Deliverable D3.5

Intermediate profiling and classification techniques

Editor(s):	María José López
Responsible Partner:	TECNALIA
Status-Version:	Final . V1.0
Date:	28/11/2018
Distribution level (CO, PU):	PU

Project Number:	GA 731533
Project Title:	DECIDE

Title of Deliverable:	Intermediate profiling and classification techniques
Due Date of Delivery to the EC:	30/11/2018

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

Abstract:	This deliverable comprises the intermediate models for CSPs, microservices and deployment schemas, and some other models for prototypical application components.
Keyword List:	Modelling, classification, profiling
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	01/10/2017	ToC	TECNALIA
V0.2	26/10/2018	First version without general sections	TECNALIA
V0.3	29/10/2018	First draft with all the sections	TECNALIA
V0.4	5/11/2018	Update of NFRs structure	TECNALIA
V0.5	6/11/2018	Ready for review	TECNALIA
V0.6	23/11/2018	Modified according the review made	TECNALIA
V0.7	28/11/2018	Included the comments from the second internal review	TECNALIA
V1.0	30/11/2018	Ready for submission	TECNALIA

Table of Contents

List of Figures

List of Tables

Terms and abbreviations

AD	Application Description
CS	cloud service
CSP	cloud service Provider
DB	Data Base
DS	Deployment Schema
IP	Internet Protocol
JSON	JavaScript Object Notation
NFP	Non Functional Properties
NFR	Non-Functional Requirement
RAM	Random Access Memeory
REST	Representational State Transfer
SLA	Service Level Agreement
UI	User Interface
URI	Unified Resource Identifier
WP3	Work Package 3

Executive Summary

The DECIDE framework manages the information needed to provide the developers with the best deployment related to the characteristics set by them. The multi-cloud application will be deployed and running in an enviroment that the DECIDE framework can assure it is the most suitable for it, built on resources from Cloud Service Providers (CSP) chosen among those registered in the DECIDE ACSmI catalogue.

The Application Description (AD) JSON file is the shared data structure that contains all the information that the DECIDE tools need. The AD document includes the data model for every data structure that the tools manage.

This deliverable, in section 2, starts explaining the model describing the microservices that DECIDE handles, it also explains which tool is in charge to obtain which properties as well as the meaning of those properties.

Cloud services are also very important in the DECIDE integrated framework, so the model reflects the information that is stored about them in the ACSmI registry, as it is described in section 3.

A new data model is introduced in this intermediate deliverable, in section 4, namely the Deployment Schema (DS), as the result of the DECIDE OPTIMUS Simulation tool. It consists of a list of associations between a cloud service and the group of microservices that will be deployed onto them. This DS will be stored into the Application Description JSON file.

Moreover, this document shows the current status of the DECIDE classification tool, section 5, in its intermediate state, the requirements that are planned to be implemented by M24, and the description of the process that it is followed to classify a microservice. This tool in its current status is the evolution of the previous one developed in M12 and will enjoy more changes until the final version. Section 6 describes the mapping between the value of the microservices classification and the classes of the Cloud services offerings gathered in the ACSmI Discovery registry

Finally, in section 7, this document describes models for some additional structures that are managed by the simulation process performed right after the classification. Those structures are based on the data stored into the Application Description JSON file and the informatioun about the Cloud services managed by the ACSmI registry. They are intermediate structures needed for OPTIMUS and related to the Deployment Schema described above.

The DECIDE framework handles a shared data structure across all DECIDE tools and does not need an specific editor for modelling applications or deployments by the developer. This document presents the current status of the data models related to the multi-cloud application and how their properties impact onto the selection of the best deployment schema. The next version of this document will show the final models handled when a developer uses DECIDE to deploy his or her multi-cloud application in the best possible way.

1 Introduction

1.1 About this deliverable

This document is the second version of the "Profiling and classification techniques" deliverable of Work Package 3, Continuous Architecting. Its aim is to establish the most appropriate way to represent the information about a multi-cloud application as well as the cloud services where it will be deployed to make it understandable by all the other DECIDE tools. Moreover, the classification process will be explained also as an important task to obtain the best match with the cloud services that are in the DECIDE ACSmI registry.

A deep description of the Application Description is included as an appendix in the deliverable *"D2.5: Detailed architecture v2"* [1]. Moreover, because all the DECIDE tools use the Application Description, the deliverables related to each of them deal with its data structures, either from the perspective of the microservices model, such as OPTIMUS [2] or from one that considers service level agreement aspects, such as MCSLA Editor [3].

1.2 Document structure

The organization of this document consists of the following sections.

Section 2 describes the model of the microservices as a part of the multi-cloud application and the tools that are responsible for collecting the related information and inserting it in the Application Description. Section 3 is concerned with the modelling of the cloud services managed by ACSmI, and stored into the ACSmI registry. Section 4 introduces the structure of the Deployment Schemas, the result of the OPTIMUS simulation process. Section 5 focuses on the current status of the OPTIMUS classification tool and section 6 complements it by explaining how that tool maps the microservices and the Cloud services where they can be deployed. Section 7 is devoted to presenting the model for some auxiliary structures and their use during the Simulation process. Finally, section 8 presents the conclusions and future activities for the next version of the deliverable.

Modelling microservices

Multi-cloud applications refer to applications that dynamically can distribute their components over heterogeneous cloud resources and still hold the functional, business and non-functional properties (NFP) declared in their SLAs [4]. Distributed multi-cloud applications are typically developed according to the microservices architecture.

In the DECIDE enviroment, the information about the application and their microservices, are stored in the Application Description, a JSON file with which all the DECIDE tools can interact to obtain and set the data managed by each of them.

In DECIDE, each microservice has the following structure of fields:

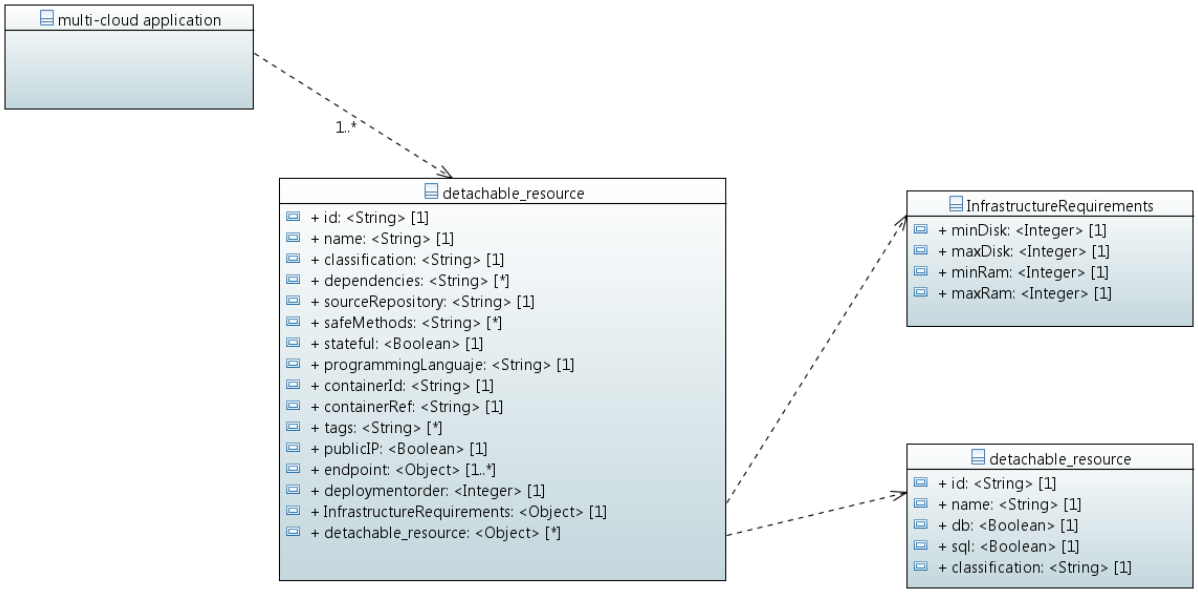

Figure 1. Microservices structure

As a part of the multi-cloud application each of the microservices will be deployed on a cloud service, which can be a different one or one of the cloud services already used for another microservice deployment. The most convenient distribution will be decided by DECIDE OPTIMUS tool taking into account some of the properties detailed in the Figure 1.

The *classification* property is the result of the classification process performed by OPTIMUS. The following properties are part of the input to decide the type of the microservice.

id: Unique Identifier for the microservice.
name: Human readable name for the microservice
dependencies: other microservices from which the microservice depends on.
sourceRepository: url of the repository where the microservice (software) will be placed
safeMethods: Information provided by DevOps Framework/General editor
stateful: True when the microservice keeps track of its status.
programmingLanguage: Type of programming language used for developing the microservice.
containerId: Id of the microservice's container
containerRef: Reference of the microservice's container
tags: Tags to relate NFRs with microservices
publicIP: True if the microservice needs a public IP address

endpoints: List of URIs to access the services and their methods
deploymentorder: If there are some dependencies from other microservices, the order in which this one should be deployed.
infrastructureRequirements: Minimum and maximum requirements for Disk and RAM.
detachable_resource: The list of resources (if any) that are going to be used by the microservice as for example external DB services

The main tools that manage those properties are Devops Framework (or the general editor plugin), and OPTIMUS. The rest of the tools can read the data but they do not modify them.

Figure 2 shows the properties that these tools read and write. The DevOps Framework requests the developer through its UI the following properties:

- name
- dependencies
- sourceRepository
- safeMethods
- stateful
- programmingLanguage
- tags

and once the developer provides the required information, the Editor writes it in the Application Description JSON file, and assigns an id for the application.

Through the OPTIMUS UI, using the classification tab in the eclipse plugin, the developer enters the following properties:

- publicIP
- endpoints (*)
- deploymentorder (*)
- InfrastructureRequirements (*)
- Detachable_resource

(*): Not implemented in this version of the UI.

Once that is finalized, OPTIMUS calculates and writes the *classification* value in the Application Description JSON file. The different values for the *classification* field are detailed in Section 5 of this document.

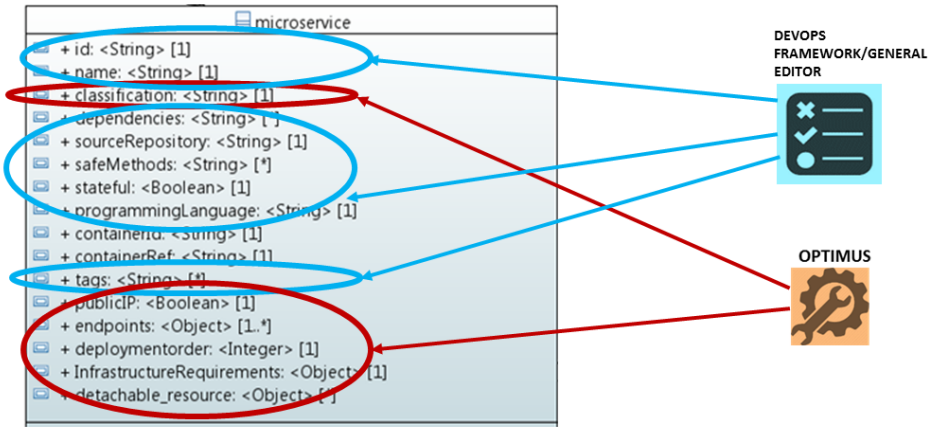

Figure 2. Tools that provide the properties of the microservices

2 Modelling Cloud services

Cloud services are important entities in the DECIDE integrated framework. Several tools, such as OPTIMUS (both in its simulation and classification processes), ADAPT, and ACSmI, will use the information related to these services and their Cloud Service Providers (CSPs).

The class diagram that models the cloud service offerings has not changed significantly since the previous version of this document, and its structure is shown in Figure 3. This model mirrors the one from ACSmI, presented in the document "*D5.3 Initial Advanced cloud service meta Intermediator*" [5]. All the information collected about the cloud services is stored in the ACSmI Service Registry. OPTIMUS calls ACSmI to request information about the cloud services stored in there in order to select the most appropriate ones following the process defined in sections 5 and 6 of this document.

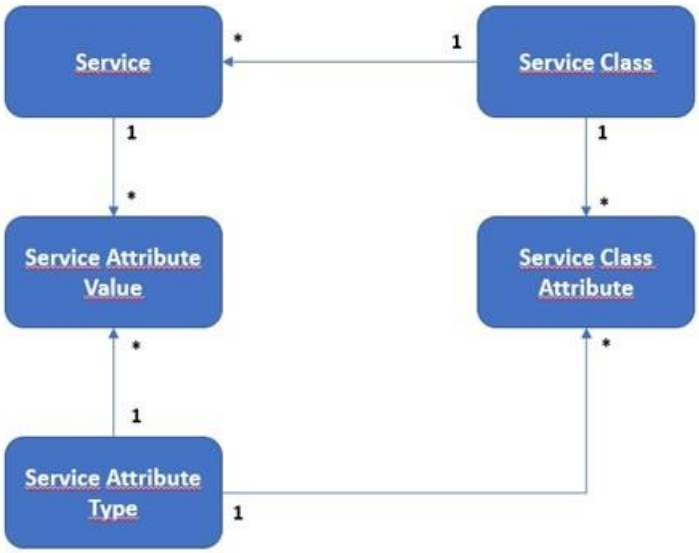

Figure 3. Class diagram of cloud services (adopted from [5])

The figure above shows the relationship among the different classes involved in the definition of the cloud services, as can be found in ACSmI.

The main ones are:

- *Service*: The different services that will be included in the DECIDE ACSmI Service registry.
- *Service Class*: This class is used by OPTIMUS when requesting for the corresponding service for the deployment. Different classes of services are Database, Storage, Virtual Machine.
- *Service Attribute Type*: This entity determines the attribute types that belong to each Service Class
- *Service Class Attribute*: This entity represents the different attributes that can be associated to the different service classes. The relationship between the "Service Attribute Type" and the "Service class" is made through "Service Class Attribute" entity as shown in Table 1.
- *Service Attribute value*: this is the value associated to an attribute for each service. There could be a preliminary list of values for each one of them. For example, for the "Region" attributes, the values could be "Ireland", "France", "US", "Germany", and so on.

Table 1. Attributes Vs cloud service Classes (adopted from [5])

Storage	DataBase	Virtual Machine
Region	Region	Region
Zone	Zone	Zone
Provider	Provider	Provider
Storage type	Database type	Virtual CPU cores
Storage subtype	Database technology	Frequency per core
Storage capacity	Data transfer IN	Memory
Storage data redundancy	Data transfer OUT	Instance storage
Availability	Virtual CPU cores	Optimized for
Request – Response time: Storage Performance	Database storage capacity	Public IP
Legal Level	Availability	Underpinning technology
Cost/Currency	Transaction Unit (DTU): Database performance	Availability
	Legal Level	Response time: Virtual Machine Performance
	Cost/Currency	Legal Level
		Cost/Currency

3 Modelling Deployment Schemas

Once the simulation process finishes (c.f. D3.8 [2] for a detailed explanation), the best deployment schemas are shown to the developer so that (s)he can confirm or select one of them. The selected schema will be the one that will be used for the deployment when ADAPT tool starts to deploy the application.

The structure of the deployment schema is also stored into the application description JSON file, as well as in the repository for historical deployments.

A *SchemaElement* represents the relationship between a specific cloud service and a microservice or a group of them. The figure 4 represents a *SchemaElement* and its properties.

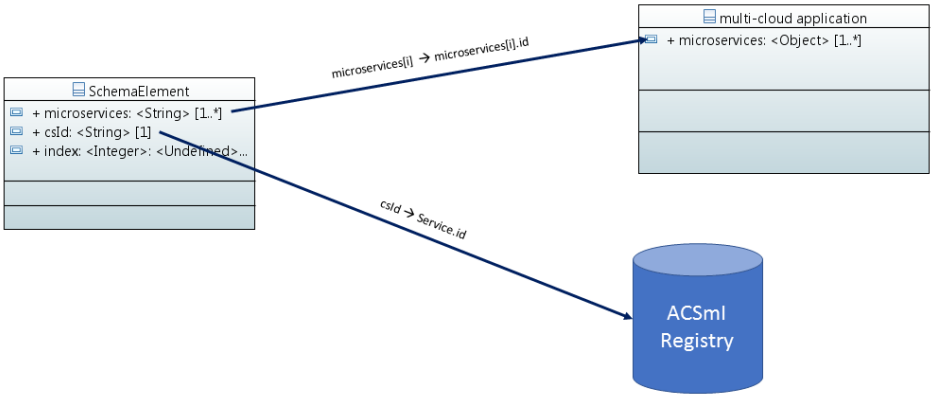

Figure 4. Schema Diagram

Each of the *SchemaElement* objects is composed of the following properties:

- *Index*: generated number to identify this association between the csId and the microservices. ACSmI uses it.
- *microservices*: Group of microservices id as represented in the Application Description. The ids of the microservices that should be deployed in the cloud service mentioned below.
- *csId*: cloud service (CS) id: the id of a cloud service selected. The information about this cloud service can be found in the ACSmI Service registry.

Equally, Non-Functional Requirements (NFRs) play an important role to identify the best deployment schema. When OPTIMUS builds the request to ACSmI in order to obtain the most appropriate cloud services, part of the features that those cloud services have to fulfil are indicated by the NFRs set by the developer.

The NFRs are managed by the NFR Editor in both the DevOps framework and the Eclipse plugin configurations. There are different types of NFR, as shown in the following Figure 5.

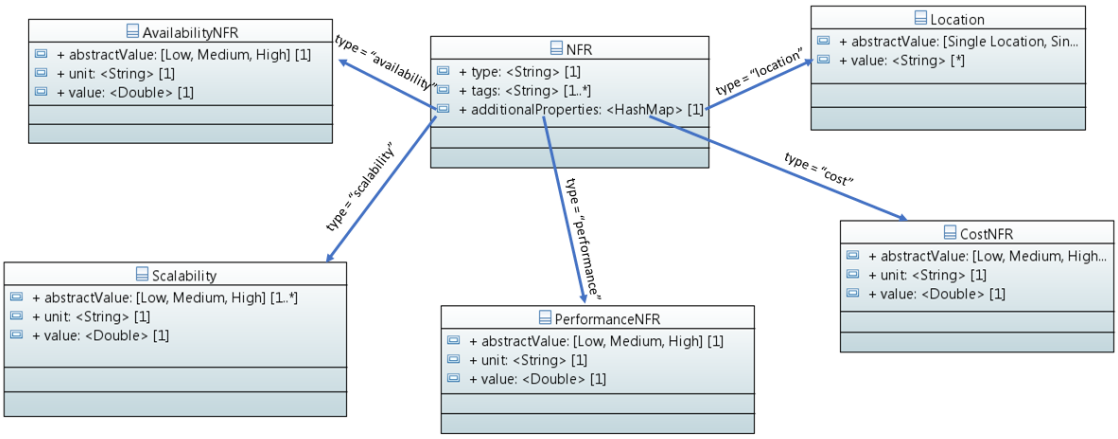

Figure 5. NFRs data structure

The "*tags*" property in the microservice class is related to the NFR that is defined with these same tags.

The NFR defined by a specific tag “moduleX”, will be associated to the microservices that have in the “*tags*” property this same value.

For example, if there is a unique NFR defined, the Availability NFR:

```
"nfrs" : [{
    "type" : "Availability",
    "tags" : [ "moduleX" ],
    "abstractValue" : "Medium",
    "value" : 98.0,
    "unit" : "percentage"
  } ],
```

If a microservice includes that same value in the “tags” property, it will mean that the developer associated that NFR and its value to that microservice:

```
"microservices" : [ {
    "id" : "eabad57d-7cae-4e2b-bebf-1d95b8534f53",
    "name" : "Queue-Master",
    "classification" : "Computing",
    "stateful" : false,
    "programmingLanguage" : "Java",
    "tags" : [ "moduleX" ],
    "publicIP" : false,
    "endpoints" : [ "http://mircroservice1.com/service" ],
    "deploymentOrder" : 0,
    "detachableResources" : [ {
      "id" : "e524abef-6814-4724-bad6-4ac048c917e0",
      "name" : "none",
      "db" : true,
      "sql" : false,
      "classification" : "storage"
    } ]
  } ]
```

4 Classification tool

The main objective of the OPTIMUS classification tool is to be the basis to select a group or class of the cloud services gathered into the ACSmI registry. These classification values are related to the cloud services class recorded in the ACSmI registry, and are designed in a previous step of profiling applications, with the aim of linking the microservices and the cloud services where they could be deployed on. The classification value provides a first filter for knowing those appropriate cloud services.

That mapping between the microservice and the cloud service where the microservice could be deployed on, is based on the classification explained in section 5.

Once that classification is made, the developer can launch the simulation process which will use this classification value for obtaining the group of cloud service that will take part of the selected deployment schema. The (group of) cloud services where a microservice could be deployed will depend on its classification. So, the process for requesting the proper cloud services to ACSmI will ask for a subset of the cloud services stored in the registry and then OPTIMUS will select the best ones by filtering on the NFRs and the characteristics related to the microservices to be deployed on them.

The access point to the classification tool will be trough OPTIMUS UI.

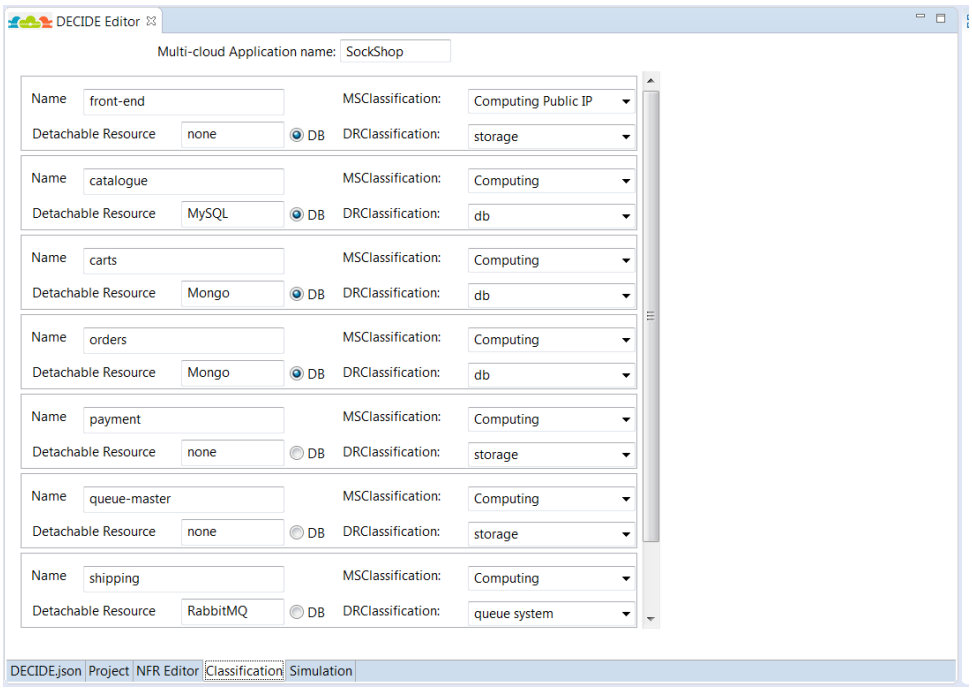

Figure 6. OPTIMUS classification tab in the Eclipse plugin of the tool.

The possible classification values for a microservice are currently identified as Computing, Computing Public IP, Storage and DB. The result of the classification is stored into the Application Description JSON file.

At this point of the project, the information about the microservices that can have an impact on their classification is related to the nature of that piece of software. Each microservice can have associated one or more detachable resources. The microservice would be the main software which the detachable resources depensd on, setting a whole element to be deployed.

The microservices can be classified as "Computer" (by default) or "Computing Public IP". The detachable resources that a microservice can have associated can be classified as "db" or "storage" (by default). Because these resources are considered associated to the main microservice, they will be deployed on the same cloud service as its main microservice.

A user manual of the classification tool is included in the deliverable D3.8 [2].

The classification tool is a component in the OPTIMUS architecture as it is shown in Figure 7

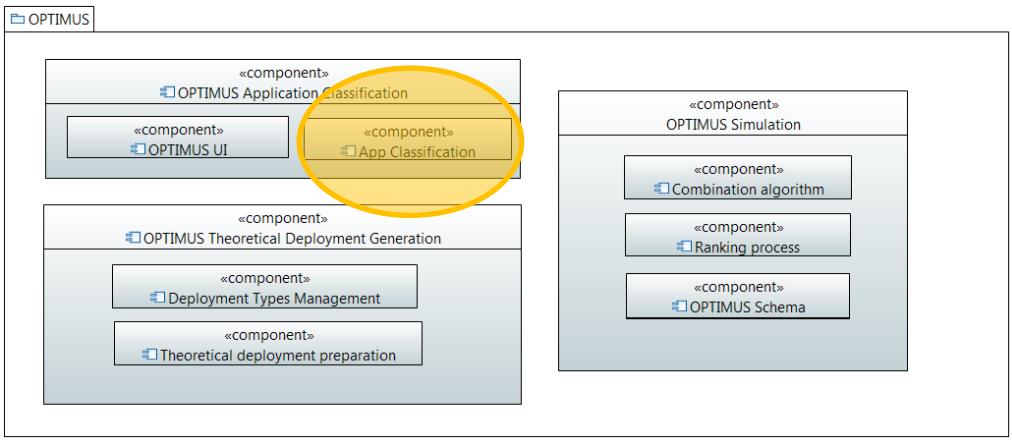

Figure 7. Classification module in OPTIMUS architecture.

The requirements for the classification tool and their status are described in Table 2.

Table 2. Requirements for OPTIMUS classification tool

Req. ID	Description	Status
WP3-PROFI-REQ1	Load/read information about the application (components).	Status: Implemented in M12 version.
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 comparing it with the information about the (component) application.	It will be completely implemented in M30. Status: It is an incremental requirement. The current version provides a classification but this value can evolve, when analyzing this current version and improving this classification depending on the source of the information
WP3-PROFI-REQ3	Request the developer to confirm the classification	Implemented in M24 Status: In this prototype the developer confirms the classification made when the developer launches the simulation with the assigned classification.

Req. ID	Description	Status
WP3-PROFI-REQ4	Store the information about the classification made.	Status: Implemented in M12 The task of storing the classification is implemented since M12. The classification value stored was a first version of the final value related to the WP3-PROFI-REQ2 requirement.
WP3-PROFI-REQ5	Mechanisms for update the "stereotypes of the components" information	Status: It will be implemented in M30

The current version of the tool is integrated into the OPTIMUS tool plugin [2].

5 Mapping microservices and Cloud services

This mapping remains almost invariable with what was presented in the previous version of this document [6].

Profiling an application or a microservice is a previous step that was performed in a previous stage of the project, namely when the classification tool was designed and its detailed explanation can be found in deliverable D3.4 [6].

Taking into account that this is the intermediate version for this subject, the profiling part was described more deeply in that previous version, and the mapping section shows how that mapping is done once the profiling is previously made.

Profiling in the context of DECIDE means “necessities for deployment”so as to identify the type of the cloud service that the microservice needs in order to be properly deployed.

Considering that microservices are going to be deployed in a cloud service of a specific provider, the profiling phase establishes the relationship between the types of necessities and these cloud services and their providers.

As explained above, microservices can be classified as Computing or Computing Public IP, considering their nature of executable software made by the developer, and sometimes requiring direct access to them by the users.

The detachable resources are components used for storing data associated to those processes, like a Database (DB) or Disk Storage.

Next, the classification is detailed:

- Computing.

  Computing classification is assigned to components (microservices) that are implemented by a developer and need some computing resources to be executed.

  The classes of cloud services related to “Computing” are Virtual Machines.

- Computing with public IP.

  This classification is for computing components that need a public IP to expose their services to the external network. This is typically used for web applications.

  The classes of cloud services related to “Computing with public IP” are Virtual Machines, but the provider has to make available an external access to the users.

- Storage.

  The Storage classification covers the detachable resources associated to a microservice that need to store a big quantity of data, like Big data analytics, backups etc.

- DB

  The DB classification includes components that access databases, as relational database or NoSQL database.

  The related cloud services are database services that are managed by the provider and can be launched with no need of any installation from the user side.

6 Modelling simulation data

Once the classification is finished, the next steps are related to the OPTIMUS simulation tool and, for running an appropriate algorithm, some additional structures will be needed.

The information to build the request for ACSmI is obtained by parsing the application description JSON file. The filter where all the criteria are included, will be created taking into account the classification, the cloud services classes related to it, the NFRs that the deployment has to acomplish, and the properties associated to each of the miroservices that form part of the multi cloud application. The M24 version of OPTIMUS does not include these structures. They are an approach for the next version of the tool.

For each microservice involved, the algorithm will create several structures linked between them. Figure 8 shows the first approach to these structures.

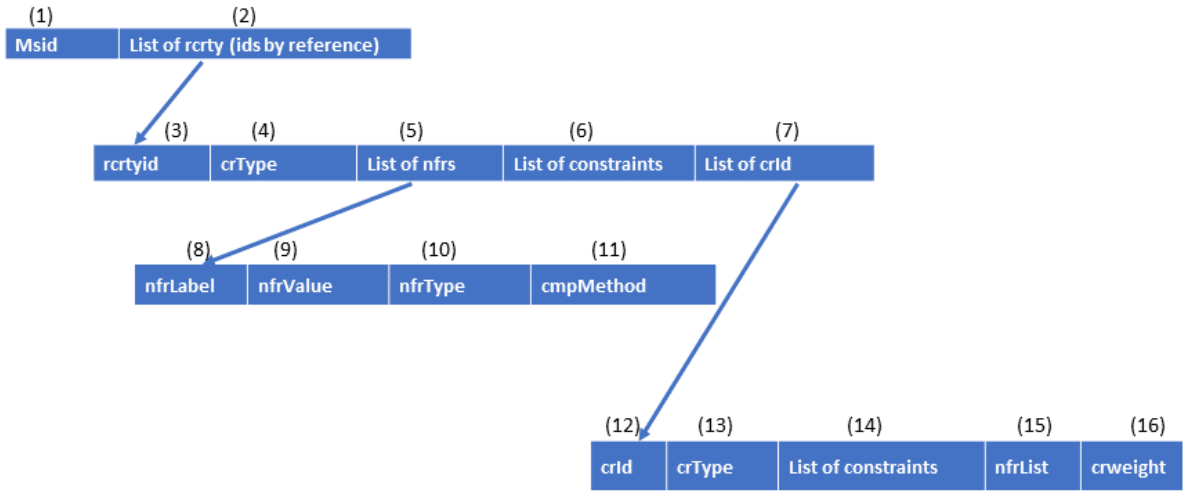

Figure 8. Simulation Structures.

(1) *Msid* : microservice identifier Id, from the application description JSON file
(2) *rcrtyList*: List of identifers of required resources types, from ACSmI registry. For example if the microservice manages a DB, it could be deployed as a DB service provided by a specific vendor or could be deployed in a VM being responsability of the developer to install the DB in the machine.
(3) *rcrtyid*: required cloud service type identifyers, from ACSmI registry
(4) *crtype*: type of the cloud service or cloud resource from ACSmI registry (classid property)
(5) *nfrList*: list of NFRs set by the developer (for the microservice or inherited from the application level), from the application description JSON file. These NFRs have to be fulfiled by the cloud services considered as candidates (3).
(6) *constraintsList*: List of characteristics needed, provided by the developer (number of cores, Public IP, …), from the application description JSON file. These constraints have to be fulfiled by the cloud services considered as candidates (3).
(7) *crList*: List of cr (cloud resources or cloud services) obtained by asking ACSmI, from ACSmI registry
(8) *nfrlabel*: name of the nfr (availability, performance, …), from the application description JSON file
(9) *nfrValue*: the nfr value for a microservice established by the developer or obtained from the application level, from the application description JSON file
(10) *nfrType*: the type of the value, for example %, from the application description JSON file

(11) *cmpMethod*: the criteria to compare this nfr objective to the nfr provided by the cloud service (or resource). For example, if the nfr is availability and the value is 90%, then the algorithm will search for greater values than that, and if the nfr is cost the algorithm will search for smaller values. This information will be stored in a resource as a configuration aspect.
(12) *crId*: Cloud Resource Id, from ACSmI registry
(13) *crtype*: type of the cloud service or cloud reource in ACSmI registry (classid)
(14) *constraintsList*: list of characteristics of the cloud resource obtained from ACSmI, excluding those considered as nfrs.
(15) *nfrList*: list of nfr, but this nfrs are the real nfrs obtained from ACSmI
(16) *crweight*: numeric value assigned to this instance of cloud resource, based on the level of fulfilment of the requested nfrs. The NFRs assigned to the Cloud Resource will be compared to the NFR objectives, applying the comparison Method (field 11) established by configuration.

When requesting ACSmI for cloud services (or cloud resources), the characteristics informed by the developer (constraints) will be mandatory to be fulfilled. The NFRs will be 'desired' but if the return message is null, that is, there are no resources matching those constraints, then the algorithm will select the closest ones.

Once the weight (see field 16, *crweight*, in the previous list) is set by OPTIMUS to each of the possible associations between the microservice and the cloud resource, the algorithm will perform several permutations. This weight will be established taking into account the level of fulfillment of the NFRs and constraints set by the developer. The aggregated weight, as a weight assigned to each of the obtained combinations of individual associations, will be the addition of the individual *rcrty* weights in this approach. These permutations will be sorted and OPTIMUS will select the five best to show them to the developer in the simulation tab of the OPTIMUS eclipse plugin.

During the simulation process, as an example of the data used by the algorithm, the information stored could be as shown in the Tables 3 and 4 below:

Table 3. Example of values associating microservices to cloud resources.

Msid (1)	Rcrtyid (3)	NfrList (5)	ConstraintsList (6)	CrId (12)
ms1	rcrty1	List of nfrs (json) for ms1	List of constraints (json) for ms1	cr1
ms1	rcrty1	List of nfrs (json) for ms1	List of constraints (json) for ms1	cr2
ms1	rcrty2	List of nfrs (json) for ms1	List of constraints (json) for ms1	cr3
ms2	rcrty1	List of nfrs (json) for ms2	List of constraints (json) for ms2	cr2
ms2	rcrty1	List of nfrs (json) for ms2	List of constraints (json) for ms2	cr4
ms2	rcrty2	List of nfrs (json) for ms2	List of constraints (json) for ms2	cr3
ms2	rcrty2	List of nfrs (json) for ms2	List of constraints (json) for ms2	cr5
ms2	rcrty2	List of nfrs (json) for ms2	List of constraints (json) for ms2	cr6
ms2	rcrty3	List of nfrs (json) for ms2	List of constraints (json) for ms2	cr7
ms2	rcrty2	List of nfrs (json) for ms2	List of constraints (json) for ms2	cr8
…..				

There are several possibilities for associating a microservice with the cloud services where it could be deployed. Each of these possibilites will have a weight, based on how the NFRs and the constraints (characteristics of the microservice) are fulfiled by the cloud service or resource indicated.

Table 4. Weight of each cloud resource

CrId (12)	Rcrtyid (3)	ConstraintsList (14)	NfrList (15)	Crweight (16)
cr1	rcrty1	List of cr1 constraints from ACSmI	List of cr1 nfrs from ACSmI	cr1weight
cr2	rcrty1	List of cr2 constraints from ACSmI	List of cr2 nfrs from ACSmI	cr2weight
cr3	rcrty2	List of cr3 constraints from ACSmI	List of cr3 nfrs from ACSmI	cr3weight
cr4	rcrty1	List of cr4 constraints from ACSmI	List of cr4 nfrs from ACSmI	cr4weight
cr5	rcrty2	List of cr5 constraints from ACSmI	List of cr5 nfrs from ACSmI	cr5weight
cr6	rcrty2	List of cr6 constraints from ACSmI	List of cr6 nfrs from ACSmI	cr6weight
cr7	rcrty3	List of cr7 constraints from ACSmI	List of cr7 nfrs from ACSmI	cr7weight
cr8	3crty3	List of cr8 constraints from ACSmI	List of cr8 nfrs from ACSmI	cr8weight

The result of the permutations - understanding this process as the result of grouping each of the individual possibilities (microservice and cloud service) with the rest of individual possibilites for the rest of the microservices - the aggregation of the weights and the final sorted list will be calculated based on the structures shown on the tables above.

When the information ARCHITECT stores into the application description JSON file is more defined, OPTIMUS will then be able to decide on more appropriate deployment schemas. For the DECIDE M24 prototypes, the information that ARCHITECT saves into the Application Description JSON file is a description about the pattern or patterns selected. WP3 has a pending task to design and analyze the impact of those selected pattern into the deployments to be executed.

7 Conclusions

This document has presented the current status of the data models related to the multi-cloud application and how their properties impact onto the best deployment schema selection. There have not been many critical modifications from the previous version in the M12 version but rather some updates due to the needs of the each of the tools. The next version of this document will show the final models handled when a developer uses DECIDE to deploy his or her multi-cloud application in the best possible way.

The conclusions of the previous deliverable D3.4, Initial profiling and classification techniques [6], remain valid since the DECIDE framework keeps handling a shared data structure across all DECIDE tools, and hence does not need a specific editor for the developer to model his applications or deployments.

The possibility of exporting the application data onto a CAMEL model has not been explored yet. Althought it could be a valuable aspect for application owners, in case they want to migrate to a system with this application representation; the decision of implementing this feature has been postponed until the end of the project.

8 References

[1] DECIDE Consortium, "D2.5: Detailed architecture v2," 2018.

[2] DECIDE Consortium, "D3.8: Intermediate DECIDE OPTIMUS," 2018.

[3] DECIDE Consortium, "D3.14: Intermediate multi-cloud native application composite CSLA definition," 2018.

[4] DECIDE Consortium, “GRANT AGREEMENT - Annex I: Description of Action,” 2016.

[5] DECIDE Consortium, "D5.3: Intermediate Advanced Cloud Service meta-Intermediator," 2018.

[6] DECIDE Consortium, "D3.4 Initial profiling and classification techniques," 2017.