1. Implementation

To verify our algorithm we have created a simple, global optimization framework which enabled us to execute and test all of the algorithms descirbe throughout this article.

We decided to use Java [**?**] as a programming language and runtime environment for this project.

Using the framework we can find optima of real, mulimodal, multidimensional and continuous functions. We just need to provide the framework with the concrete fitness function implementation, domain and by specifing the problem domain.

The framework principle is to separate the interfaces from implementation. Our API provides interfaces for evolutionary algorithms, fitness assignment and fitness deterioration, genetic oprators like selection and reproducion, phenotype, clustering algorithms and many more. This makes it compact, extensible and easy to indroduce new implementations. For the detailed description of the classes and interfaces see subsection 'Implementation in Java'.

1.1. Architecture

The project consists of six main packages, which communicates with the others through clearly specified interfaces and together constitute a complete, lightweight framework for testing and execution of the global optimization algorithms especially the evolutionary algorithms.

- *Algorithm module*
- *Clustering module*
- *Fitness deterioration module*
- *Evolutionary Algorithms module*
- *Printing Modul*
- *Statistics Utils*

1.2. Implementation in Java

clean structure, good test coverage, modular architercture, extensible,

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.s

  <bean
    class="org.springframework.beans.factory.config.PropertyPlaceholderConfigu
    <property name="locations" value="classpath:algorithm.properties" />
  </bean>

  <bean id="algorithm"
    class="ki.edu.agh.algorithm.SequentialDeteriorationEAWithOpticsClustering"
    <property name="problemDomain">
      <bean class="ki.edu.agh.problem.MultimodalRealSpaceProblem">
        <!-- set domain and functor -->
        <property name="domain">
          <bean class="ki.edu.agh.problem.Domain">
            <property name="multidimensionalCube">
              <list>
                <!-- first dimension interval -->
                <bean class="ki.edu.agh.problem.Interval">
                  <constructor-arg value="-2." />
                  <constructor-arg value="4." />
                </bean>
                <!-- second dimension interval -->
                <bean class="ki.edu.agh.problem.Interval">
                  <constructor-arg value="-2." />
                  <constructor-arg value="4." />
                </bean>
              </list>
            </property>
          </bean>
        </property>
        <property name="fitnessFunction">
          <bean class="ki.edu.agh.fintess.StandardFitnessFunction">
            <constructor-arg>
              <bean class="ki.edu.agh.functors.BiModalFunction" />
            </constructor-arg>
          </bean>
        </property>
        <!-- specify if the the problem is minimization or maximization -->
        <property name="minimization" value="false" />
      </bean>
    </property>

    <property name="iterationCount" value="${algorithm.iterationCount}" />

    <!-- clustering algorithm -->
    <property name="clusteringAlgorithm">
      <bean class="ki.edu.agh.clustering.optics.OpticsClustering">
        <constructor-arg>
          <bean
            class="ki.edu.agh.clustering.optics.OpticsParamteres">
            <property name="minPoints" value="${optics.min_pts}" />
            <!-- problem's domain dependent -->
            <property name="epsilon" value="${optics.eps}" />
          </bean>
        </constructor-arg>
      </bean>
    </property>

    <!-- specifies evolutionary algorithm used -->
    <property name="evolutionaryAlgorithm" ref="sga" />

    <!-- fitness deterioration algorithm -->
    <property name="fitnessDeterioration">
      <bean
        class="ki.edu.agh.deterioration.SimpleGaussianFitnessDeterioration" />
    </property>
  </bean>

  <bean id="sga" class="ki.edu.agh.evolutionary.algorithm.SGA">
    <!-- SGA configuration -->
  </bean>
</beans>
```

1.2.1. Technologies

- Spring [?] - application framework
- Maven [?] - project management and build automation

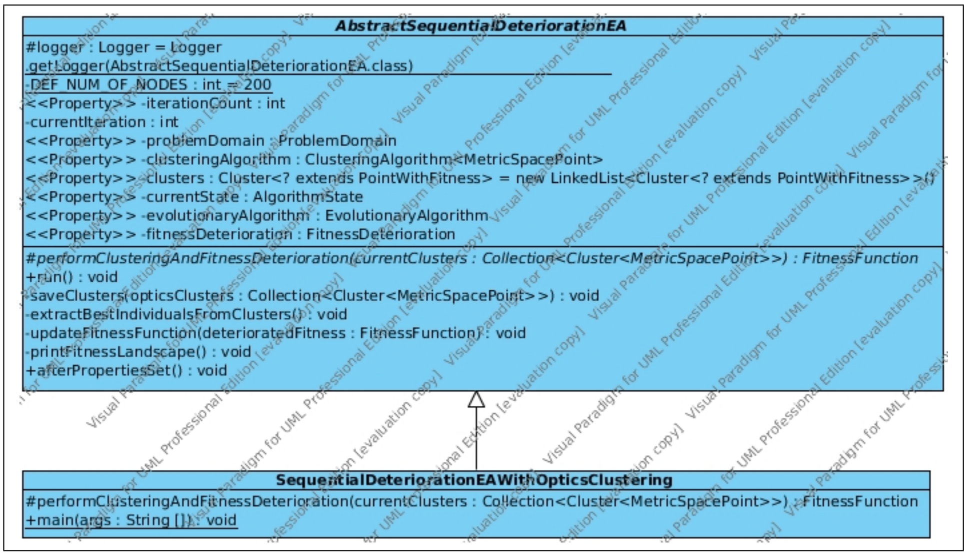

AbstractSequentialDeteriorationEA
#logger : Logger = Logger
.getLogger(AbstractSequentialDeteriorationEA.class)
-DEF_NUM_OF_NODES : int = 200
<<Property>> -iterationCount : int
-currentIteration : int
<<Property>> -problemDomain : ProblemDomain
<<Property>> -clusteringAlgorithm : ClusteringAlgorithm<MetricSpacePoint>
<<Property>> -clusters : Cluster<? extends PointWithFitness> = new LinkedList<Cluster<? extends PointWithFitness>>()
<<Property>> -currentState : AlgorithmState
<<Property>> -evolutionaryAlgorithm : EvolutionaryAlgorithm
<<Property>> -fitnessDeterioration : FitnessDeterioration
#performClusteringAndFitnessDeterioration(currentClusters : Collection<Cluster<MetricSpacePoint>>) : FitnessFunction
+run() : void
-saveClusters(opticsClusters : Collection<Cluster<MetricSpacePoint>>) : void
-extractBestIndividualsFromClusters() : void
-updateFitnessFunction(deterioratedFitness : FitnessFunction) : void
-printFitnessLandscape() : void
+afterPropertiesSet() : void
SequentialDeteriorationEAWithOpticsClustering
#performClusteringAndFitnessDeterioration(currentClusters : Collection<Cluster<MetricSpacePoint>>) : FitnessFunction
+main(args : String []) : void

Figure 1.1: Main algorithm package

- Mockito [?] - testing framework

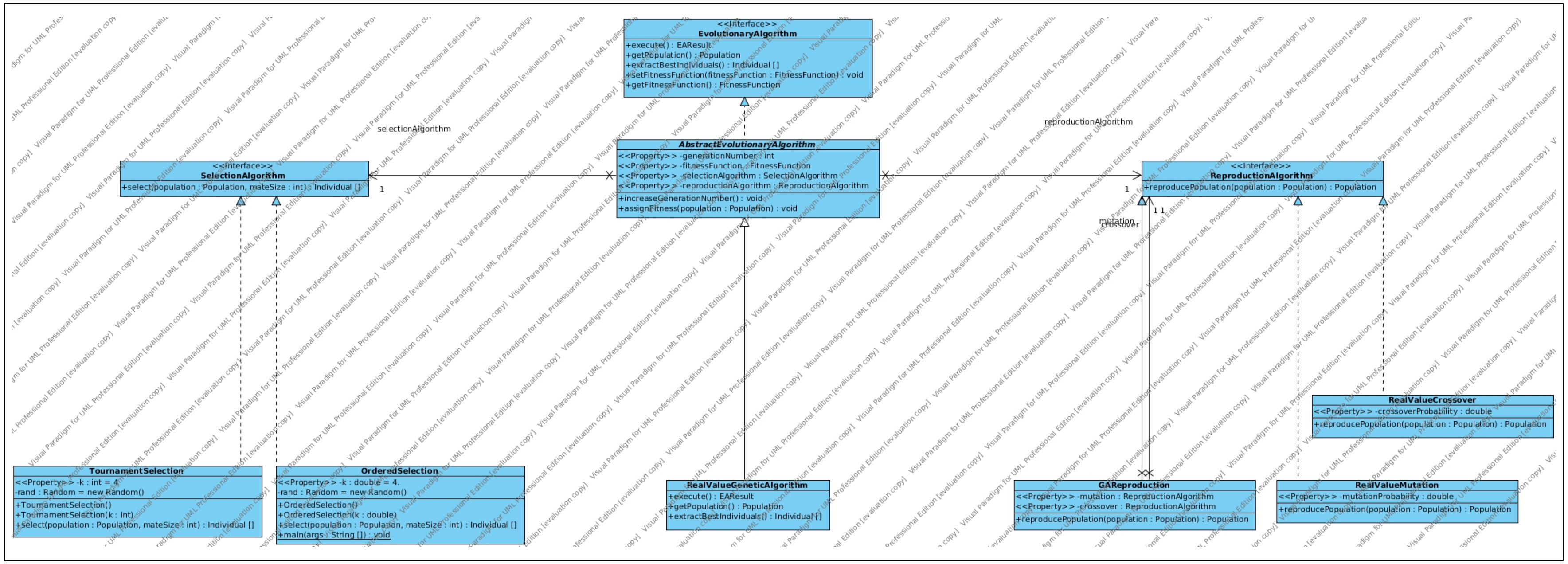
- JAMA [?] - linear algebra package

Spring, Maven, JUnit, Mockito, JAMA, TDD approach

1.2.2. Diagrams

class diagrams, sequence diagrams

Below you may find class diagrams for each of the module implemented in our framework. It shows a general overview of the structure of a system: used classes, their attributes, operations and the relationships between the classes.



<<Interface>>
EvolutionaryAlgorithm
+execute() : EAResult
+getPopulation() : Population
+extractBestIndividuals() : Individual []
+setFitnessFunction(fitnessFunction : FitnessFunction) : void
+getFitnessFunction() : FitnessFunction
selectionAlgorithm
reproductionAlgorithm
<<Interface>>
SelectionAlgorithm
+select(population : Population, mateSize : int) : Individual []
1
AbstractEvolutionaryAlgorithm
<<Property>> -generationNumber : int
<<Property>> -fitnessFunction : FitnessFunction
<<Property>> -selectionAlgorithm : SelectionAlgorithm
<<Property>> -reproductionAlgorithm : ReproductionAlgorithm
+increaseGenerationNumber() : void
+assignFitness(population : Population) : void
1
<<Interface>>
ReproductionAlgorithm
+reproducePopulation(population : Population) : Population
mutation
crossover
1 1
TournamentSelection
<<Property>> -k : int = 4
-rand : Random = new Random()
+TournamentSelection()
+TournamentSelection(k : int)
+select(population : Population, mateSize : int) : Individual []
OrderedSelection
<<Property>> -k : double = 4.
-rand : Random = new Random()
+OrderedSelection()
+OrderedSelection(k : double)
+select(population : Population, mateSize : int) : Individual []
+main(args : String []) : void
RealValueGeneticAlgorithm
+execute() : EAResult
+getPopulation() : Population
+extractBestIndividuals() : Individual []
GAReproduction
<<Property>> -mutation : ReproductionAlgorithm
<<Property>> -crossover : ReproductionAlgorithm
+reproducePopulation(population : Population) : Population
RealValueCrossover
<<Property>> -crossoverProbability : double
+reproducePopulation(population : Population) : Population
RealValueMutation
<<Property>> -mutationProbability : double
+reproducePopulation(population : Population) : Population

Figure 1.2: Evolutionary algorithms package

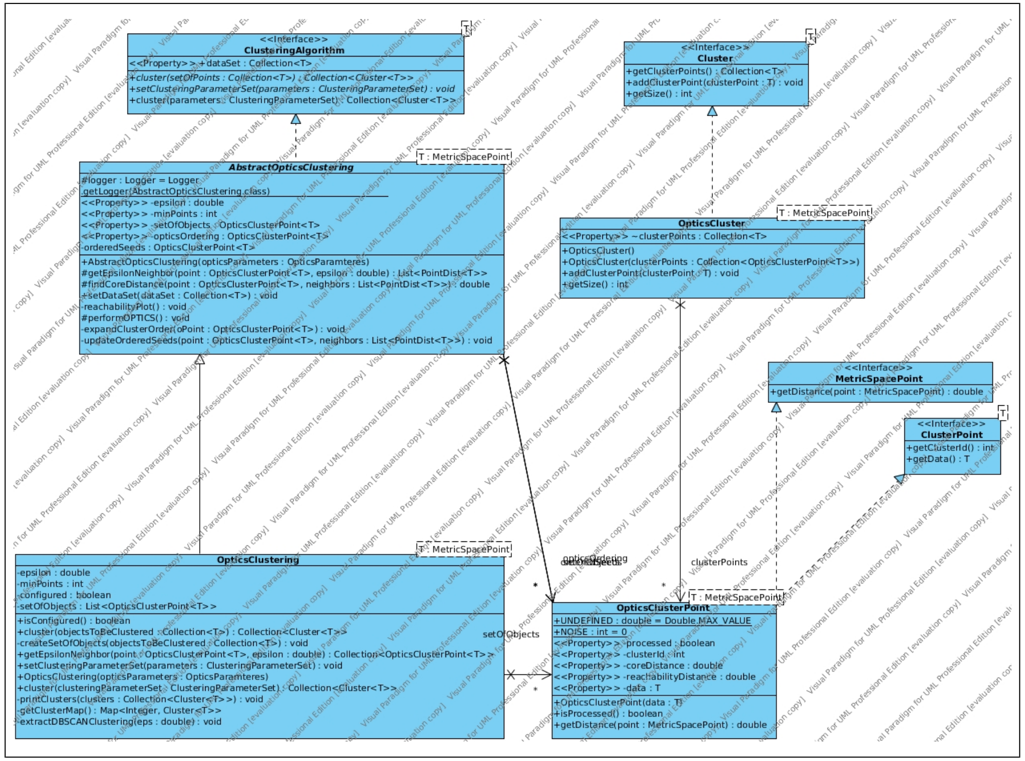

<<Interface>>
ClusteringAlgorithm
T
<<Property>> +dataSet : Collection<T>
+cluster(setOfPoints : Collection<T>) : Collection<Cluster<T>>
+setClusteringParameterSet(parameters : ClusteringParameterSet) : void
+cluster(parameters : ClusteringParameterSet) : Collection<Cluster<T>>
AbstractOpticsClustering
T : MetricSpacePoint
#logger : Logger = Logger.getLogger(AbstractOpticsClustering.class)
<<Property>> -epsilon : double
<<Property>> -minPoints : int
<<Property>> -setOfObjects : OpticsClusterPoint<T>
<<Property>> -opticsOrdering : OpticsClusterPoint<T>
-orderedSeeds : OpticsClusterPoint<T>
+AbstractOpticsClustering(opticsParameters : OpticsParamteres)
#getEpsilonNeighbor(point : OpticsClusterPoint<T>, epsilon : double) : List<PointDist<T>>
#findCoreDistance(point : OpticsClusterPoint<T>, neighbors : List<PointDist<T>>) : double
+setDataSet(dataSet : Collection<T>) : void
-reachabilityPlot() : void
#performOPTICS() : void
-expandClusterOrder(oPoint : OpticsClusterPoint<T>) : void
-updateOrderedSeeds(point : OpticsClusterPoint<T>, neighbors : List<PointDist<T>>) : void
OpticsClustering
T : MetricSpacePoint
-epsilon : double
-minPoints : int
-configured : boolean
-setOfObjects : List<OpticsClusterPoint<T>>
+isConfigured() : boolean
+cluster(objectsToBeClustered : Collection<T>) : Collection<Cluster<T>>
-createSetOfObjects(objectsToBeClustered : Collection<T>) : void
+getEpsilonNeighbor(point : OpticsClusterPoint<T>, epsilon : double) : Collection<OpticsClusterPoint<T>>
+setClusteringParameterSet(parameters : ClusteringParameterSet) : void
+OpticsClustering(opticsParameters : OpticsParamteres)
+cluster(clusteringParameterSet : ClusteringParameterSet) : Collection<Cluster<T>>
-printClusters(clusters : Collection<Cluster<T>>) : void
-getClusterMap() : Map<Integer, Cluster<T>>
-extractDBSCANClustering(eps : double) : void
<<Interface>>
Cluster
T
+getClusterPoints() : Collection<T>
+addClusterPoint(clusterPoint : T) : void
+getSize() : int
OpticsCluster
T : MetricSpacePoint
<<Property>> ~clusterPoints : Collection<T>
+OpticsCluster()
+OpticsCluster(clusterPoints : Collection<OpticsClusterPoint<T>>)
+addClusterPoint(clusterPoint : T) : void
+getSize() : int
<<Interface>>
MetricSpacePoint
+getDistance(point : MetricSpacePoint) : double
<<Interface>>
ClusterPoint
T
+getClusterId() : int
+getData() : T
opticsOrdering
*
clusterPoints
*
setOfObjects
*
OpticsClusterPoint
T : MetricSpacePoint
+UNDEFINED : double = Double.MAX_VALUE
+NOISE : int = 0
<<Property>> -processed : boolean
<<Property>> -clusterId : int
<<Property>> -coreDistance : double
<<Property>> -reachabilityDistance : double
<<Property>> -data : T
+OpticsClusterPoint(data : T)
+isProcessed() : boolean
+getDistance(point : MetricSpacePoint) : double

Figure 1.3: Clustering package

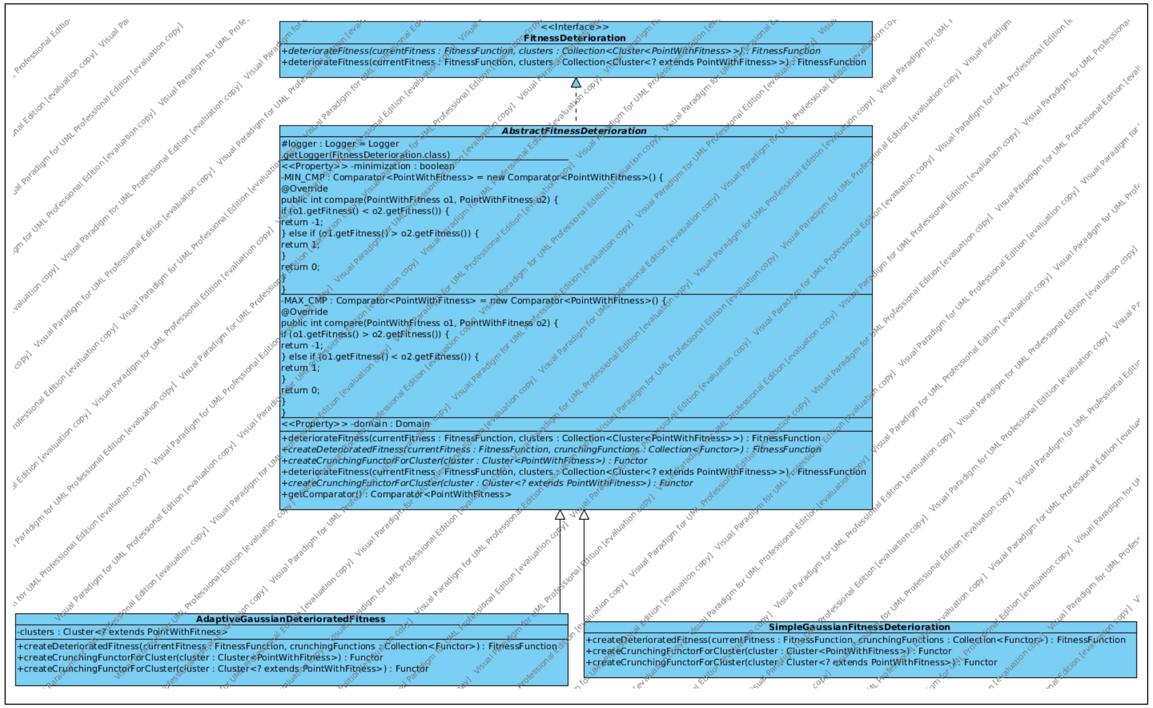

<<Interface>>
FitnessDeterioration
+deteriorateFitness(currentFitness : FitnessFunction, clusters : Collection<Cluster<PointWithFitness>>) : FitnessFunction
+deteriorateFitness(currentFitness : FitnessFunction, clusters : Collection<Cluster<? extends PointWithFitness>>) : FitnessFunction
AbstractFitnessDeterioration
#logger : Logger = Logger
.getLogger(FitnessDeterioration.class)
<<Property>> -minimization : boolean
-MIN_CMP : Comparator<PointWithFitness> = new Comparator<PointWithFitness>() {
@Override
public int compare(PointWithFitness o1, PointWithFitness o2) {
if (o1.getFitness() < o2.getFitness()) {
return -1;
} else if (o1.getFitness() > o2.getFitness()) {
return 1;
}
return 0;
}
}
-MAX_CMP : Comparator<PointWithFitness> = new Comparator<PointWithFitness>() {
@Override
public int compare(PointWithFitness o1, PointWithFitness o2) {
if (o1.getFitness() > o2.getFitness()) {
return -1;
} else if (o1.getFitness() < o2.getFitness()) {
return 1;
}
return 0;
}
}
<<Property>> -domain : Domain
+deteriorateFitness(currentFitness : FitnessFunction, clusters : Collection<Cluster<PointWithFitness>>) : FitnessFunction
+createDeterioratedFitness(currentFitness : FitnessFunction, crunchingFunctions : Collection<Functor>) : FitnessFunction
+createCrunchingFunctorForCluster(cluster : Cluster<PointWithFitness>) : Functor
+deteriorateFitness(currentFitness : FitnessFunction, clusters : Collection<Cluster<? extends PointWithFitness>>) : FitnessFunction
+createCrunchingFunctorForCluster(cluster : Cluster<? extends PointWithFitness>) : Functor
+getComparator() : Comparator<PointWithFitness>
AdaptiveGaussianDeterioratedFitness
-clusters : Cluster<? extends PointWithFitness>
+createDeterioratedFitness(currentFitness : FitnessFunction, crunchingFunctions : Collection<Functor>) : FitnessFunction
+createCrunchingFunctorForCluster(cluster : Cluster<PointWithFitness>) : Functor
+createCrunchingFunctorForCluster(cluster : Cluster<? extends PointWithFitness>) : Functor
SimpleGaussianFitnessDeterioration
+createDeterioratedFitness(currentFitness : FitnessFunction, crunchingFunctions : Collection<Functor>) : FitnessFunction
+createCrunchingFunctorForCluster(cluster : Cluster<PointWithFitness>) : Functor
+createCrunchingFunctorForCluster(cluster : Cluster<? extends PointWithFitness>) : Functor

Figure 1.4: Fitness deterioration package

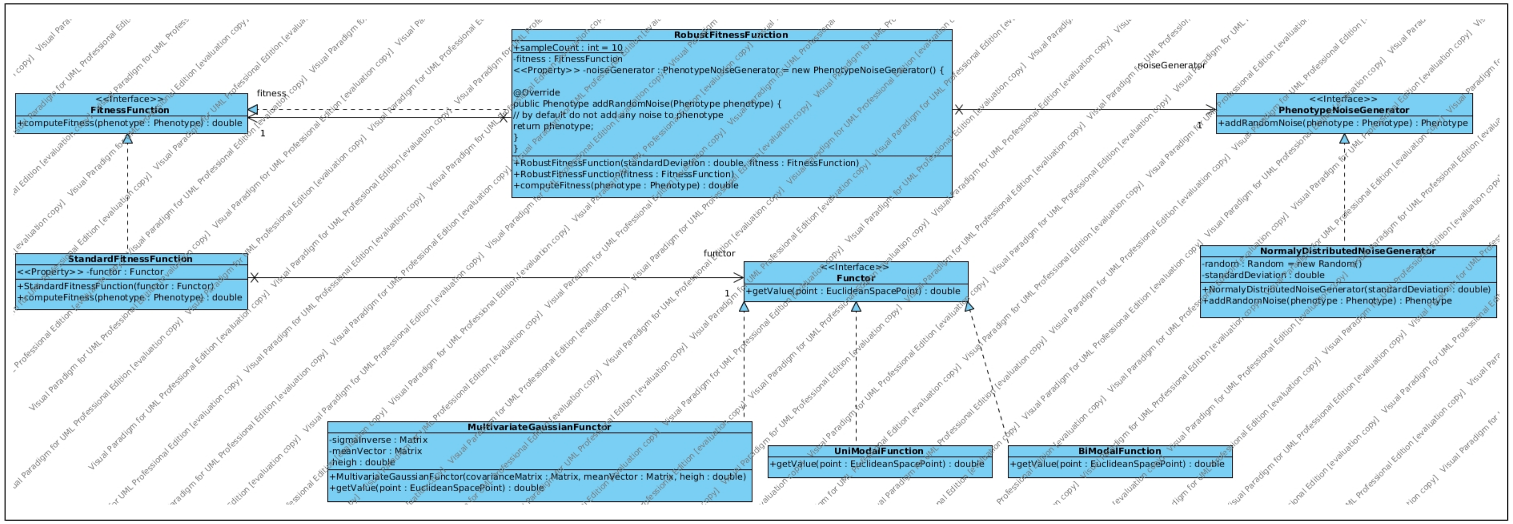

<<Interface>>
FitnessFunction
+computeFitness(phenotype : Phenotype) : double
fitness
1
RobustFitnessFunction
+sampleCount : int = 10
-fitness : FitnessFunction
<<Property>> -noiseGenerator : PhenotypeNoiseGenerator = new PhenotypeNoiseGenerator() {
@Override
public Phenotype addRandomNoise(Phenotype phenotype) {
// by default do not add any noise to phenotype
return phenotype;
}
}
+RobustFitnessFunction(standardDeviation : double, fitness : FitnessFunction)
+RobustFitnessFunction(fitness : FitnessFunction)
+computeFitness(phenotype : Phenotype) : double
noiseGenerator
1
<<Interface>>
PhenotypeNoiseGenerator
+addRandomNoise(phenotype : Phenotype) : Phenotype
StandardFitnessFunction
<<Property>> -functor : Functor
+StandardFitnessFunction(functor : Functor)
+computeFitness(phenotype : Phenotype) : double
functor
1
<<Interface>>
Functor
+getValue(point : EuclideanSpacePoint) : double
NormalyDistributedNoiseGenerator
-random : Random = new Random()
-standardDeviation : double
+NormalyDistributedNoiseGenerator(standardDeviation : double)
+addRandomNoise(phenotype : Phenotype) : Phenotype
MultivariateGaussianFunctor
-sigmaInverse : Matrix
-meanVector : Matrix
-heigh : double
+MultivariateGaussianFunctor(covarianceMatrix : Matrix, meanVector : Matrix, heigh : double)
+getValue(point : EuclideanSpacePoint) : double
UniModalFunction
+getValue(point : EuclideanSpacePoint) : double
BiModalFunction
+getValue(point : EuclideanSpacePoint) : double

Figure 1.5: Fitness and functors package

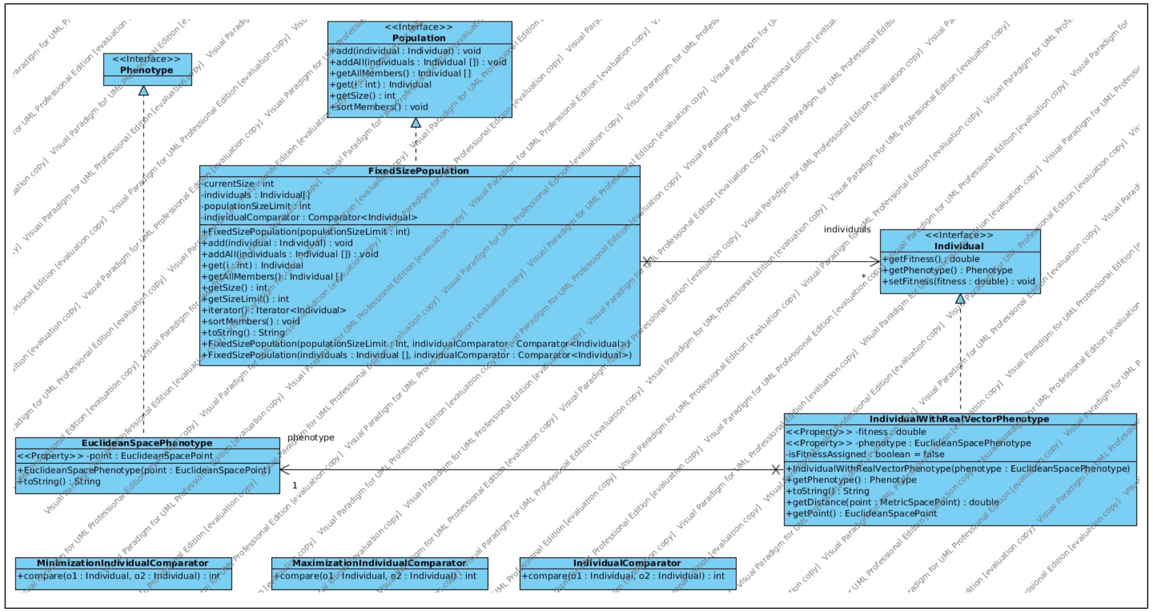

<<Interface>>
Phenotype
<<Interface>>
Population
+add(individual : Individual) : void
+addAll(individuals : Individual []) : void
+getAllMembers() : Individual []
+get(i : int) : Individual
+getSize() : int
+sortMembers() : void
FixedSizePopulation
-currentSize : int
-individuals : Individual[]
-populationSizeLimit : int
-individualComparator : Comparator<Individual>
+FixedSizePopulation(populationSizeLimit : int)
+add(individual : Individual) : void
+addAll(individuals : Individual []) : void
+get(i : int) : Individual
+getAllMembers() : Individual []
+getSize() : int
+getSizeLimit() : int
+iterator() : Iterator<Individual>
+sortMembers() : void
+toString() : String
+FixedSizePopulation(populationSizeLimit : int, individualComparator : Comparator<Individual>)
+FixedSizePopulation(individuals : Individual [], individualComparator : Comparator<Individual>)
individuals
*
<<Interface>>
Individual
+getFitness() : double
+getPhenotype() : Phenotype
+setFitness(fitness : double) : void
EuclideanSpacePhenotype
<<Property>> -point : EuclideanSpacePoint
+EuclideanSpacePhenotype(point : EuclideanSpacePoint)
+toString() : String
phenotype
1
IndividualWithRealVectorPhenotype
<<Property>> -fitness : double
<<Property>> -phenotype : EuclideanSpacePhenotype
-isFitnessAssigned : boolean = false
+IndividualWithRealVectorPhenotype(phenotype : EuclideanSpacePhenotype)
+getPhenotype() : Phenotype
+toString() : String
+getDistance(point : MetricSpacePoint) : double
+getPoint() : EuclideanSpacePoint
MinimizationIndividualComparator
+compare(o1 : Individual, o2 : Individual) : int
MaximizationIndividualComparator
+compare(o1 : Individual, o2 : Individual) : int
IndividualComparator
+compare(o1 : Individual, o2 : Individual) : int
Visual Paradigm for UML Professional Edition [evaluation copy]

Figure 1.6: Population package