SOC_LAB5_ GENETIC ALGORITHM

In this project you'll find the source code of a project that uses GENETIC ALGORITHM to calculate the best composition of Services or Web Services based on quality attributes such as Response time or Performance, Availability, Reliability and Cost. The combination of these 4 attributes define the QoS (Quality of Service) of each Service and of the entire Workflow. Of these 4 attributes, Reliability and Availability have a direct relationship with the QoS of a Service while Response time and Cost have an inverse relationship with QoS.

Instructions

------ setup and database connection ----------

1. Extract everything and copy the project to a folder and load it into intellij.

2. Edit or replace the Word file with the Data located in the data folder.

3. Load dependencies using SBT and IntelliJ.

4. The application can be run as a console application (genetic / console.java) or by running the application as a Play project. Both will output the best set of services for the Workflow.

------- HOW TO USE THE APPLICATION ------------

After running the application you will be able to access the API at
http://localhost:9000/api/web-service-composition or you can run the console application.

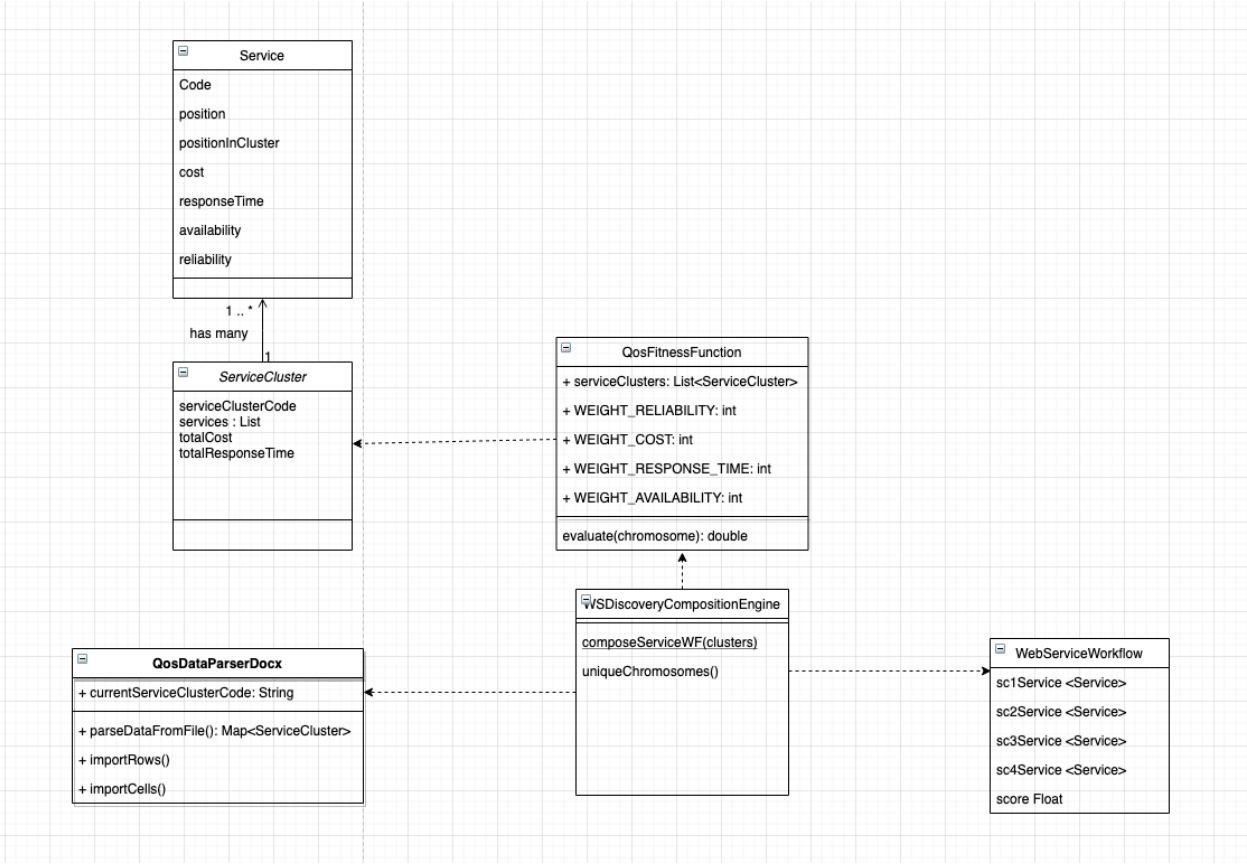
Project Structure

2. This code is organized in 5 packages (Controllers, Genetic.engine, Genetic.fitnessFunctions, Genetic.models and Genetic.Parsers).

- **Controllers** contains the Controllers needed to handle API calls.
- **Genetic.engine** contains the Workflow composition engine that uses JGAP and a Class to handle the outcome of the algorithm.

- **Genetic.fitnessFunctions** contains the fitness function class used in this project.
- **Models.** Contains the classes used to manage Service Clusters and Services.
- **Parsers.** Contains a Class to manage data from Word files using Apache Poi and an abstract class that has some common behavior and functions that must be implemented by this and other parsers.

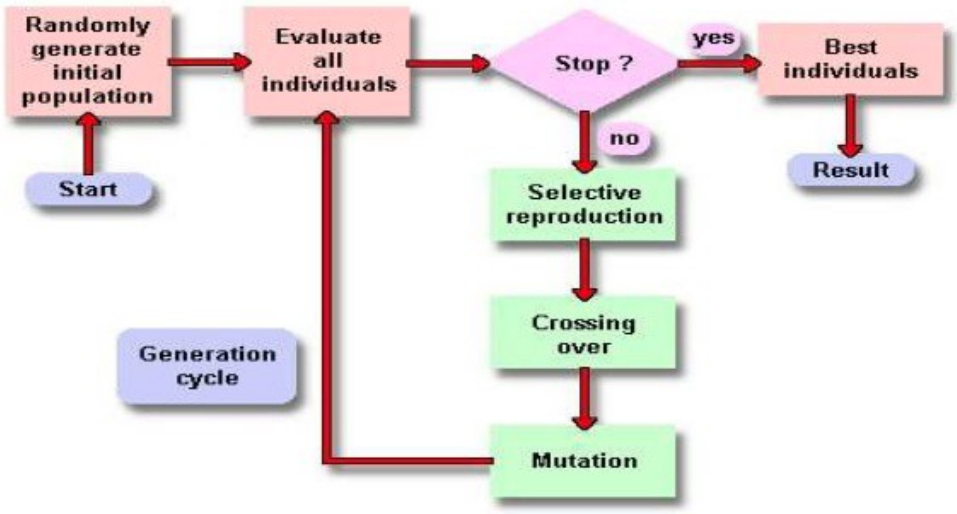
The main classes of the project can be seen in the following class diagram:



What is Genetic Algorithm and how it Works

Genetic Algorithm is an algorithm that imitates the concept of natural selection and evolution, where nature selects the best fitted organism and improves them through their childs using mutation and crossover of genes. For this algorithm to work developers must do two things. First they must translate the problem to Genes and Chromosomes (Individuals and characteristics, Solutions and components) and second they must create a mechanism or function that evaluates if a chromosome and its genes make the most fit organism or option that should continue in the evolution process.

The process of how Genetic Algorithm works can be seen in the following diagram:



https://becominghuman.ai/my-new-genetic-algorithm-for-time-series-f7f0df31343d

Our problem and how to translate it to GA

In the case of a list of Service Clusters and the composition of a Workflow that uses a single Service from each Cluster, we can translate this scenario to chromosomes and genes like this.

1. A chromosome refers to a set of Services that compose a flow. This means that Chromosomes refer to all the possible combinations that we can create for this flow by selecting a single Service from each cluster.

2. Chromosomes have Genes or properties. Likewise, our flow has 3 properties or Service Clusters that can vary from Chromosome to Chromosome. Therefore, in this problem a Gene represents the Service selected from a single Service Cluster.

3. Also, in Genetics, for every Gene we have Alleles, or variants of a Gene, which in this case are all the possible values that a Service Cluster can have based on the Services available in it.

The Fitness of each chromosome depends on its properties which in this case refers to the combination of the 4 quality attributes mentioned before. Also the fitness is impacted by the flow since we have some Clusters or Services that depend on others and their outcome before they can act.

Implementation of a Fitness Function

For my fitness function I divided the entire flow in multiple components or sequences and calculated the formulas for the 4 quality attributes mentioned before.

Sequence 1. From S1 to S2 … I'll refer to this sequence as W

- cost (W) = cost(S1) + cost(S2) //cost
- rt (W) = rt(S1) + rt(S2) // response time
- av(W) = min(av(S1) , av(S2)) // availability
- rel(W) = min(rel(S1) , rel(S2)) // reliability

Sequence 2. From W and S1 to S2 (Joint Parallel) …. I'll refer to this as Y

- cost (Y) = cost(W) //cost … only W since W includes S1
  cost (Y) = cost(S1) + cost(S2)
- rt (Y) = max(rt(W) , rt(S1)) // response time
  rt (Y) = rt(S1) + rt(S2)
- av(Y) = av(W) x av(S1) // availability
  av(Y) = min(av(S1) , av(S2)) x av(S1)
- rel(Y) = rel(W) x rel(S1) // reliability
  rel(Y) = min(rel(S1) , rel(S2)) x rel(S1)

Sequence 3. From Y to S3 (Sequential) … I'll refer to this as Z

- cost (Z) = cost(Y) + cost(S3) //cost
  cost (Z) = cost(S1) + cost(S2) + cost(S3)

- rt (Z) = rt(Y) + rt(S3) // response time
  rt (Z) = rt(S1) + rt(S2) + rt(S3)

- av(Z) = min(av(Y) , av(S3)) // availability
  av(Z) = min(min(av(S1) , av(S2)) x av(S1), av(S3))

- rel(Z) = min(rel(Y) , rel(S3)) // reliability
  rel(Z) = min(min(rel(S1) , rel(S2)) x rel(S1), rel(S3))

This fitness function is implemented inside of the class **QosFitnessFunction** located inside of **Genetic.FitnessFunctions** package. This class extends from the Fitness Functions Classes available in **jgap**.

The actual implementation of the fitness function mentioned before can be seen inside of the method or function **CalculateQoS.** This function does the calculation but before it normalizes cost and response time to a scale from 0 to 1 by dividing the selected service with the total metric (rt or cost) from the members of the Cluster that it belongs to. This total are stored during the import data process (**QosDataParserDocx.java**).

```
public float calculateQoS(Service s1, Service s2, Service s3, ServiceCluster sc1, ServiceCluster sc2, ServiceCluster sc3) {

    //1. calculate flow cost and normalize it
    float cost = s1.getCost() / sc1.getTotalCost() + s2.getCost() / sc2.getTotalCost() + s3.getCost() / sc3.getTotalCost();
    //2. calculate flow response time and normalize it
    float responseTime = s1.getResponseTime() / sc1.getTotalResponseTime() + s2.getResponseTime() / sc2.getTotalResponseTime() + s3.getResponseTime() / sc3.getTotalResponseTime();
    //3. Calculate reliability of flow
    float reliability = this.calculateReliability(s1.getReliability() , s2.getReliability() , s3.getReliability());
    //4. Calculate availability of flow
    float availability = this.calculateAvailability(s1.getAvailability() , s2.getAvailability() , s3.getAvailability());

    //5. Calculate QoS using the weight of each attribute. Cost and response time have an inverse effect, this means that the lower the better
    return (1 - cost) * QosFitnessFunction.WEIGHT_COST / 100 + (1 - responseTime) * QosFitnessFunction.WEIGHT_RESPONSE_TIME / 100 + reliability
            * QosFitnessFunction.WEIGHT_RELIABILITY / 100 + availability * QosFitnessFunction.WEIGHT_AVAILABILITY / 100;
}
```

The actual calculation of Reliability and Availability can be found in the following functions:

```
     * @return
     */
    private float calculateReliability(float rel1, float rel2, float rel3){
        return Math.min(rel3, Math.min(rel1, rel2) * rel1);
    }

    /**
     *
     * @param av1
     * @param av2
     * @param av3
     * @return
     */
    private float calculateAvailability(float av1, float av2, float av3){
        return Math.min(av3, Math.min(av1, av2) * av2);
    }
```

Executing and optimizing the Genetic Algorithm

Instead of using 16 bits to represent all our possible options I decided to use an alternative approach using the . What I did was I used the Integer Gene in three genes and limit each of those to the number of Services available for each Cluster. Then when a gene is selected I would use the number as to reference the selected service by its index in the list of Services available in each Service Cluster. This limits the number of options to just the 120 options available instead of the 256 options available when using 8 bits.

```
ServiceCluster sc1 = serviceClusters.get("SC1");
ServiceCluster sc2 = serviceClusters.get("SC2");
ServiceCluster sc3 = serviceClusters.get("SC3");

Gene[] sampleGenes = new Gene[3];
sampleGenes[0] = new IntegerGene(conf, 0, sc1.getServices().size() - 1); // SC1
sampleGenes[1] = new IntegerGene(conf, 0, sc2.getServices().size() - 1); // SC2
sampleGenes[2] = new IntegerGene(conf, 0, sc3.getServices().size() - 1); // SC3
```

You can see this process inside of **WSDiscoveryCompositionEngine.java.**

This decision allows the algorithm to run fast with a population size of 20.

```
S14_S22_S33 = 0.4370543360710144
S11_S22_S33 = 0.7015547752380371
S11_S22_S34 = 0.682857871055603
S11_S22_S33 = 0.7015547752380371
Total evolution time: 110 ms
Best Score: 0.7015547752380371


S11->S22->S33
Score = 0.7015547752380371
```

REST API

The project has a single API call

http:localhost:9000/api/web-service-composition

The output of this API route will be like this


```
{
    "sc1Service": {
        "code": "S11",
        "serviceClusterCode": "SC1"
    },
    "sc2Service": {
        "code": "S22",
        "serviceClusterCode": "SC2"
    },
    "sc3Service": {
        "code": "S33",
        "serviceClusterCode": "SC3"
    },
    "score": 0.7015547752380371
}
```


****

IF YOU ARE NOT ABLE TO RUN THIS PROJECT PLEASE HAVE A LOOK AT THE SCREENSHOTS FOLDER OR ASK ME ,
ALSO YOU CAN ASK FOR ACCESS TO THE REPOSITORY

****