Deliverable D3.8

Intermediate DECIDE OPTIMUS

Editor(s):	María José López
Responsible Partner:	TECNALIA
Status-Version:	Final – 1.0
Date:	28/11/2018
Distribution level (CO, PU):	CO

Project Number:	GA 726755
Project Title:	DECIDE

Title of Deliverable:	D3.8 Intermediate DECIDE OPTIMUS
Due Date of Delivery to the EC:	30/11/2018

Workpackage responsible for the Deliverable:	WP3 – Continuous Architecting
Editor(s):	TECNALIA
Contributor(s):	TECNALIA
Reviewer(s):	Javier Gavilanes (EXPERIS)
Approved by:	All Partners
Recommended/mandatory readers:	WP3, WP4, WP5

Abstract:	This software deliverable comprises the intermediate OPTIMUS simulation engine. This deliverable is the result of T3.2 and T3.3. The software will be accompanied by a Technical Specification Report
Keyword List:	Simulation, classification, Eclipse IDE, plugin, java
Licensing information:	This program and the accompanying materials are made available under the terms of the Eclipse Public License 2.0 which is available at https://www.eclipse.org/legal/epl-2.0/ The document itself is delivered as a description for the European Commission about the released software, so it is not public.
Disclaimer	This deliverable reflects only the author's views and 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	20/11/2018	First internal Draft version	TECNALIA
v0.2	21/11/2018	Version ready for internal review	TECNALIA
V0.3	28/11/2018	Implemented the changes suggested by the reviewer	TECNALIA
v.10	30/11/2018	Ready for submission	TECNALIA

Table of Contents

List of Figures

List of Tables

Terms and abbreviations

ACSmI	Advanced Cloud Service meta-Intermediator
CS	Cloud Service
DB	Database
EC	European Commission
UI	User Interface
JSON	JavaScript Object Notation
MCSLA	Multi Cloud Service Level Agreement
NFR	Non-Functional Requirements
VH	Violation Handler

Executive Summary

This document contains the technical description of the DECIDE OPTIMUS tool in its intermediate version. The DECIDE OPTIMUS M24 prototype provides the developer with the best possible theoretical deployment for his or her multi-cloud application based on the classification of the microservices that compose the application, the Non-Functional Requirements involved, and the cloud services handled by ACSmI (Discovery). The application classification is the first step of the OPTIMUS process [1].

Moreover, the document includes sections about how to install, and use DECIDE OPTIMUS, and the license under it is published.

The details about the functionality and the technical aspects are described in the corresponding sections as well as the manual and the instructions to run the software.

The general architecture and design of DECIDE OPTIMUS tool is described in this document due to the fact that there is no specific document for it. So, the evolution and the requirements fulfilled in each prototype can be found in the corresponding version of the deliverable. The general requirements and the functionalities can be consulted in the initial version of the document [2].

Next version of this document is the final one and will present the final status of the DECIDE OPTIMUS tool, the features that the DECIDE OPTIMUS tool will provide to the developer and the technical characteristics associated to it. This version is planned for being ready in the month 30.

1 Introduction

1.1 About this deliverable

This document presents the functionalities, design and development of the DECIDE OPTIMUS tool in its intermediate version, as a prototype. The content is the corresponding one to the current version of prototype planned for month 24.

1.2 Document structure

The global architecture and the functionalities covered by this M24 prototype, as well as all the instructions for installing the software and using it, are described in this deliverable.

This document is composed of four (4) main sections:

1. General introduction about the content and structure of the document.
2. The implementation of the DECIDE OPTIMUS, including the eclipse plugin and the REST service generated with Swagger [3]. The requirements that the two parts cover, the architecture and the description of its main components. Also, the technical development aspects and how OPTIMUS fits in the global DECIDE architecture are included.
3. The delivery and usage section, about how to install it, how to use it and any licensing information to run the prototype.
4. Conclusions and future work

.

2 Implementation

2.1 Functional description

DECIDE OPTIMUS will provide the best possible application deployment schemas, based on the non-functional requirements set by the developer and the requirements of the multi-cloud application, automating the provisioning and selection of cloud services offering for multi-cloud applications.

The DECIDE OPTIMUS tool consists of two separate parts, the first one covers the classification process through a local eclipse plugin installation and the second one can be invoked by that plugin or by the DECIDE Framework as it is a REST [4] service by which the simulation process can be launched.

Functionalities:

The main functionalities of the DECIDE OPTIMUS covered in this M24 prototype are:

1. **Multi-cloud application classification**. This functionality consists in associating the components (microservices) that form the multi-cloud application to a group of Cloud Services where they could be deployed through a classification type.

   For this purpose, the profiling of the microservices of the multi-cloud applications is a way to match the characteristics of those microservices with the group of Cloud Services features where they will be deployed.

   This classification will be based on the information provided by the developer and the information handled by "Types management" subcomponent.

   This functionality is partially covered considering that the final information requested to the developer will increase as the simulation process becomes more complex.

   The current prototype presents the UI for introducing details about the application and the microservices, such as the name of the multi-cloud application, the name of each microservice that it is composed of, if it has a detachable resource, if this resource access to a DB or not, and some characteristics of the microservice. The classification associated is presented in a combo with the value of it, allowing to the developer to change this value.

2. **Theoretical deployment generation.** When the classification is made, and the NFRs informed by the developer, OPTIMUS prepares a request to invoke ACSmI Discovery and obtains the cloud services that fulfil the requirements of the microservices for their deployment.

   This request is composed of generic Cloud Services and the list of resources that the microservices need. At that moment of the project, the only Cloud Service class that is going to be handled is the Virtual Machine. This functionality requires interacting with the ACSmI API to obtain the list of Cloud Services that meet the criteria requested.

   The request to ACSmI discovery is built and performed based on the classification of the microservice, the characteristics associated to it and the NFRs for the application level. The aggregate and disaggregate values needed for calculating the level of fulfilment of the NFRs will be defined for the next version and implemented as part of the simulation algorithm.

   Moreover, when the information about patterns handled by ARCHITECT tool is defined, this will be part also of that algorithm, considering the impact that the pattern can have over the deployment schema.

   Therefore, this functionality is partially covered in this M24 prototype.

3. **Simulation**. The combination of the different possibilities of deployment, considering the theoretical and individual deployment possibilities for each microservice and the list of cloud services (from ACSmI Discovery) that suit them, will be ranked to select the best of them. For the M30 final version of the tool, five schemas will be presented to the developer to confirm the first of them or to select another one of that list of five. This M24 prototype shows only one schema and it assumes that the developer agrees if he does not perform another simulation changing some of the characteristics.

   The Schema includes information about the id of the cloud service in the ACSmI registry and the ids of the microservices that are set to deployed on it. This information, as well as a date to identify when this schema was obtained, is stored into the historical repository managed by the App Controller.

   The functionality is partially covered for M24 prototype. The REST service has been implemented and the simulation can be performed.

   The Eclipse plugin launches the simulation process for the current application and the result with the best deployment schemas is shown to the developer, who can store it in the application description, and therefore in the schemas history, or launch again the simulation changing some data about the application.

   It is the responsibility of the developer to upload the application description JSON file to the repository where it belongs, before and after launching the simulation. For that, the eclipse framework where the application description JSON file has been created or cloned, allows to commit the file once the information needed for the simulation is completed. If the developer needs to use another DECIDE tool inside the DevOps framework, he should upload again the JSON file.

   It has been implemented the interface with the Violation Handler tool which allows launching a simulation through the DevOps Framework

Requirements:

The requirements covered by the M24 OPTIMUS prototype are described in the Table 1.

The three first rows are related to the classification process, performed by the Eclipse plugin part of OPTIMUS, the rest of them are about the simulation process.

Table 1. Requirements covered by the M24 prototype.

Req. ID	Description	Requirement coverage by the prototype
WP3-PROFI-REQ1	Load/read information about the application (components).	Implemented. The prototype reads the information stored in the application description about the microservices and the NFRs.
WP3-PROFI-REQ2	Classify the application, based on the "stereotypes of the components" that we defined in the design phase of the profiling tool, and compare it with the information about the (component) application.	The status covers the classification considering the information about the microservices the tool knows. This requirement is partially covered, and it is planned to be finished in M30 release. The development was planned to be

Req. ID	Description	Requirement coverage by the prototype
		incremental from the beginning of the project.
WP3-PROFI-REQ3	Request the developer to confirm the classification	Covered. This version considers there is a confirmation when the developer saves the result of the classification into the application description JSON file.
WP3-OPTI-REQ3	OPTIMUS will analyse the application's NFRs and the classification (FR) in order to ask ACSmI for information about cloud services that cover the requirements (F/NF) of the multi-cloud application.	Implemented the creation of a filter to ask ACSmI about the cloud services that fulfil the requirements, considering the NFRs, and the characteristics of the microservices.
WP3-OPTI-REQ4	For each component of the multi cloud application, OPTIMUS engine builds the theoretical composition of services needed to the best possible deployment topology	Covered partially due to its incremental nature. The current implementation is done considering the information about the NFRs and the classification. More input data will require more implementation and complexity.
WP3-OPTI-REQ5	Once OPTIMUS engine runs the simulations for each component of the multi cloud application, each of them will be ranked	Covered partially due to its incremental nature. The current implementation covers a simple ranking taking the best match of each microservice for a cloud service and combining with the rest of the microservices. The combined schema consists in selecting the best match between a microservice and a cloud service for all the microservices and then group the microservices considering the cloud service selected for deploying it. More complexity in the selection and combination processes will be performed for the M30 prototype version.
WP3-OPTI-REQ7	OPTIMUS shall provide the developer with the information about the proposed deployment schema (those with the highest rank) for the application to cover the required NFR and FR, and the technological risk that each of these configurations imply. This	This requirement changed because it is not the scope of the best schemas to inform about the technological risks associated to its deployment. The new redefined requirement is covered partially because it is an incremental requirement which is being continuously improved as the input information can change over time.

Req. ID	Description	Requirement coverage by the prototype
	will show in the UI and will require confirmation from the developer	The information about the selected schema is the schema itself, with the id of the cloud service where a group of microservices can be deployed, and the ids if that group of microservices.
WP3-OPTI-REQ8	OPTIMUS tool can define new schema from developer side (proactively) and from results coming from ADAPT (reactively) to set up a new deployment schema, if a malfunctioning of a deployed multi-cloud application occurs	Implemented the possibility for launching the simulation by the ADAPT Violation Handler module. The status is ready for integration tests

2.1.1 Fitting into overall DECIDE Architecture

OPTIMUS is the DECIDE tool that is responsible for obtaining the five best deployment schemas for the multi cloud application. The global DECIDE architecture is shown in Figure 1. The role of OPTIMUS in this global architecture has not changed compared to the previous version delivered in D3.7 [5].

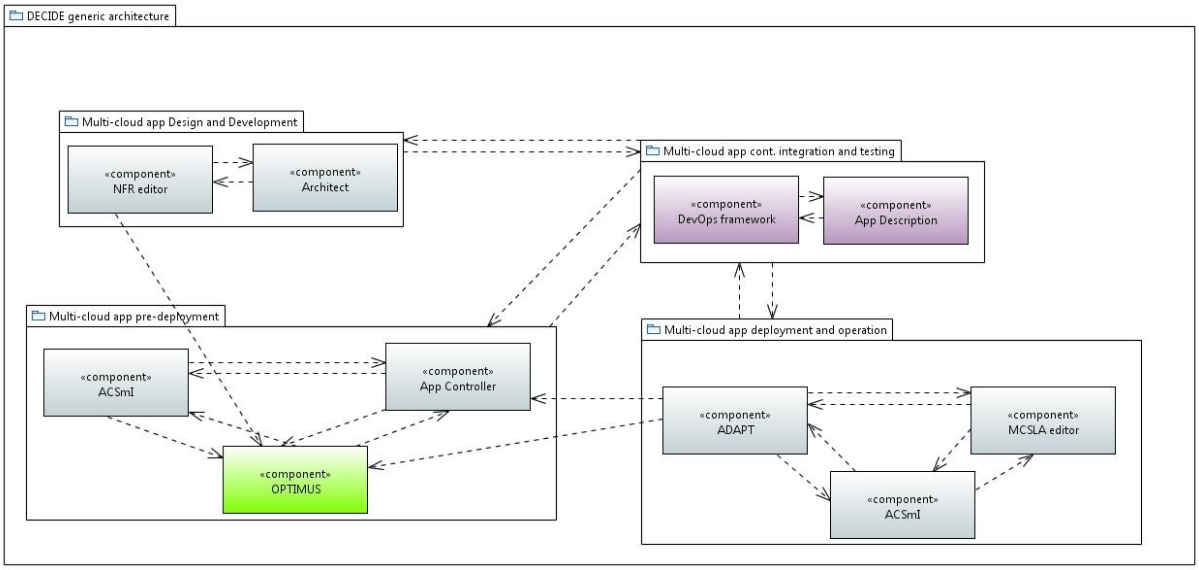

Figure 1. OPTIMUS in DECIDE architecture**.**

The interaction with OPTIMUS and the rest of the DECIDE tools remains as planned and was described in the previous deliverable [2].It is presented in the Figure 2.

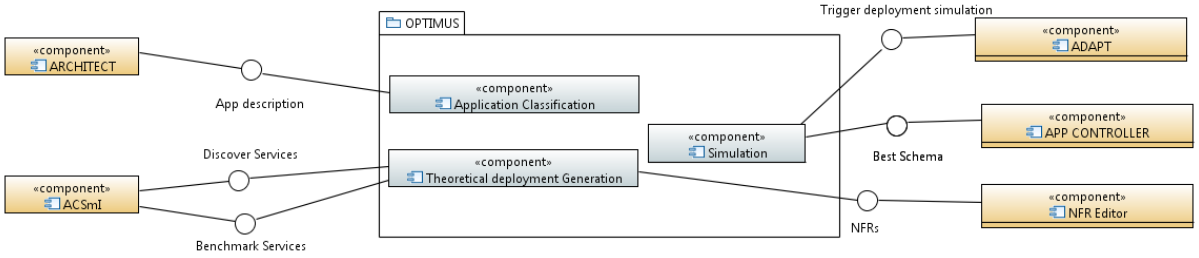

Figure 2. OPTIMUS interfaces within DECIDE framework

These interactions are as follows:

- ARCHITECT: This is the starting point to the DECIDE framework when the developer wants to use OPTIMUS tool. It provides a General Editor with which the information of the application will be stored into the application description JSON file.
- ACSmI: The information about the cloud services where the microservices can be deployed on, is provided by ACSmI Discovery.
- ADAPT: When a violation of the MCSLA is discovered, ADAPT Violation Handler triggers OPTIMUS to obtain a new best deployment schema.
- Application Controller: The structure of the application description JSON file is defined by the Application Controller component, as well as the operations that can be performed with it. Moreover, the deployment schema obtained by OPTIMUS Simulation is managed also through the Application Controller library when storing it into the historic repository and consulting it to avoid a deployment under the same schema.
- NFR Editor: The NFR Editor is part of the General Editor in ARCHITECT. For each microservice the developer can specify Non-Functional Requirements associated to it. OPTIMUS tool uses the information about the NFRs to build the request to ACSMI Discovery and obtain the best schema for the deployment.

2.2 Technical description

In this section, the technical aspects about the development of this version of the OPTIMUS prototype are presented.

2.2.1 Prototype architecture

The general architecture planned for OPTIMUS tool is shown in Figure 3.

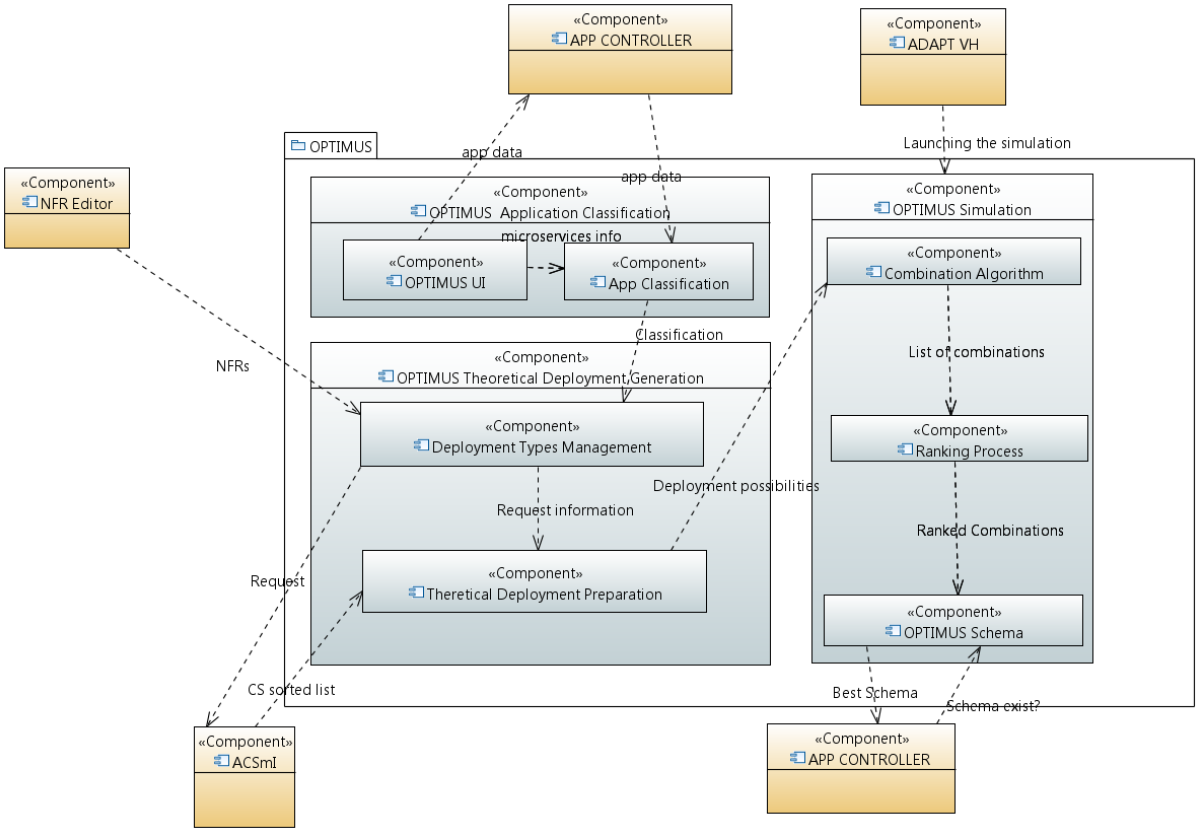

Figure 3. OPTIMUS High level architecture

This architecture has changed with respect to the one depicted in the previous deliverable.

The *Types management* and the *Apps classification repo* components have been included into the other two sub-components of the "*OPTIMUS Theoretical Deployment generation*" component. This was due to a technical decision, that concluded that these components do not have enough individual entity.

M24 OPTIMUS prototype architecture consists of all the components of the general architecture, although their implementation is not totally done.

2.2.2 Components description

The main components detailed in the OPTIMUS general architecture are represented in Figure 4.

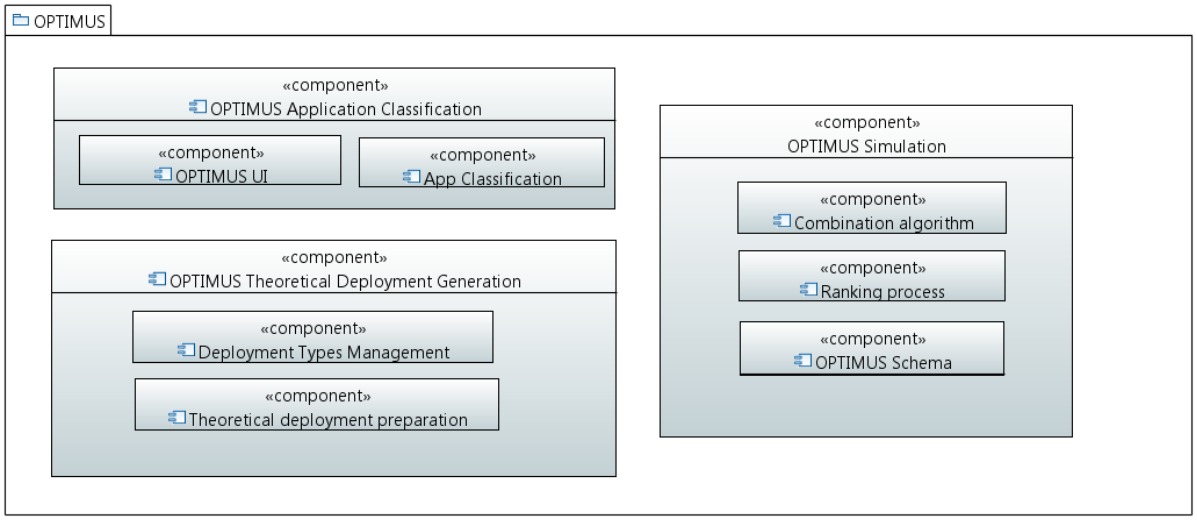

Figure 4. OPTIMUS component diagram

Application classification

The Application Classification component is presented to the developer as a tab of the OPTIMUS eclipse plugin. Through this UI the developer provides the information about the application and the microservices which it is composed of, to classify each of those microservices.

The App Classification subcomponent will match the information stored about the different possible classifications and the characteristics associated to each of the multi-cloud application microservices, writing the corresponding value into the application description JSON file using the Application controller library.

The classification process is explained more deeply in the DECIDE deliverable D3.5 Intermediate profiling and classification techniques [1]

Theoretical deployment generation

Considering the classification of each microservice, OPTIMUS is capable of associating the cloud service class that can be candidate for its deployment. Knowing the cloud services objectives, together with the NFRs and some characteristics established by the developer, OPTIMUS builds a request to obtain from ACSmI the cloud services that fit the requirements set in that request.

Processing the ACSmI answer, storing each possibility as a structured object of information, the Theoretical Deployment preparation arranges all the input that the Simulation component needs.

Simulation

The deployment possibilities group is a list of elements where are associated a cloud service and a microservice with a specific weight that means the level of fulfilment of this association with the requirements stablished by the developer for that microservice, including the NFRs. The combination of that information consists of the creation of the different possibilities of grouping all the microservices in a single deployment schema. The result of this combination will be a list of the possible schemas considering each element of the schema in an individual way, that is, one cloud service and one microservice and the weight associated.

With the individual associations as schemas, the next step is grouping the microservices that have the same cloud service associated and calculate de weight for each of them.

Once the five best ranked schemas have been obtained, they will be presented to the developer to confirm it, and then sent the selected one to the App controller.

In this M24 OPTIMUS prototype, they will be presented just the best of them.

This Simulation phase can be also triggered by DECIDE ADAPT Violations Handler (VH) and in this case the schema is planned to be the best one and it will be stored automatically in the application description JSON file.

2.2.3 Technical specifications

The DECIDE OPTIMUS M24 prototype has been developed as two separate parts, one in the form of an Eclipse plugin with the structure of a multipage editor, and the other one as a REST service that allows to be invoked from different parts of other tools or software elements. So, the tool must be installed in an Eclipse framework for its use by the developer, and the REST service can be triggered for the Violation Handler or launched by the DevOps framework if the project would have considered to add this possibility.

Moreover, the plugin is executed together with the General Editor part of ARCHITECT. The implementation has been made using the extensions mechanism that allows adding tabs from a plugin to another previously launched.

The multipage editor consists of three tabs. The first of them is part of the General Editor and contains the Application Description JSON file. It will reflect the information that the developer introduces using the other tabs. The second tab is for the classification, where a group of microservices are shown if they are in the application description JSON file. The third tab corresponds to the simulation process, from where the simulation can be launched, and its result can be seen.

For developing the graphical object related to the OPTIMUS UI (multipage editor), the WindowBuilder [6] Eclipse plugin has been used, which has been developed to create Java GUI applications by dragging and dropping elements from a palette onto a design surface, in this case the tabs of the multipage editor. The structure of the developed plugin has five (5) basic elements and each of them is a java class element:

- *Classification.java*: it manages the appearance of the Classification tab and the processes assigned to each element on it.
- *MicroserviceClassification.java*: it is the element that is responsible for creating the group of objects to gather the information about one microservice. Each time the developer pushes the "Add microservice" button, the same group of objects will be presented for him to fill them with the corresponding information.
- *SimulateSchema.java*: it manages the Simulation tab. This OPTIMUS element will create this tab and will include the results of the simulation, that is, the best schemas for the deployment.

- *ClassificationPageBuilder*: This is the element that implements IPageBuilder and creates the additional tab for the classification. It follows the rules for working with extensions.
- *SimulationPageBuilder*: This is the element that implements IPageBuilder and creates the additional tab for the simulation. It follows the rules for working with extensions.

On the other hand, the REST Service has been created using the Swagger specification [3] for defining the service and the format to manage it.

This specification is a JSON file from which Swagger generates the server with the REST service and the template for the implementation of the service, and the client, which will be the library for using the service from other modules or tools.

The template generated by Swagger for the server has been added to the Eclipse development framework as a Maven [7] project, so, once the lines of code needed for providing the planned functionality have been introduced, the server can be deployed in the integration framework for testing it.

The main elements developed in this project are three (3) class elements described as follows:

- *Bootstrap.java:* This is the starting point when the REST service generated by Swagger is called. It has been generated by Swagger and completed by TECNALIA.
- *SimulatorThread:* This class contains the intelligence to obtain the schema, performing the request to ACSmI discovery and returning the best schema to the Eclipse plugin, or saving it into the application description JSON file in case the VH has been the service caller.
- *ApplicationApiServiceImpl:* This file gathers the different methods related to the available REST operations that the service publishes. It has been generated by Swagger and completed by TECNALIA.

More java elements are generated by Swagger and are included in the Maven Eclipse project. They can be seen in Figure 5.

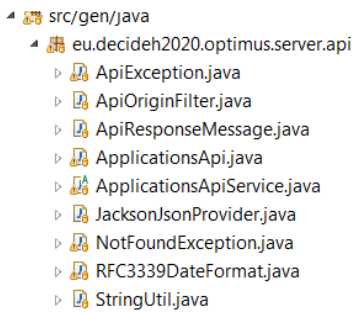

Figure 5. Generated java classes by Swagger**.**

3 Delivery and usage

3.1 Package information

The structure of the OPTIMUS plugin package in Eclipse is as follows:

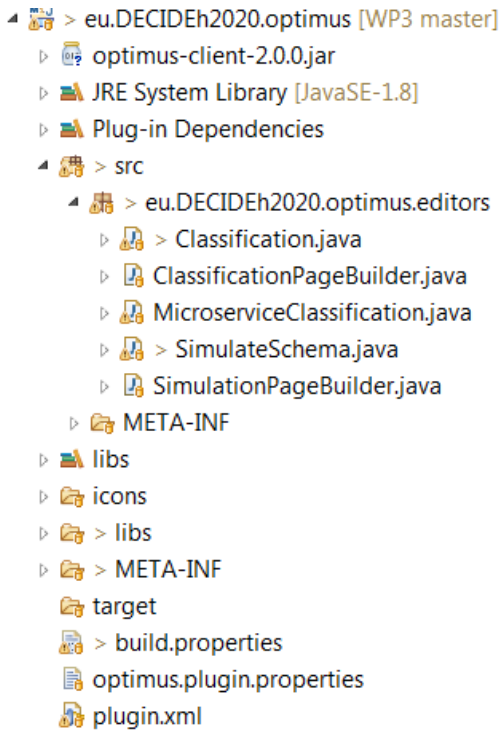

Figure 6. Eclipse Source folder structure of OPTIMUS plugin component**.**

The package *eu.DECIDEh2020.optimus.editors* contains the source code for the OPTIMUS plugin.

The server with the corresponding code for the REST service is a separate Maven project and has the following structure:

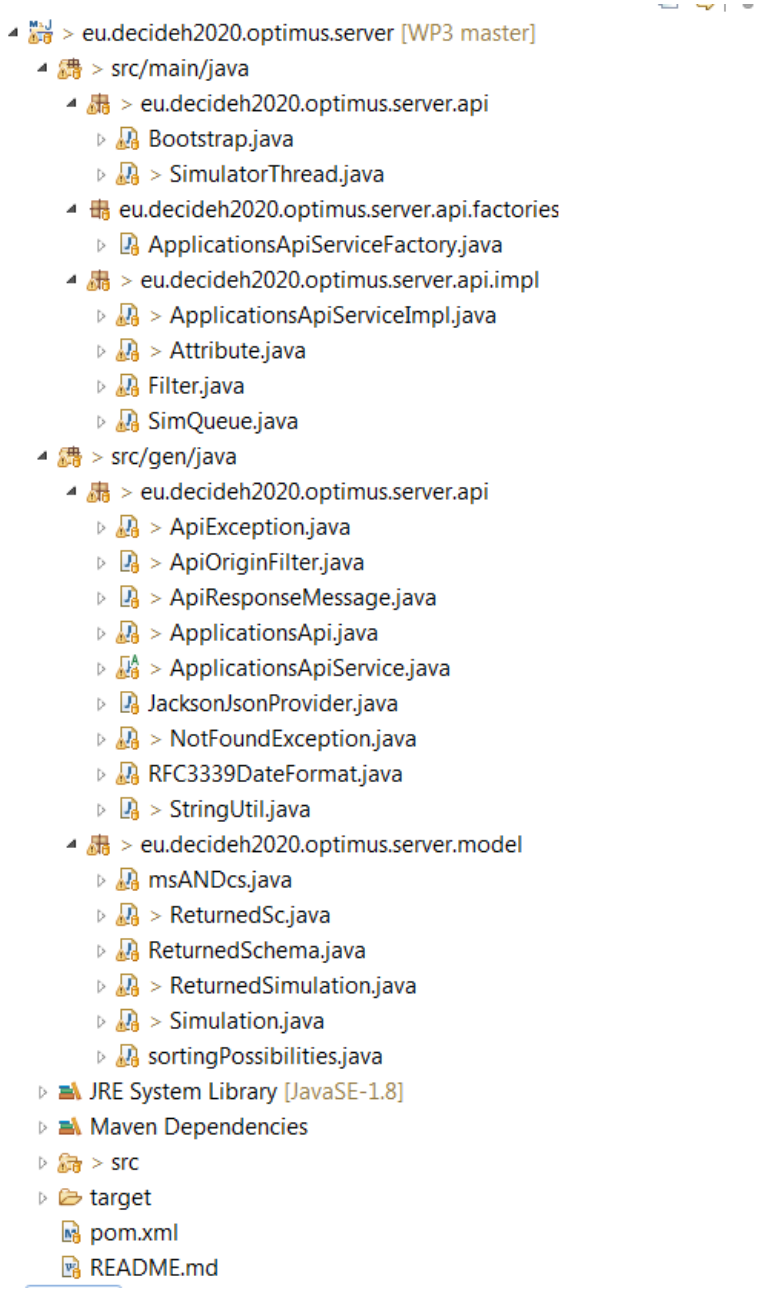

Figure 7. Eclipse Source folder structure of OPTIMUS server component**.**

The different packages depicted above contain the following elements:

- (main) *eu.decideh2020.optimus.server.api*: Where the main elements of the server implementation are placed, the starting point and the code for the simulation.
- (main) *eu.decideh2020.optimus.server.api.factories*: A generated file for starting and creating the service.
- (main) *eu.decideh2020.optimus.server.api.impl:* Elements needed for the implementation of the main java files placed in (main) *eu.decideh2020.optimus.server.api* package.
- (gen) *eu.decideh2020.optimus.server.api*: Elements to be internally used, generated by swagger.
- (gen) *eu.decideh2020.optimus.server.model*: Some data structures generated by swagger and completed by TECNALIA.

3.2 Installation instructions

The DECIDE OPTIMUS tool has too different parts of code, the eclipse plugin and the project maven with the server. The installation of the eclipse plugin requires the eclipse IDE. The installation of the server should be done in an accessible infrastructure.

The software needed for running OPTIMUS classification M24 prototype is available in this url:

https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components.git

For running OPTIMUS plugin tool M24 prototype, via Eclipse java project:

a. Start Eclipse IDE (Eclipse Oxygen)
b. Clone the repository indicated above
c. Import the following projects:
    i. eu.DECIDEh2020.optimus
    ii. eu.decideh2020.optimus.client (as maven project)
    iii. AppController (as maven project)
d. Run as maven install eu.decideh2020.optimus.client and AppController.
e. Include in the build path of eu.DECIDEh2020.optimus project, the jar obtained from eu.decideh2020.optimus.client
f. Run As an Eclipse application.

There are some lines of code where this version includes some specific URLs and credentials. Before running OPTIMUS, these lines should be changed:

- SimulateSchema.java in eu.DECIDEh2020.optimus project:
    - Line 154: Substitute ***Appurl*** by the git url where is stored the application description JSON file.
    - Line 162. Substitute ***URLoptimusserver*** by the URL where OPTIMUS server is installed.
- SimulatorThread in eu.decideh2020.optimus.server project:
    - Line 130: Substitute ***user*** and ***password*** by credentials to access to the git where the application description JSON file is placed.
    - Line 301: Substitute ***acsmiServicesURL*** by the URL where ACSmI is installed.

For installing and running the server the following software is needed:

- Docker to create the container.

To get the prototype up and running it must to follow these steps:

1. Download the source code from the DECIDE repository.

```
git clone
https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components.git
```

2. Go to the folder where the docker file is located:

```
cd <OPTIMUS server folder>\
eu.decideh2020.int.optimus.server.src.dvp\src\main\docker
```

i. Build the docker image with the following arguments:

```
docker build -f /docker.optimus.server -t
tecnalia/eu.decideh2020.int.optimus.server
```

ii. Run the docker image with the following arguments:

```
docker run -d --restart=always -p 8090:8090 --name
eu.decideh2020.optimus.server
tecnalia/eu.decideh2020.optimus.server
```

3.3 User Manual

Once the developer runs the plugins, he must create a new DECIDE JSON file, containing the application description, licking right mouse button and selecting New → Other

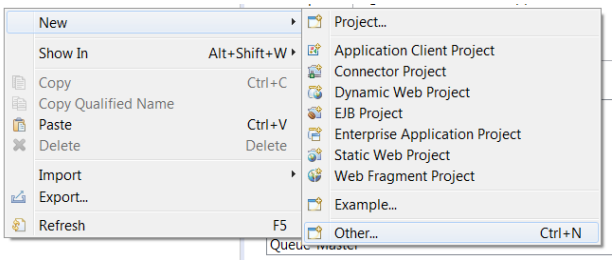

Figure 8. Creation of a new file.

In the next window, select "DECIDE project" from the DECIDE Wizards option

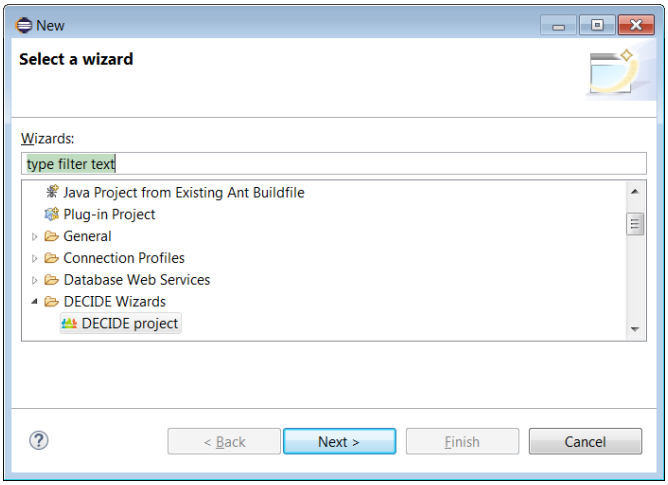

Figure 9. Selection of the DECIDE Editor file.

Then, indicate the Project name, the local folder, and the information about the git repository where the JSON file is stored for "Clone Remote Repository" option.

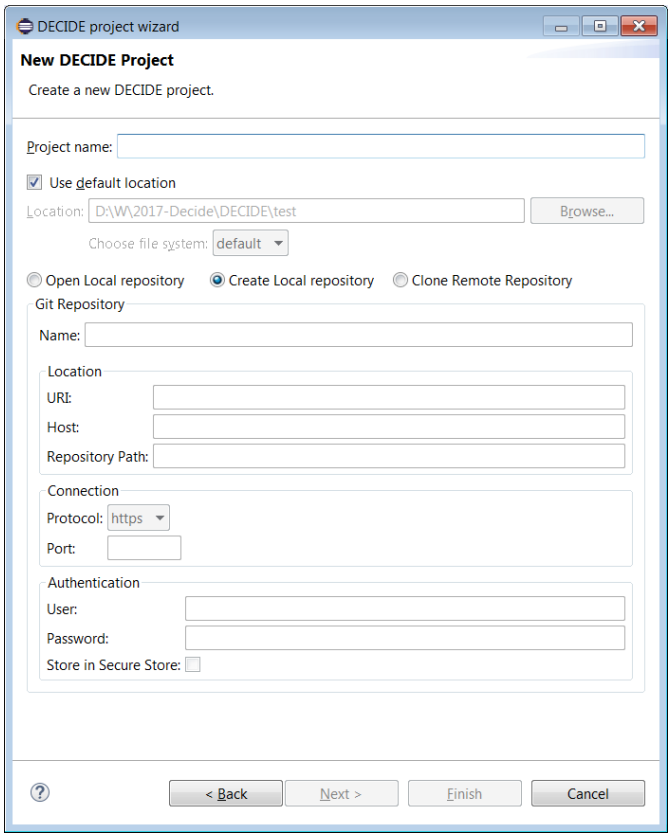

Figure 10. Selection of the DECIDE JSON file's information.

When selecting the "Finish" button, the initial content of the file is shown in the first tab of the editor.

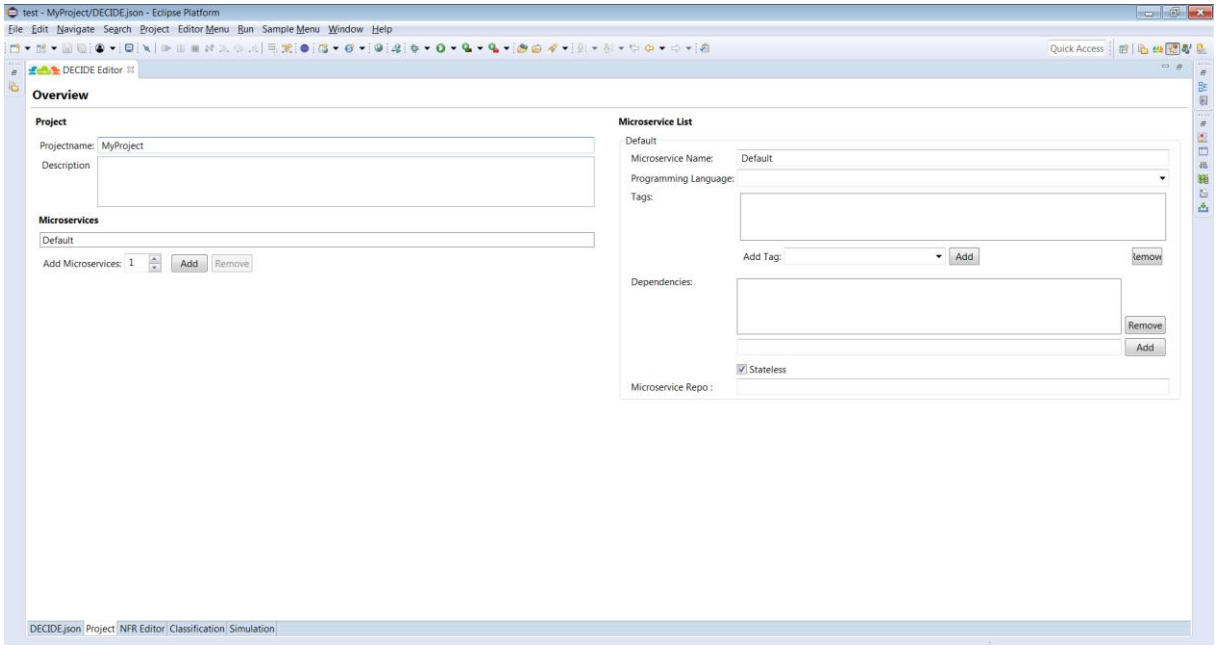

Figure 11. Initial new file.

The first three tabs correspond to the current version of the General Editor and show the JSON file in raw (DECIDE.json tab), the information as a UI (Project tab) and the information about the NFRs (NFR Editor tab). These tabs are not part of the OPTIMUS tool and they are implemented under the ARCHITECT tool.

OPTIMUS and ARCHITECT are two tools that can be executed in an Eclipse framework, so they need first a General Editor to allow the developer to introduce the general data about the Application, as well as the DevOps framework does.

The first OPTIMUS tab is the Classification tab. The new file has by default just one microservice, and the classification tab reflects this situation:

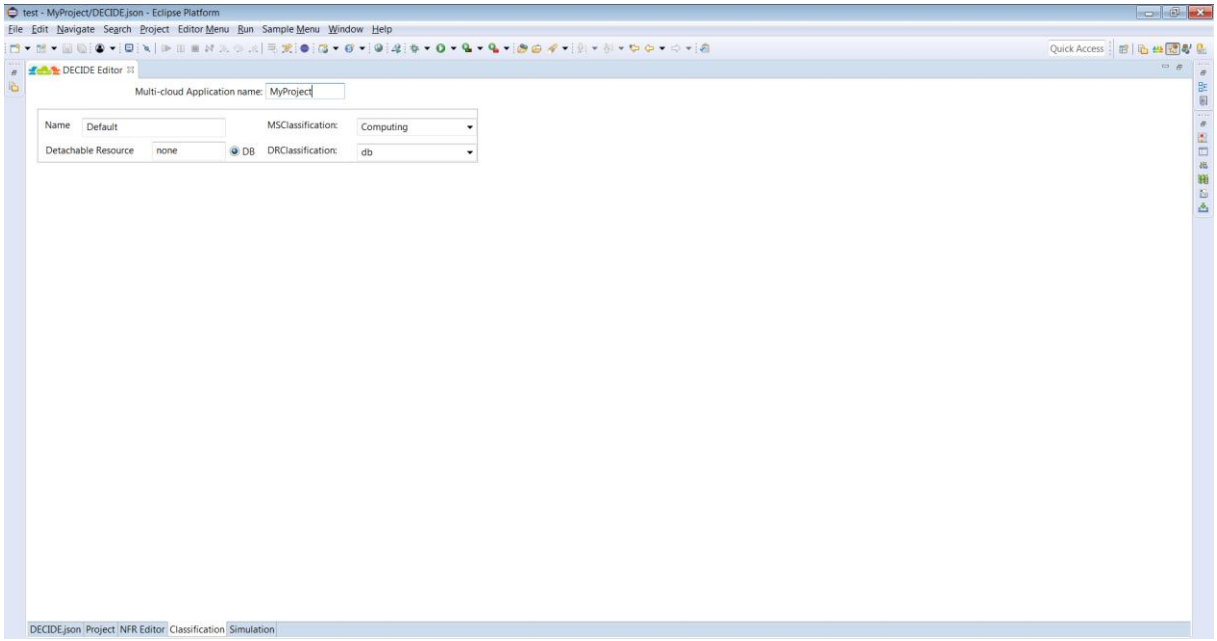

Figure 12. Classification tab.

The simulation tab is the area where the developer, once the information about the microservices has been fulfilled, can launch the simulation for that specific application.

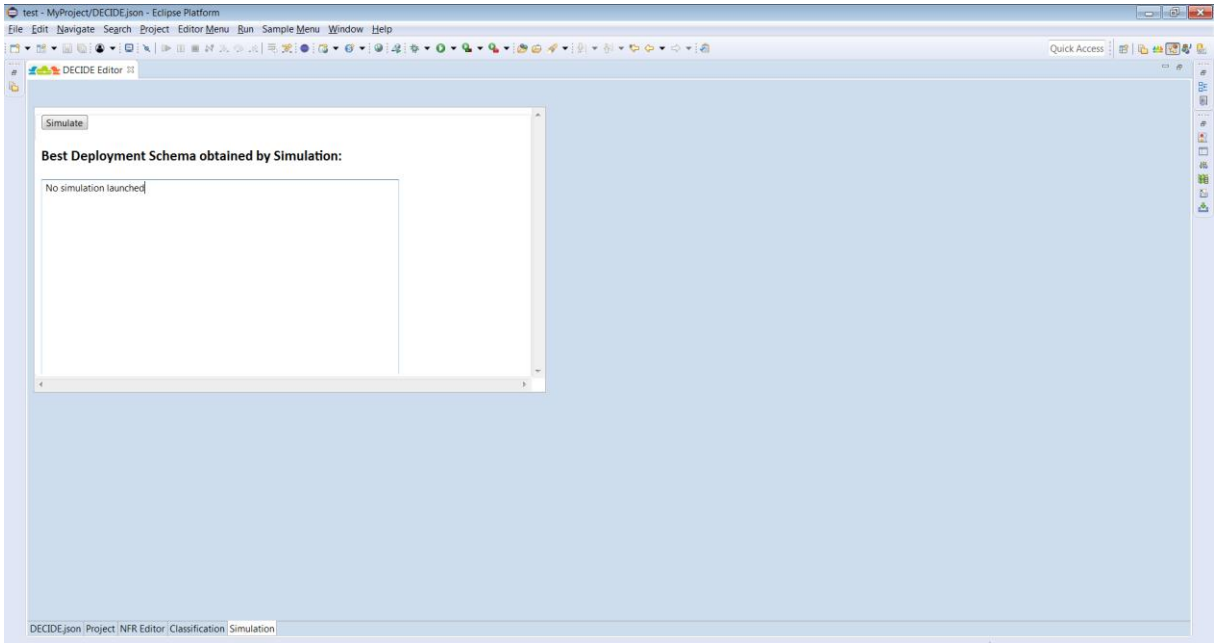

Figure 13. Simulation tab.

For classifying a microservice in the Classification tab, this prototype assigns the "Computer" value by default allowing the developer to change it to "Computing Public IP" if he considers it more appropriate.

Each microservice can have a detachable resource associated to it, of which the developer has to introduce the name and select the DB aspect when corresponds. In that moment, the value of its classification will change, and it will show the value "db", otherwise the value will be "storage".

More microservices can be added using the General Editor Tab, but the detachable resources can only be specified through the Classification tab. These elements are considered elements associated to a principal microservice, and in the simulation they will be deployed in the same Cloud Service as its main microservice.

The DECIDE.json for the SockShop application [8] can be analyzed in the following pictures:

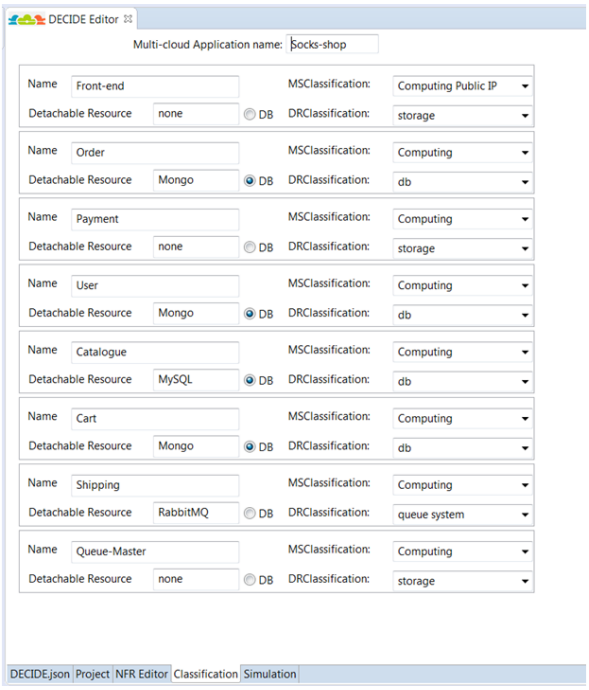

Figure 14. Classification tab for SockShop Application example.

The Classification tab is complete for that example and if the developer wants to obtain a deployment schema for that application considering the NFRs stablished and the characteristics of the microservices, he must push the “Simulate” button to launch the REST service that performs the simulation process.

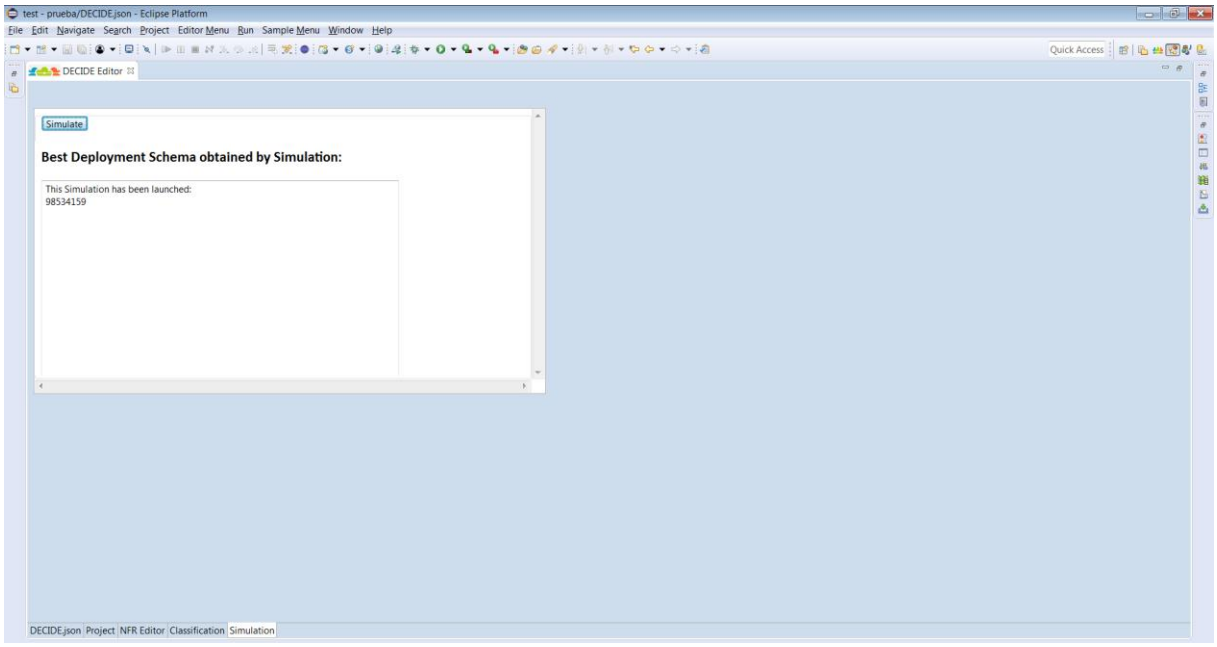

Figure 15. Simulation launched.

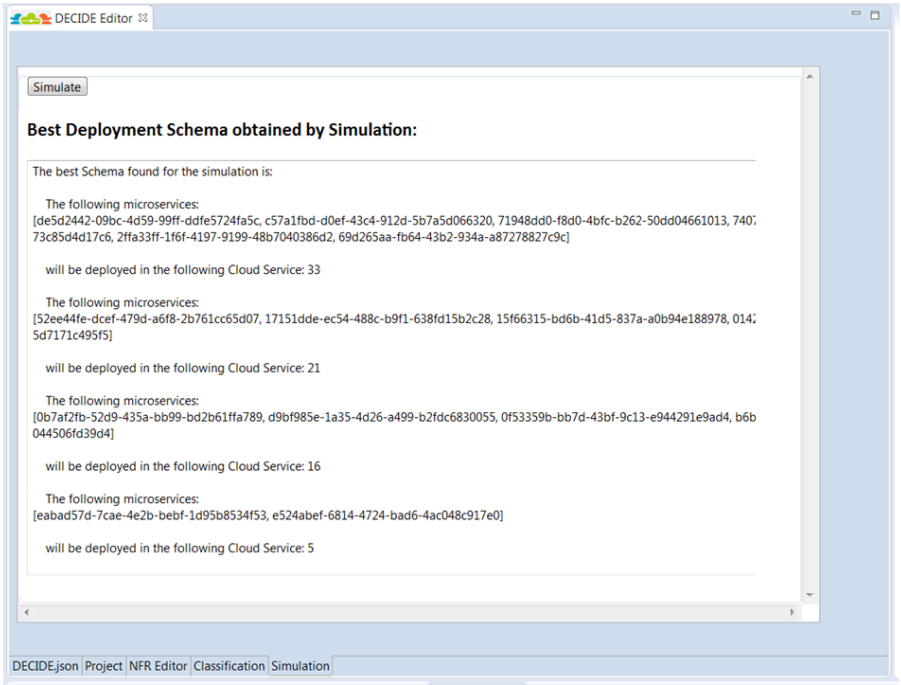

Figure 16. Deployment Schema.

The result of the simulation appears in the textbox and the elements of the deployment schema obtained are:

- index: internal index to identify a specific association of CSid and microservices Ids. It is useful for the further deployment process. Not appearing in this tab.
- List of elements composed by:
    - Group of microservices id: The ids of the microservices that should be deployed in the Cloud Service mentioned bellow.
    - Cloud Service id: The id of a selected Cloud Service. The information about this CS can be found in the ACSmI Discovery registry.

For example, the schema saved into the DECIDE.json following Figure 16 result


```
"schema" : [ {
    "microservices" : [ "de5d2442-09bc-4d59-99ff-ddfe5724fa5c", "c57a1fbd-d0ef-
43c4-912d-5b7a5d066320", "0f53359b-bb7d-43bf-9c13-e944291e9ad4", "b6bfeb8d-f881-
4e4f-9b0c-044506fd39d4" ],
    "csId" : "16",
    "index" : 0
  }, {
    "microservices" : [ "71948dd0-f8d0-4bfc-b262-50dd04661013", "7407e313-d2fc-
44a9-8830-73c85d4d17c6" ],
    "csId" : "23",
    "index" : 1
  }, {
    "microservices" : [ "0b7af2fb-52d9-435a-bb99-bd2b61ffa789", "d9bf985e-1a35-
4d26-a499-b2fdc6830055", "2ffa33ff-1f6f-4197-9199-48b7040386d2", "69d265aa-fb64-
43b2-934a-a87278827c9c", "15f66315-bd6b-41d5-837a-a0b94e188978", "01420849-546d-
4c34-950b-5d7171c495f5" ],
    "csId" : "8",
    "index" : 2
  }, {
    "microservices" : [ "52ee44fe-dcef-479d-a6f8-2b761cc65d07", "17151dde-ec54-
488c-b9f1-638fd15b2c28" ],
    "csId" : "19",
    "index" : 3
  }, {
    "microservices" : [ "eabad57d-7cae-4e2b-bebf-1d95b8534f53", "e524abef-6814-
4724-bad6-4ac048c917e0" ],
    "csId" : "7",
    "index" : 4
  } ],
```


This Schema means that the microservices corresponding to these ids: "de5d2442-09bc-4d59-99ff-ddfe5724fa5c", "c57a1fbd-d0ef-43c4-912d-5b7a5d066320", "0f53359b-bb7d-43bf-9c13-e944291e9ad4", "b6bfeb8d-f881-4e4f-9b0c-044506fd39d4 will be deployed in the Cloud Service stored in ACSmI registry with the number 16. This row or specific association, is called as an individual deployment number 0 (index).

The following individual deployments have the same meaning. They are four more, in total five Cloud Services used to deploy the whole application.

3.4 Licensing information

The information about the license under which the software will be distributed, has been placed at the header of all the code files (*.java files).

These headers are composed of the following lines:

```
/*******************************************************************************
* Copyright (c) 2018 Tecnalia.
*
* This program and the accompanying materials are made
* available under the terms of the Eclipse Public License 2.0
* which is available at https://www.eclipse.org/legal/epl-2.0/
*
* SPDX-License-Identifier: EPL-2.0
* Contributors (in alphabetical order):
* Alberto Molinuevo                         Tecnalia
* Gorka Benguria                            Tecnalia
* Iñaki Etxaniz                             Tecnalia
* Juncal Alonso                             Tecnalia
* Leire Orue-Echevarria                     Tecnalia
* Maria Jose Lopez                          Tecnalia
```

```
* Marisa Escalante                          Tecnalia
* Initially developed in the context of DECIDE EU project www.DECIDE-h2020.eu
*************************************************************************/
```

3.5 Download

The eclipse java source code of the OPTIMUS project is available in the DECIDE open git repository, accessing this URL:

https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components/tree/M24/OPTIMUS

This code is the project with the source code corresponding to the DECIDE OPTIMUS classification tool M24 prototype.

4 Conclusions

This deliverable presents the evolution of the DECIDE OPTIMUS tool and its status as M24 prototype. Future versions of this deliverable will include the final version of the tool, covering all the features that the DECIDE OPTIMUS tool will provide to the developer.

The details about the functionality and the technical aspects are described in the corresponding sections as well as the manual and the instructions to test the software.

The evolution of the application description and the need to integrate all the DECIDE tools, could lead to modifications of all DECIDE tools and their interactions, and consequently OPTIMUS will have to adapt to the new schema.

The most important future step for OPTIMUS is the algorithm for establishing the different possibilities of combinations of the available cloud services, to present the best schema considering not only the NFRs and the characteristics of the microservices but also the impact the selection of a specific pattern would have over the best schema for the deployment.

References

[1] DECIDE Consortium, “D3.5 – Intermediate profiling and classification techniques,” 2018.

[2] DECIDE, “D3.4 Initial profiling and classification techniques.,” 2017.

[3] S. Software, “SWAGGER,” 2018. [Online]. Available: https://swagger.io/. [Accessed November 2018].

[4] W3C, “Web Services Architecture,” 2004. [Online]. Available: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest. [Accessed November 2018].

[5] DECIDE Consortium, “D3.7 Initial DECIDE OPTIMUS,” 2017.

[6] I. Eclipse Foundation, “WindowBuilder,” [Online]. Available: https://www.eclipse.org/windowbuilder/. [Accessed 2018].

[7] T. A. S. Foundation, “Apache Maven project,” 2018. [Online]. Available: https://maven.apache.org/. [Accessed 2018].

[8] I. Weaveworks, “Sock Shop: A Microservices Demo Application,” 2017. [Online]. Available: https://github.com/microservices-demo/microservices-demo/blob/master/internal-docs/design.md. [Accessed November 2018].