Deliverable D3.6

Final profiling and classification techniques

Editor(s):	María José López
Responsible Partner:	TECNALIA
Status-Version:	Final version v1.0
Date:	28/05/2019
Distribution level (CO, PU):	PU

Project Number:	GA 731533
Project Title:	DECIDE

Title of Deliverable:	Final profiling and classification techniques
Due Date of Delivery to the EC:	31/05/2019

Workpackage responsible for the Deliverable:	WP3 - Continuous Architecting
Editor(s):	TECNALIA
Contributor(s):	Maria José López (TECNALIA)
Reviewer(s):	Kyriakos Stefanidis (FhG)
Approved by:	All Partners
Recommended/mandatory readers:	WP3, WP4, WP5

Abstract:	This deliverable comprises the final structures for repsenting the Simulation process and for adapting the Algorithm to the DECIDE OPTIMUS Simulation process.
Keyword List:	Modelling, classification, profiling, algorithm, genetic
Licensing information:	This work is licensed under Creative Commons Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0) http://creativecommons.org/licenses/by-sa/3.0/
Disclaimer	This document reflects only the author's views and the Commission is not responsible for any use that may be made of the information contained therein

Document Description

Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	21/05/2019	First draft with all the sections	TECNALIA
v0.2	27/05/2019	Internal review	FHG
V0.3	28/05/2019	Modifications due to the internal review with comments	TECNALIA
V1.0	28/05/2019	Final version – Ready for submission	TECNALIA

Table of Contents

List of Figures

Terms and abbreviations

AD	Application Description
CS	cloud service
IP	Internet Protocol
JSON	JavaScript Object Notation
NFR	Non-Functional Requirement
RAM	Random Access Memory
NSGA-II	Non-dominated Sorting Genetic Algorithm II
ACSmI	Advanced Cloud Service meta-Intermediator
ICT	Information and Communications Technologies
HW	Horton Works
CL	Cloudera
MR	MapR
CPU	Central Processing Unit

Executive Summary

This document describes the Simulation process that is implemented in the DECIDE OPTIMUS tool in its final version. The Selection of the Algorithm used is also explained as well as the adaptations made to cover the OPTIMUS problem.

The data managed through the DECIDE workflow is stored in the DECIDE.json file and in databases owned by the different tools. The models for the data structure of Cloud Services, Deployment Schemas and microservices are explained in the previous deliverable D3.5 Intermediate profiling and classification techniques [1], as well as the classification process that has not changed since its last version [1].

The last version of the properties of the application description (AD) can be found in the document D2.2 Detailed requirements specifications v2 [2].

The OPTIMUS classification tool process is basically the same that was described in the previous deliverable [1], which its implementation is described in the document D3.9 Final DECIDE OPTIMUS [3]

This deliverable starts, in section 2, describing the genetic algorithm NSGA-II, used for optimizing the Simulation process, taking into account the porperties of the application and the Cloud Services available.

The decision of using that genetic algorithm has been taken considering previous experience and work done in the topic by the OPTIMA area in TECNALIA which is published under the title of “A heuristic approach to the multicriteria design of IaaS cloud infrastructures for Big Data applications” [4]. The objective of this paper is related to the same problem context that DECIDE faces, thus, it was deemed that the best option was to adapt the method but changing those elements that are specific to the DECIDE problem.

The process where the algorithm is integrated is described in section 3, explaining the points where the optimization is done once the Application Description is completed by the developer. Before running the algorithm, some structures have to be created to facilitate the access to the information by the algorithm, as it will be explained in sections 4 and 5. Section 4 is about the configuration of the DECIDE Simulation problem, while section 5 contains the details for undestanding how the solutions are obtained.

Both prolem and solution are part of the specific DECIDE OPTIMUS Simulation problem, that has been adjusted to run the algorithm using the java library MOEA Framework [5], following a proper configuration by the authors.

1 Introduction

1.1 About this deliverable

This document is the final version of the "Profiling and classification techniques", deliverable included in the work produced by WP3, Continuous Architecting. The objective is to describe the combined and optimized Simulation process followed for obtain the five best Deployment Schemas for a specific multi-cloud application. The core of this process is based on the NGSA-II genetic Algorithm.

The previous documents describe the representation of the information about the multi-cloud application managed by the DECIDE ecosystem and the data existing in the ACSmI Discovery Registry about the Cloud Services. The models about those elements have not had significant changes since the last versions. These last versions can be found in the corresponding DECIDE documents D2.2 Detailed requirements specifications v2 [2] and D5.4 Final Advanced Cloud Service meta-Intermediator (ACSmI) [6]

1.2 Document structure

This document is organized in 4 sections.

Section 2 consists on a general description of the NGSA-II algorithm and the explanation of the work gathered in the paper "A heuristic approach to the multicriteria design of IaaS cloud infrastructures for Big Data applications". Section 3 explains the whole process where the algorithm is implemented and the preparation of the information for running the algorithm. Sections 4 and 5 are devoted to present the configuration needed to adapt the problem that the library takes as input to the DECIDE OPTIMUS simulation problem, and how the solutions are interpreted and shown as the five bests deployment schemas. Finally, section 6 presents the conclusions and comments for the future steps until the end of the project.

2 NGSA-II algorithm

Genetic algorithms are methods for solving optimization problems, based on natural selection, reproducing the biological evolution way. The process basically consists on several modifications of a initial population of individual solutions, selecting from it randomly individuals at each step to be parents and using them to produce children for the next generation.

Over successive modifications the population evolves toward an optimal solution [7].

The problem existing in the DECIDE framework consists on having a list of microservices and a list of Cloud Services, with the challenge of obtaining the best combination of Cloud Services where to deploy the multi-cloud application. The solution has to be optimized and the elements that are part of it has to be combined following a method depending on that optimization of the solution, modifying them in a way in which the objectives are being maximized.

The algorithm selected to solve this challenge is the NGSA-II. It is reckoned as "*one of the most popular multi objective optimization algorithms with three special characteristics, fast non-dominated sorting approach, fast crowded distance estimation procedure and simple crowded comparison operator*" [8].

The process that these algorithms follow is:

1. Initialize: Create an initial population, randomly generated following the established configuration of the problem element.
2. Evaluate: Once the initial population is created, each of its members is evaluated regarding the requirements for the optimization.
3. Select: Not all solutions meet the requirements, and only the best elements of the population are kept. The fitness value for each individual determines if it is part of the next generation of the solutions.
4. Crossover and mutation: Creation of new individuals by combining aspects of the selected individuals expecting to improve them. This creation is made altogether with small random changes because if there are no modifications, the combination of the initial population will be in the initial population.
5. And then repeat the process from step 2, until the objectives are reached, or a predefined condition occurs.

The "OPTIMA" business area of TECNALIA, belonging to the ICT Division where this DECIDE project is being implemeted, is dedicated to big data and optimization algorithms for a range of sectors and applications. Team members of this group published a paper "A heuristic approach to the multicriteria design of IaaS cloud infrastructures for Big Data applications" that is taken as baseline for the DECIDE OPTIMUS optimization algorithm, while adapting it to the DECIDE problem context, and more specifically, to the simulation – related activities [4].

In the above-mentioned paper, the authors present a method in which they try to solve the problem of optimizing the definition of IaaS cloud models for hosting Big Data Platforms, considering the requirements of cost, reliability and computing capacity.

The platforms used as elements to optimize are some existing ones from several providers, and the optimization consists on how to configure them in certain aspects such as number of master nodes for managing the Big Data cluster state, number of worker nodes to store data and process them, RAM available in every node (master or worker), number of available CPU cores for each instance (master or worker), storage capacity (master or worker), operatng system and level of replications.

Althoug the solution of the DECIDE OPTIMUS simulation problem can follow the same method as the one described in the paper, some elements have to be modified and adapted. The solution is not a

predefined platform and neither exists a unique provider for the whole deployment schema. The solution consists on an ad-hoc platform or Schema containing Cloud Services instances for different providers and not dedicated to manage Big Data applications exclusively.

The experiment related in the paper was developed using the “inspyred” library for python [9].

In that case, the problem consists on a solution space with this format:

[platform, cloud, region, numNodosMaster, numNodosWorker, ramMaster, ramWorker, cpuMaster, cpuWorker, storageMaster, storageWorker, so, factorReplicacion]

In that example, each element has a numeric value meaning the index of the different values that can take it. For example, the platform element can vary with this set of values {0:'HW',1:'CL',2:'MR'}. The initialization for this space is randomly performed following some rules for each element.

The algorithm evolves that solution, following the established criteria and matching it with the best until that moment, storing it if it is better and mutating the solution according to three values that have to be optimized. These values are called fitness values, and work as “score” objective for the evaluating process.

These three fitnes values (objectives) are cost, computing capacity and reliability. Cost objective has to be minimized to consider it optimized, and the other two maximized.

When the python process ends, a possible solution can be the following:

[0, 1, 2, 65, 713, 10, 7, 11, 4, 50, 181, 0, 0.0687471581286122] : [0.17102518790591437, 0.06335078568746313, 0.5341062586145904]

Where:

platformType = {0:'HW',1:'CL',2:'MR'}

cloudType = {0:'Amazon',1:'Microsoft',2:'Google'}

regionType = {0:'Europa',1:'EEUU',2:'Asia'}

so = {0:'Unix', 1:'Windows'}

and the three objectives are cost: 0.17102518790591437, computing capacity: 0.06335078568746313 and reliability: 0.5341062586145904 for that infrastructure configuration.

The implementation also shows a Figure representing the estimated Pareto front including all the solutions.

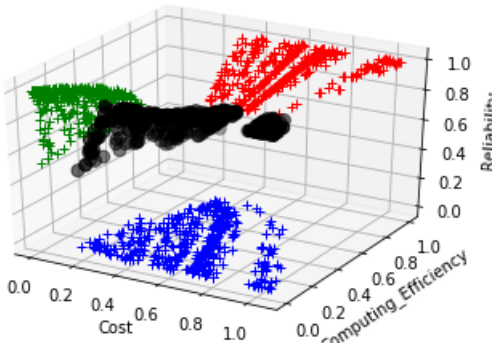


Figure 1. Estimated Pareto front by the set of archived solutions [4]

The adjustment to the algorithm for configuring the DECIDE OPTIMUS Simulation problem is basically made by modifying the "problem" element and the process to evaluate the solutions.

The main function that executes the algorithm calls the run function as follows:

```
NondominatedPopulation result = new Executor()
                            .withProblemClass(OPTIMUSproblem.class)
                            .withAlgorithm("NSGAII")
                            .withMaxEvaluations(10000)
                            .run();
```

That main function and the rest of problem configuration are implemented as part of the OPTIMUSproblem java class, and indicated as a parameter of the Executor element creation.

OPTIMUS has been developed in Java and for including the NGSA-II Algorithm, the libraries provided by this MOEAFramework [5] have been used.

3 Simulation process description

OPTIMUS is a module of the DECIDE system, and can be accessed through DevOps Framework or a standalone plugin. The information needed to run this module is done by the DECIDE.json, written by the previous tools in the Framework, such as the DevOps framework NFR Editor, and ARCHITECT.

Once the developer has defined the properties of the application and its requirements, including the microservices that compose the whole application and the Non Functional Requirements (NFRs), the DECIDE ecosystem is able to obtain the best possible deployment schema. The elements that are involved in this process are depicted in the Figure 2.

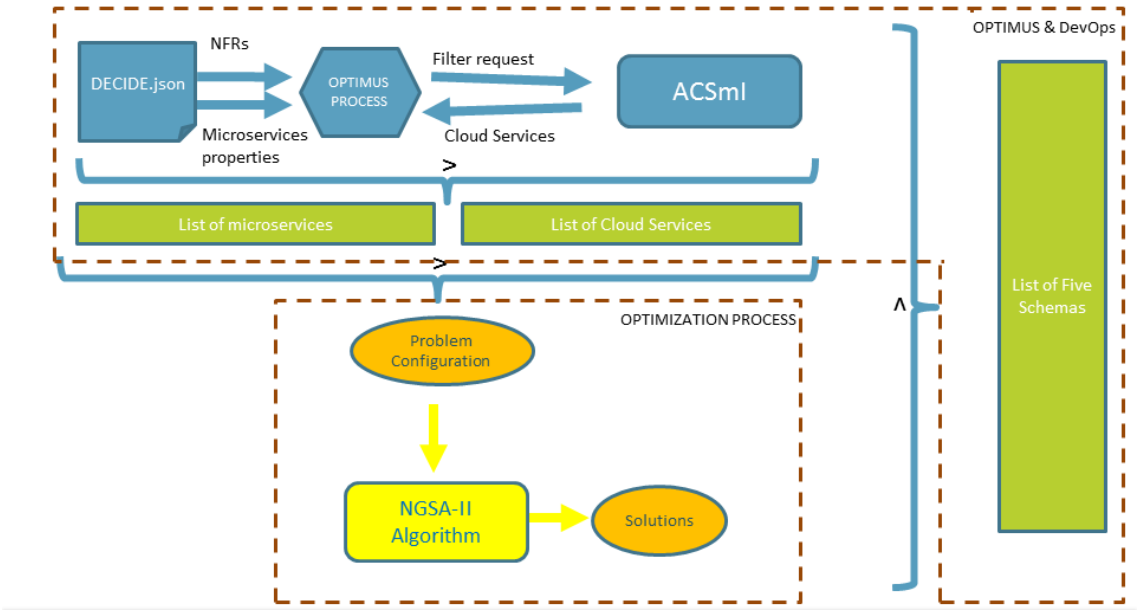


Figure 2. Elements of the Simulation Process.

The NFR Editor and the General Editor are part of the DevOps Framework which is responsible for taking the information from the developer and storing it in the DECIDE.json file. Following that, OPTIMUS plugin completes data about the microservices and their detachable resources.

The first step of OPTIMUS Simulation is to build a request for ACSmI (Discovery) in order to know which Cloud Services are available that fullfil the microservices NFRs and general properties such as disk capacity, memory and number of cores.

The system then has on one hand, the list of microservices with their requirements and on the other hand, the list of the Cloud Services that fulfil them. That information is all the information needed for deducing the best possible deployment schema, and with it, the problem element of the NSGA-II Algorithm is created.

The NSGA-II Algorithm searches through the different combinations of Cloud Services for the microservices and finds the 'perfect' combination of them which could result in a list of best solutions as they are built as part of the problem.

The elements that can be seen in the Figure 2 are:

- DECIDE.json: The file (Application Description) that contains all the information about the multicloud application. Every tool of the DECIDE ecosystem contributes to add properties to this file. Its detailed structure is described in the D2.2 [2].

- Filter request: OPTIMUS builds an object taking into account the classification of the microservice, the NFRs set and the characteristics informed by the developer. The format for that request is defined by the ACSmI Discovery java client module [10].

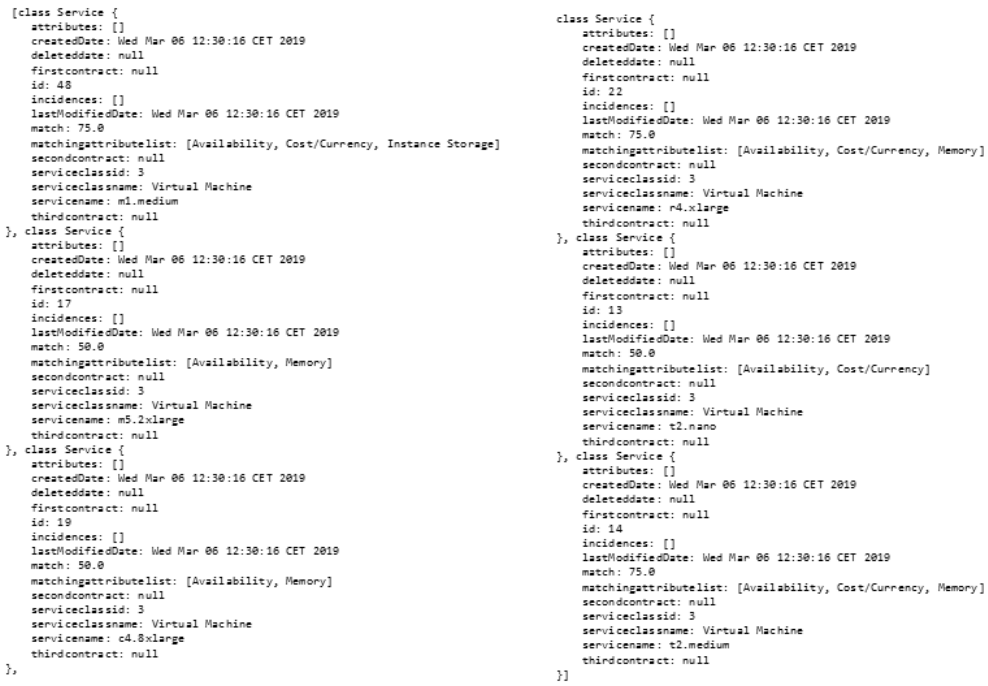

```
{
  "classid" : 3,
  "provider" : "Amazon",
  "incidences" : false,
  "attributes" : [ {
    "typeid" : 20,
    "childid" : 0,
    "name" : "Availability",
    "value" : "98.0",
    "unit" : "percentage"
  }, {
    "typeid" : 25,
    "childid" : 0,
    "name" : "Cost",
    "value" : "250.0",
    "unit" : "€"
  }, {
    "typeid" : 9,
    "childid" : 0,
    "name" : "Public IP",
    "value" : "IP",
    "unit" : null
  }, {
    "typeid" : 7,
    "childid" : 0,
    "name" : "Instance Storage",
    "value" : "200",
    "unit" : "GB"
  }, {
    "typeid" : 6,
    "childid" : 0,
    "name" : "Memory",
    "value" : "2",
    "unit" : "GB"
  } ]
}
```


Figure 3. Filter for requesting Cloud Services to ACSmI

The filter in Figure 3 requests a Virtual Machine (classid 3) provided by Amazon, with no incidences in the last month, and fulfilling some properties such as availability 98%, Cost no more than 250 Euros, with accessible IP, with a minimun of disk resources of 200 GB and Memory at least 2 GB.

- Cloud Services: List of Cloud Services returned by ACSmI (Discovery) when the request is performed.

Figure 4 shows the response for the request in Figure 3.



```
[class Service {
    attributes: []
    createdDate: Wed Mar 06 12:30:16 CET 2019
    deleteddate: null
    firstcontract: null
    id: 48
    incidences: []
    lastModifiedDate: Wed Mar 06 12:30:16 CET 2019
    match: 75.0
    matchingattributelist: [Availability, Cost/Currency, Instance Storage]
    secondcontract: null
    serviceclassid: 3
    serviceclassname: Virtual Machine
    servicename: m1.medium
    thirdcontract: null
}, class Service {
    attributes: []
    createdDate: Wed Mar 06 12:30:16 CET 2019
    deleteddate: null
    firstcontract: null
    id: 17
    incidences: []
    lastModifiedDate: Wed Mar 06 12:30:16 CET 2019
    match: 50.0
    matchingattributelist: [Availability, Memory]
    secondcontract: null
    serviceclassid: 3
    serviceclassname: Virtual Machine
    servicename: m5.2xlarge
    thirdcontract: null
}, class Service {
    attributes: []
    createdDate: Wed Mar 06 12:30:16 CET 2019
    deleteddate: null
    firstcontract: null
    id: 19
    incidences: []
    lastModifiedDate: Wed Mar 06 12:30:16 CET 2019
    match: 50.0
    matchingattributelist: [Availability, Memory]
    secondcontract: null
    serviceclassid: 3
    serviceclassname: Virtual Machine
    servicename: c4.8xlarge
    thirdcontract: null
},
```

```
class Service {
    attributes: []
    createdDate: Wed Mar 06 12:30:16 CET 2019
    deleteddate: null
    firstcontract: null
    id: 22
    incidences: []
    lastModifiedDate: Wed Mar 06 12:30:16 CET 2019
    match: 75.0
    matchingattributelist: [Availability, Cost/Currency, Memory]
    secondcontract: null
    serviceclassid: 3
    serviceclassname: Virtual Machine
    servicename: r4.xlarge
    thirdcontract: null
}, class Service {
    attributes: []
    createdDate: Wed Mar 06 12:30:16 CET 2019
    deleteddate: null
    firstcontract: null
    id: 13
    incidences: []
    lastModifiedDate: Wed Mar 06 12:30:16 CET 2019
    match: 50.0
    matchingattributelist: [Availability, Cost/Currency]
    secondcontract: null
    serviceclassid: 3
    serviceclassname: Virtual Machine
    servicename: t2.nano
    thirdcontract: null
}, class Service {
    attributes: []
    createdDate: Wed Mar 06 12:30:16 CET 2019
    deleteddate: null
    firstcontract: null
    id: 14
    incidences: []
    lastModifiedDate: Wed Mar 06 12:30:16 CET 2019
    match: 75.0
    matchingattributelist: [Availability, Cost/Currency, Memory]
    secondcontract: null
    serviceclassid: 3
    serviceclassname: Virtual Machine
    servicename: t2.medium
    thirdcontract: null
}]
```


Figure 4. ACSmI response

The response is a list of Cloud Services, identified by an id property. It has 6 elements which means that 6 Cloud Services in the ACSmI Discovery Registry fulfil total or partially the requirements.

For example, the Cloud Service with id 48, has a match of 75% of the requirements, and they are Availability, Cost, and Instance Storage. The only property that is not fulfilled is Memory.

In this case, there is no Cloud Service in the Registry that is 100% suitable for the microservice, and the best deployment is only appropriate at 75%.

However, the best deployment schema is not an isolated Cloud Service for a unique microservice. OPTIMUS has to deduce the best combination of Cloud Service for the set of microservices that constitute the multi cloud application. For obtaining that optimized combination of elements, the NGSA-II algorithm is used. To this end, the following info is needed:

- List of microservices: Gathering the information that OPTIMUS has about the microservices, there is a list with a record for each microservice and consisting of the id of the microservice, the NFRs associated to it, and the characteristics asked by the developer. This list is the information that is used to check how suitable is each of the solutions obtained from the NGSA-II, for building the Schemas.
- List of Cloud Services: That list is composed by the Clod Services obtained by requesting ACSmI for the elements that fulfil the requirements set by the microservices.

These two lists are the information used to build the problem element that the NGSA-II Algorithm manages in its optimization process. First, for the combination of the solutions space and then, for optimizing those solutions taking into account the objectives fulfiled. This process will be explained in the following sections.

- List of Five Schemas: DECIDE framework establishes that the developer selects the best deployment schema among five of them. The result of the algorithm has to be analyzed and shown to the developer including the Cloud Services involved for each microservice and the requirements fulfilled by it. The lists above mentioned are part of the information used.

The elements involved in the optimization process are described in the following sections.

4 Simulation problem

OPTIMUS Simulation module has been developed using the eclipse framework for java. The server where the simulation code is placed is a maven project, and the way to use that library is by including the dependency in the pom.xml file.

Each problem to solve needs to create a new class where that problem is defined and the rules for evaluating it are established. Then, the main part of the class calls to the principal element of the library, namely, the Executor.java.

With that call, the properties of that execution is set, and the algorithm knows the elements to use in the process. This is shown next:

```
NondominatedPopulation result = new Executor()
                                .withAlgorithm("NSGAII")
                                .withProblem("OPTIMUSproblem.class")
                                .withMaxEvaluations(10000)
                                .run();
```

The java file for defining the OPTIMUS Simulation problem is OPTIMUSproblem.class, and the configuration for it is:

- numberOfVariables = number of microservices of the multi-cloud application
- numberOfObjectives = for this problem two objectives have been considered.

The constructor for the problem will create an OPTIMUSproblem with two objectives and N variables, because the problem structure depends on the number of the microservices.

The result variable contains a list of solutions, that are created and initialized in the OTPIMUSproblem.class.

The space of solutions for this problem is:

[CS0, CS1, .. , CSN]

with different sizes of the solution each time the algorithm is called by the DECIDE framework.

These values are related to the Cloud Service where the solution proposes to deploy the corresponding microservice. For keeping the relationship between the microservices and the position in the solution, it is necessary to access the list of microservices from the OPTIMUS simulation process described in section 2.

List of microservices

This list contains the same information about the microservice as the Filter with which OPTIMUS asks ACSmI for the suitable Cloud Services.

Microservice-id	classification	List of constraints	List of NFRs

This list has a number of elements [0,..,N] being the first element of this list, the microservice to which is referred the first element of the space of solutions.

The other element that has to be accessed is the list of Cloud Services.

List of Cloud Services

The list of Cloud Services is the range of values that each element of the space of solutions can have.

ClodServiceInstance-id	class	List of attributes

Then, taking into account all this information, a solution can be explained as in the Figure 5.

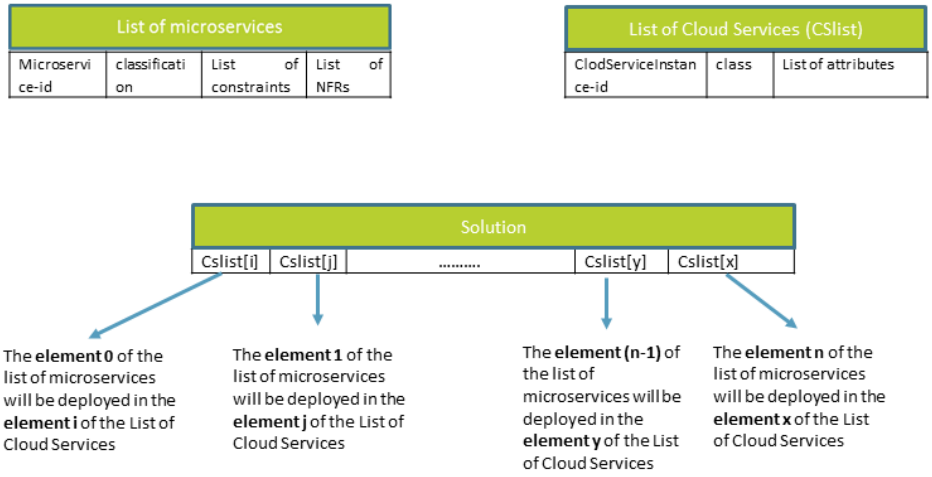


Figure 5. Space of Solutions - structure

5 Simulation Solutions

The evaluation function is implemented in the OPTIMUSproblem java class and assigns the fitness value for each solution. This function is called from the abstract class for the algorithm AbstractAlgorithm.java.

The evaluate function analyzes the input parameter (solution) assigning two objectives based on how suitable the Cloud Services are for deploying on them the microservices involved.

These two objectives are:

- f1: % of fulfilment of the NFRs required by the microservices.
- f2: % of fulfilment of the characteristics set by the developer for the microservices.

The value of fitness f1:

The NFRs managed in the DECIDE framework are Location, Availability, Cost, Performance and Legal Level.

Each of the microservices can have a specific related value for each NFR, and the method for obtaining a score for them varies depending on which NFR is being evaluated.

- Location: The exact Location has to match with the value provided by the Cloud Service, otherwise any point is added to the f1 value for this microservice.
- Availability: The availability that the Cloud Service provides has to be greater than the availability required by the microservice. For each microservice that has a match with the availability required, one point is added to the f1 value.
- Performance: Same behaviour as for Availability.
- Cost: The addition of the costs of the Cloud Service that are included in the solution has to be less than the addition of the individual costs of the microervices, or if the cost is set at application level, less than this global value of Cost.
- Legal Level: The legal level provided by each Cloud Service has to be less or equal than the one that is required for that microservice. Then one point is added to the f1 value.

The value of fitness f2:

The characteristics related to the microservices are the classification, public IP, disk space, RAM and number of cores.

- Classification: This value is set by OPTIMUS classification and it is used to select the Cloud Services classes that match with the microservices. For example, if the classification is “computing”, the Cloud Service class has to be Virtual machine or container. If the classification matches with the Cloud Servcie class, one point is added to the f2 value.
- Public IP: If the Cloud Service allows accessing to it by the users of the application and the microservice needs this public access, then the f1 value will be incremented.
- Disk Space: The sum of the disk space requirements of the microservices that are set to the same Cloud Service has to be less or equal to the disk space that the Cloud Service offers. Otherwise no points will be added.
- RAM: The same method as with the disk space. The sum of the RAM requirements of the microservices that are set to the same Cloud Service has to be less or equal to the Memory that the Cloud Service offers. Otherwise no points will be added.
- Number of cores: The sum of the cores required by the microservices that are set to the same Cloud Service has to be less or equal to the number that the Cloud Service offers. Otherwise no points will be added.

Both values f1 and f2 are maximized.

Once the five best solutions are returned by the algorithm, OPTIMUS restructures the information contained in them and accesses the lists that have been created with the problem in order to create a list of the five best Deployment Schemas.

The developer should then choose his/her favorite, taking into account the information regarding each of them, which consists on the requirements fulfiled for each microservice by the Cloud Services where it will be deployed.

The first Schema will be the best of all the solutions but the developer could choose another one depending on the prioritization that he or she makes for the NFRs or characteristics associated.

This selected Schema will be stored in the DECIDE.json as well as in the Schema History repository.

6 Conclusions

Since the models for representing the microservices and the Cloud Services have been described in the previous deliverable [1], the objective of this deliverable has been to describe the algorithm used for combining and optimizing the process of Simulation.

The algorithm selected is the NGSA-II with the configuration studied in the paper described in section 2. The decision has been taken based on the knowledge acquired about this area through the practical previous experience applying genetic algorithms to select IaaS Cloud Infrastructures for Big Data Applications. It can be considered another case study of the same work.

The objectives to maximize are related to the fulfilment of the microservices requirements, NFRs and specific characteristics, by the attributes provided for the Cloud Services that are part of the selected Deployment Schema.

In the previous deliverable we considered the possibility of exporting the application data included in the AD, onto a CAMEL model. This idea has not been implemented because it has not been considered as a general benefit for the users in the context of the DECIDE project, but rather a specific requirement that can be developed as an external, additional, feature.

Further tasks will be related to the integration of OPTIMUS in the DECIDE framework, and changes can be demanded in order to solve any potential issues arising from the integration tasks. However, the core of the models and the Simulation process are set and stable, and not prone to further changes.

7 References

[1] DECIDE Consortium, "D3.5 Intermediate profiing and classification techniques," Bilbao, 2018.

[2] DECIDE Consortium, "D2.2 Detailed requirements specifications v2," Madrid, 2018.

[3] DECIDE Consortium, "D3.9 Final DECIDE OPTIMUS," Bilbao, 2019.

[4] A. T. M. N. B. J. D. S. María Arostegi, "A heuristic approach to the multicriteria design of IaaS cloud infrastructures for Big Data applications," *Expert Systems,* 2018.

[5] D. H. a. o. contributors, "A Free and Open Source Java Framework for Multiobjective Optimization," 2019. [Online]. Available: https://github.com/MOEAFramework/MOEAFramework/tree/41b4a77076bfe83820ffb20598cb8f474e72e250. [Accessed April 2019].

[6] DECIDE Consortium, "D5.4 Final Advanced Cloud Service meta-Intermediator (ACSmI)," Bilbao, 2019.

[7] I. The MathWorks, "MathWorks," 2019. [Online]. Available: https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html. [Accessed May 2019].

[8] M. S. N. A. M. Z. Yusliza Yusoff*, "Overview of NSGA-II for Optimizing Machining Process Parameters," *Science Direct,* p. 6, 2011.

[9] A. Garrett, "inspyred: Bio-inspired Algorithms in Python," 2012. [Online]. Available: https://aarongarrett.github.io/inspyred/. [Accessed May 2019].

[10] DECIDE Consortium, "DECIDE repository," 2019. [Online]. Available: https://git.code.tecnalia.com/decide/WP5/tree/master/ACSmI_discovery/eu.decideh2020.acsmi.backend.services.client. [Accessed 2019].