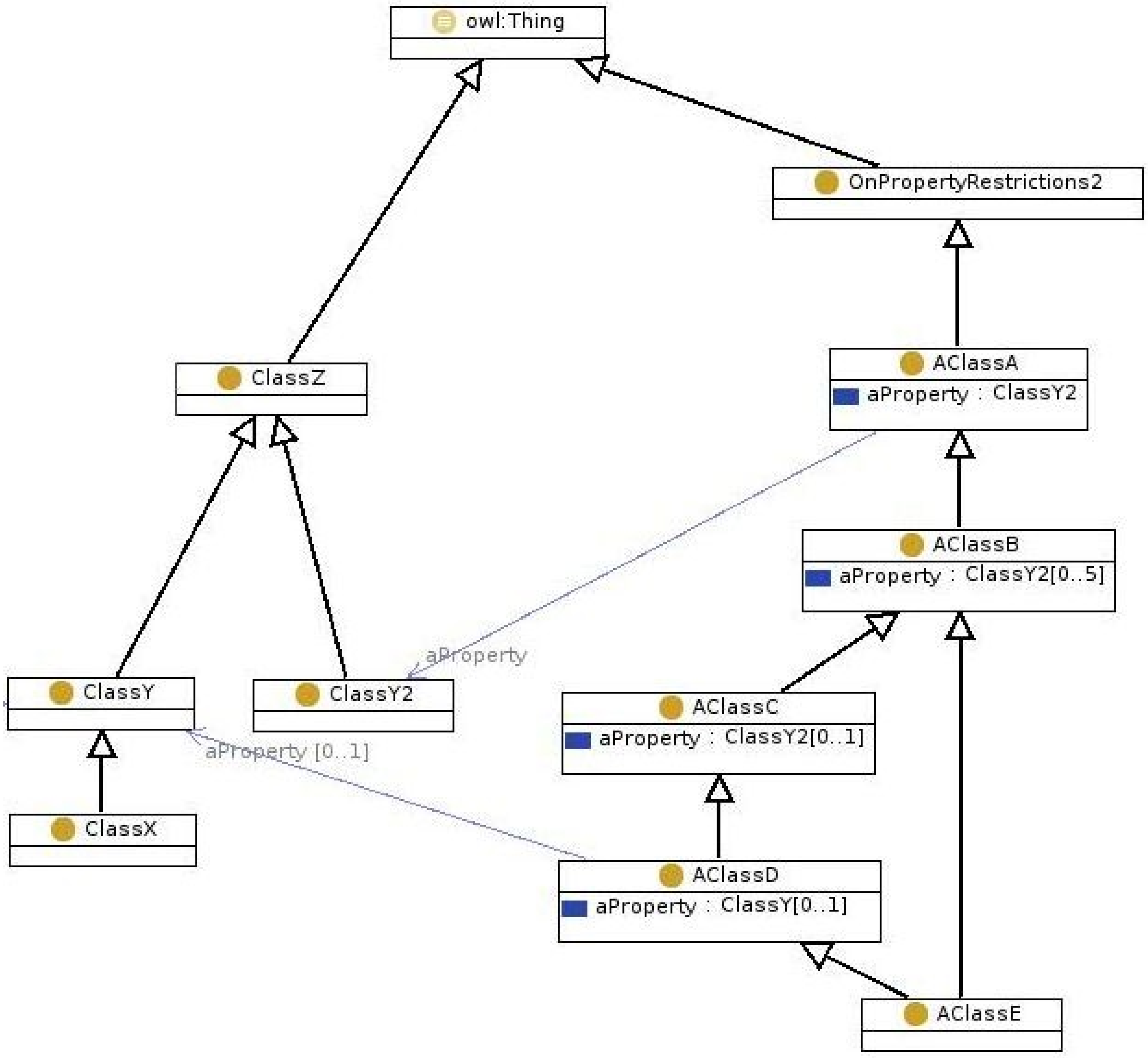
- ~~existsAProperty(),~~
- ~~ClassY getAProperty() setAProperty(ClassY value), removeAProperty()~~
- ~~ClassZ getAProperty() setAProperty(ClassZ value)~~
- ~~Iterator<Object> listAProperty(), Iterator<ClassZ> listAProperty(), Iterator<ClassY> listAProperty()~~ **>>>> Name clash! <<<<**
- ~~hasAProperty(Object value), hasAProperty(ClassZ value), hasAProperty(ClassY value)~~
- ~~hasAProperty(Object value), hasAProperty(ClassZ value), hasAProperty(ClassY value)~~
- ~~addAProperty(Object value)~~, ~~addAProperty(ClassZ value)~~, addAProperty(ClassY value)
- ~~removeAProperty (Object value)~~, ~~removeAProperty(ClassZ value)~~, ~~removeAProperty(ClassY value)~~

Results:

1. Generating accessor methods for all possible ranges may lead to issues with identical method signatures with different return types.
2. The number of accessor methods is large to very large.

Ex 2: Multiple Inheritance

This example shows the effects of multiple class inheritance and the implications on the property in the class hierarchy. The class hierarchy and property restrictions as shown below are used. A hierarchy of helper classes ClassZ to ClassX is used.



Results:

1. AClassE inherits restrictions via two parent class graphs.
2. For AClassE the range of the property is given as ClassY2 defined in AClassB and ClassY defined in AClassD, i. e. as intersection of ClassY2 and ClassY. As ClassY and ClassY2 are sibling classes the intersection is empty therefore leading to a range of aProperty in AClassE of owl:Nothing.

Conclusions

1. Generating accessor methods for all possible ranges may lead to issues with identical method signatures with different return types. An alternative naming schema like listAPropertyAsRange() can solve this problem yet leads to unwieldy method names and a large number of different methods.
2. If multiple inheritance is used in the ontology the determination of the range or cardinality of a property requires a traversal of all inheritance paths / graphs and the creation of new intersection classes paths or a reasoner. Very complex.