Design Rule 1 Parent Class should not have an Attribute referring to a Child Class

```
context Class inv:
  let children:Set(NamedElement) = self.namespace.oclAsType(
      Package).packagedElement->
    select(pe:PackageableElement|pe.oclIsTypeOf(Class) and
      pe.oclAsType(Class).allParents()->includes(self)) in
  self.ownedAttribute->forAll(p:Property|p.type.oclIsTypeOf(
      Class) implies
    children->excludes(p.type.oclAsType(Class)))
```

Design Rule 2 Parent Class should not have a Method with a Parameter referring to a Child Class

```
context Class inv:
  let children:Set(NamedElement) = self.namespace.oclAsType(
      Package).packagedElement->
    select(pe:PackageableElement|pe.oclIsTypeOf(Class) and
      pe.oclAsType(Class).allParents()->includes(self)) in
  self.ownedOperation->forAll(o:Operation|o.ownedParameter->
    forAll(p:Parameter|p.type.oclIsTypeOf(Class) implies
      children->excludes(p.type.oclAsType(Class))))
```

Design Rule 3 Message Action must be defined as an Operation in Receiver's Class

```
context Message inv:
  self.receiveEvent.oclAsType(InteractionFragment).covered->
    forAll(represents.type.oclAsType(Class).ownedOperation->
        exists(name=self.name))
```

Design Rule 4 Message Direction must match Class Association

```
context Message inv:
  self.receiveEvent.oclAsType(InteractionFragment).covered->
    exists(let rc:Class=represents.type.oclAsType(Class) in
      self.sendEvent.oclAsType(InteractionFragment).covered->
        exists(let sc:Class=represents.type.oclAsType(Class) in
          sc.ownedAttribute->exists(association<>null implies
              type=rc)))
```

Design Rule 5 The connected Classifier of the Association End should be included in the Namespace of the Association

```
context Association inv:
  self.memberEnd<>null and self.memberEnd->
    forAll(p|p.type<>null and p.type.namespace=self.namespace)
```

Design Rule 6 AssociationEnds must have unique Names within the Association

```
context Association inv:
  self.memberEnd->forAll(p1,p2:Property|p1<>p2 implies p1.name<>
      p2.name)
```

Design Rule 7 At most one AssociationEnd may be an Aggregation or Composition

```
context Association inv:
  self.memberEnd->size()>0 implies
    self.memberEnd->select(p|p.aggregation<>AggregationKind::
        none)->size()<=1
```

Design Rule 8 A Classifier my not declare an Attribute that has been declared in Parent Classifiers

```
context Class inv:
  self.allParents()->forAll(c:Classifier|c.oclAsType(Class).
      ownedAttribute->
    forAll(p:Property|p.class.ownedAttribute->collect(name)->
        excludes(p.name)))
```

Design Rule 9 A Class may use Unique Attribute Names

```
context Class inv:
  self.ownedAttribute->forAll(p1,p2:Property|p1<>p2 implies p1.
      name<>p2.name)
```

Design Rule 10 A Classifier my not belong by Composition to more than one Composite Classifier

```
context Property inv:
  (self.association<>null and self.aggregation=AggregationKind::
      composite) implies
    (self.upper>=0 and self.upper<=1)
```

Design Rule 11 The Elements owned by a Namespace must have unique Names

```
context Package inv:
  self.packagedElement->forAll(e1,e2:PackageableElement|(e1<>e2)
       implies (e1.name<>e2.name))
```

Design Rule 12 An Interface can only contain Public Operations and no Attributes

```
context Interface inv:
  self.ownedAttribute->forAll(pr:Property|pr.association<>null
      or
    pr.visibility=VisibilityKind::public) and
    self.ownedOperation->forAll(o:Operation|o.visibility=
        VisibilityKind::public)
```

Design Rule 13 No two Class Operations may have the same Signature

```
context Class inv:
  self.ownedOperation->forAll(o1,o2:Operation|o1<>o2 implies
    (o1.name <> o2.name or o1.ownedParameter->size() <> o2.
        ownedParameter->size() or
      let ops1:Collection(Type)=o1.ownedParameter->collect(type)
          in
      let ops2:Collection(Type)=o2.ownedParameter->collect(type)
          in
        ops1->exists(t:Type|ops2->excludes(t))or ops2->exists(t:
            Type|ops1->excludes(t))))
```

Design Rule 14 Operation Parameters must have unique Names

```
context Operation inv:
  self.ownedParameter->forAll(p1,p2:Parameter|p1<>p2 implies p1.
      name<>p2.name)
```

Design Rule 15 The Type of Operation Parameters must be included in the Namespace of the Operation Owner

```
context Operation inv:
  self.ownedParameter->forAll(p:Parameter|p.type<>null implies
    p.type.namespace=self.owner.oclAsType(Class).namespace)
```

Design Rule 16 The Parent must be included in the Namespace of the Generalizable Element

```
context Generalization inv:
  self.source->forAll(e1:Element|e1.oclIsKindOf(NamedElement)
      implies
    self.target->forAll(e2:Element|e2.oclIsKindOf(NamedElement)
        and
    e1.oclAsType(NamedElement).namespace = e2.oclAsType(
        NamedElement).namespace))
```

Design Rule 17 No circular Inheritance allowed

```
context Class inv:
  not self.allParents()->includes(self)
```

Design Rule 18 Statechart Action must be defined as an Operation in the Owner's Class

```
context Transition inv:
  self.owner.oclAsType(Region).stateMachine<>null implies
    let classifier:BehavioredClassifier=self.owner.oclAsType(
        Region).stateMachine.context in
      classifier.oclIsTypeOf(Class) implies
        classifier.oclAsType(Class).ownedOperation->exists(o:
            Operation|o.name=self.name)
```

Design Rule 19 An Operation has at most one return Parameter

```
context Operation inv:
  self.ownedParameter->select(p:Parameter|p.direction=
      ParameterDirectionKind::return)->size()<=1
```