Deliverable D2.5

Detailed architecture v2

Editor(s):	Juncal Alonso
Responsible Partner:	TECNALIA
Status-Version:	Final - v1.0
Date:	31/10/2018
Distribution level (CO, PU):	PU

Project Number:	GA 731533
Project Title:	DECIDE

Title of Deliverable:	D2.5 – Detailed architecture v2
Due Date of Delivery to the EC:	31/10/2018

Workpackage responsible for the Deliverable:	WP2 – DECIDE requirements and DECIDE solution integration
Editor(s):	TECNALIA
Contributor(s):	Juncal Alonso, Gorka Benguria, Marisa Escalante, Maria Jose Lopez, Iñaki Etxaniz, Alberto Molinuevo, Leire Orue-Echevarria (TECNALIA), Javier Gavilanes, Antonio Fernandez (Experis), Lorenzo Blasi, Paolo Barone (HPE), Simon Dutkowski, Kyriakos Stefanidis (Fraunhofer), Vitalii Zakharenko, Andrey Sereda(CB)
Reviewer(s):	TECNALIA
Approved by:	All Partners
Recommended/mandatory readers:	WP3, WP4, WP5

Abstract:	This is the second release of the detailed design of the DECIDE framework including its components, modules, interfaces updated according to the comments received from the use cases implementation.
Keyword List:	Architecture, components, technical design, interfaces, interoperability, deployment, DevOps.
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	26/09/2018	First draft version including TOC and assignments	TECNALIA
v0.2	05/10/2018	Updated and included content in several sections: 1, 2.1, 2.2.	TECNALIA
v0.3	12/10/2018	Included content in sections: 2,3,4.3.3 Annex 1.	TECNALIA
v0.4	13/10/2018	Included content in sections 4.2.1, 4.4.1,5.2	EXPERIS, HPE
v0.5	18/10/2018	Included new sub-section 1.2-Innovation of the deliverable Included content in sections 4.3.1, 5.1 and Annex 1	TECNALIA, HPE
v0.6	22/10/2018	Included content in sections 4.1.1,4.1.2,4.3.2, 4.4.1, 5.1and Annex 1	Fraunhofer, Experis
v0.7	25/10/2018	Included content in section 5.1. Addressed internal review comments	CB, TECNALIA
v0.8	25/10/2018	Updated Annex 1.	Fraunhofer
v0.9	29/10/2018	New sub- section introduced: 3. Re-deployment workflow	TECNALIA
V1.0	30/10/2018	Ready for submission	TECNALIA

Table of Contents

Table of Contents 4
List of Figures 5
List of Tables 6
Terms and abbreviations 7
Executive Summary 8
1 Introduction 9
1.1 About this deliverable 9
1.2 Innovation of this deliverable 9
1.3 Document structure 10
2 Overview of the DECIDE integrated conceptual architecture 11
2.1 Multi-Cloud classification 11
2.2 DevOps and DECIDE extended DevOps 13
2.3 DECIDE Tools for multi-cloud applications architecting 14
2.3.1 NFR Editor 14
2.3.2 ARCHITECT 15
2.4 DECIDE Tools for multi-cloud applications continuous development and integration 15
2.4.1 DevOps framework 15
2.5 DECIDE Tools for multi-cloud applications (pre) deployment 15
2.5.1 OPTIMUS 15
2.5.2 App Controller 15
2.6 DECIDE tools for multi-cloud applications continuous delivery 16
2.6.1 ACSmI 16
2.7 DECIDE tools for multi-cloud applications continuous adaptation 16
2.7.1 ADAPT deployment and monitoring 16
2.7.2 MCSLA Editor 16
3 Alternative workflows 17
3.1 Re-deployment workflow 17
3.2 Starting the workflow from OPTIMUS 20
3.3 Starting the workflow from ACSmI 22
3.4 Starting the workflow from ADAPT 23
4 Detailed DECIDE integrated architecture 24
4.1 DECIDE tools for multi-cloud applications design and development 24
4.1.1 NFR editor 24
4.1.1.1 Structural description 24
4.1.1.2 Behavioural description 25
4.1.2 ARCHITECT 26

4.1.2.1 Structural description 26
4.1.2.2 Behavioural description 28
4.2 DECIDE tools for multi-cloud applications continuous integration and testing 29
4.2.1 DevOps framework 29
4.2.1.1 Structural description 29
4.2.1.2 Behavioural description 31
4.3 DECIDE tools for multi-cloud applications (pre) deployment 32
4.3.1 OPTIMUS 32
4.3.1.1 Structural description 32
4.3.1.2 Behavioural description 33
4.3.2 Application controller 34
4.3.2.1 Structural description 34
4.3.2.2 Behavioural description 36
4.3.3 ACSmI 36
4.3.3.1 Structural description 36
4.3.3.2 Behavioural description 40
4.4 DECIDE Tools for multi-cloud applications continuous operation 42
4.4.1 ADAPT 42
4.4.1.1 Structural description 42
4.4.1.2 Behavioural description 43
4.4.2 MCSLA Editor 45
4.4.2.1 Structural description 45
4.4.2.2 Behavioural description 47
5 DECIDE tool suite deployment 48
5.1 DECIDE tools deployment options 48
5.2 Information exchange between DECIDE tools 51
6 Conclusions 52
7 References 53
Annex 1: App Description 55

List of Figures

FIGURE 1. DECIDE INTEGRATED GENERIC ARCHITECTURE. 11
FIGURE 2. EVOLUTION IN SOFTWARE DEVELOPMENT AND DEPLOYMENT ARCHITECTURES 12
FIGURE 3. DECIDE EXTENDED DEVOPS APPROACH 14
FIGURE 4. DECIDE WORKFLOW [1] 17
FIGURE 5. DECIDE RE-DEPLOYMENT WORKFLOW. 18
FIGURE 6. NFR EDITOR COMPONENT DIAGRAM 24
FIGURE 7. NFR EDITOR EXTERNAL INTERFACES COMPONENT DIAGRAM 25

FIGURE 8. NFR EDITOR SEQUENCE DIAGRAM. 26
FIGURE 9 COMPONENTS OF ARCHITECT 27
FIGURE 10. USE CASES OF ARCHITECT 28
FIGURE 11 CREATE NEW DECIDE PROJECT, SEQUENCE DIAGRAM. 29
FIGURE 12. DEVOPS FRAMEWORK COMPONENT DIAGRAM 30
FIGURE 13. DEVOPS FRAMEWORK INTERFACES DIAGRAM 31
FIGURE 14. DEVOPS FRAMEWORK SEQUENCE DIAGRAM 31
FIGURE 15. OPTIMUS COMPONENT DIAGRAM 32
FIGURE 16. OPTIMUS EXTERNAL INTERFACES COMPONENT DIAGRAM 33
FIGURE 17. OPTIMUS SEQUENCE DIAGRAM 33
FIGURE 18. COMPONENT DIAGRAM FOR APPLICATION CONTROLLER 35
FIGURE 19. SEQUENCE DIAGRAM FOR WRITING AND READING DEPLOYMENT CONFIGURATION 36
FIGURE 20. ACSMI HIGH LEVEL ARCHITECTURE 38
FIGURE 21. ACSMI EXTERNAL INTERFACES 40
FIGURE 22. ACSMI SEQUENCE DIAGRAM 41
FIGURE 23. ADAPT COMPONENT DIAGRAM AND INTERFACES 42
FIGURE 24. INTERACTIONS OF ADAPT WITH OTHER DECIDE TOOLS 44
FIGURE 25. COMPONENT DIAGRAM FOR MCSLA EDITOR 46
FIGURE 26. SEQUENCE DIAGRAM FOR CREATING AN MCSLA 47

List of Tables

TABLE 1. ALTERNATIVE WORKFLOW 1: OPTIMUS + ACSMI + ADAPT + MCSLA EDITOR 20
TABLE 2. ALTERNATIVE WORKFLOW 2: ACSMI + ADAPT + MCSLA EDITOR 22
TABLE 3. ALTERNATIVE WORKFLOW 3: ADAPT + MCSLA EDITOR + ACSMI MONITORING 23
TABLE 4. DECIDE KRS' DEPLOYMENT OPTIONS 48
TABLE 5. APPLICATION DESCRIPTION MODEL 55
TABLE 6. MCSLA OBJECTIVES OBJECT DESCRIPTION (NESTED ELEMENTS FOR "OBJECTIVES") 64
TABLE 7. NESTED ELEMENTS FOR VIOLATIONTRIGGERRULE 65
TABLE 8. MCSLA METRIC DATA MODEL FOR MONITORING 65
TABLE 9. NESTED ELEMENTS FOR EXPRESSION 67
TABLE 10. NESTED ELEMENTS FOR PARAMETER 68
TABLE 11. NESTED ELEMENTS FOR RULE 68
TABLE 12. NESTED ELEMENTS FOR REMEDY 69

Terms and abbreviations

API	Application Programming Interface
App	Application
APP	Application
CSP	Cloud Service Provider
DB	Data Base
DevOps	Development and Operation
DoA	Description of Action
EC	European Commission
IDE	Integrated Development Environment
KR	Key Result
Mx	Month x, where x represents a number
MCSLA	Multi Cloud Service Level Agreement
MTTR	Mean Time To Recovery
NFR	Non-Functional Requirement
RCP	Rich Client Platform
SDK	Software Development Kit
SLA	Service Level Agreement
SPA	Single Page Application
Sw	Software
UI	User Interface
WP	Work Package

Executive Summary

This document provides the specification of the DECIDE integrated architecture at month 23, a second and final specification[1] of the entire DECIDE integrated architecture, aiming to ensure a smooth integrated specification at conceptual, functional and technical level of the different building blocks, i.e. Key Results (KR), that constitute the DECIDE tool-suite. DECIDE tool-suite enables users (developers and operators) of multi-cloud applications to implement the DECIDE extended DevOps approach, by providing a comprehensive set of tools that assists users to complete the DECIDE lifecycle [1]. The presented DECIDE extended DevOps approach comprises the following phases and supporting tools:

- Multi-cloud applications architecting: Supported by the following DECIDE tools: NFR Editor and ARCHITECT.
- Multi-cloud applications continuous development and integration: Supported by the following DECIDE tools: DevOps Framework.
- Multi-cloud applications (pre)deployment: Supported by the following DECIDE tools: OPTIMUS and App Controller.
- Multi-cloud applications continuous delivery: Supported by the following DECIDE tools: ACSmI.
- Multi-cloud applications continuous adaptation: Supported by the following DECIDE tools: ADAPT and MCSLA Editor.

DECIDE comprises diverse technical and scientific activities that contribute altogether to the jointly materialization of the DECIDE outcomes. However, a successful instantiation of these techniques and tools requires an integration task force, materialized in this document, which aims to converge these different conceptual and technical approaches and the earlier detection and fixing of potential misalignments that may occur during the initial design and development phases.

In this scope, the global and integrated architecture described in this document aims to: a) provide an overall and comprehensive conceptual and functional description of the DECIDE tool-suite in their current state, b) ensure a smooth conceptual and technical interoperability among DECIDE tools that guarantees a correct instantiation of the DECIDE extended DevOps approach, c) describe the tools interoperability needs in terms of messages consumed and provided by each tool, d) provide a detailed structural and behavioural view of the different tools, grouped on packages of related functionality, and e) expose and discuss the available possibilities for the deployment of DECIDE tool-suite. The current document (which has been created and delivered in the context of WP2) is describing at general level the different components inside the DECIDE tool suite. The details of the components are described in the deliverables to be generated in the different technical WPs (WP3, WP4 and WP5). Both the novel approaches for the multi-cloud applications and DECIDE extended DevOps and how DECIDE tools supports them are the main innovations presented in the current deliverable.

This document is the reference for the next DECIDE tools implementation (new versions to be released in M24 and 30) and will be continuously checked, used, aligned and updated as a result of other DECIDE activities during final development of their components and tools to ensure that they are conceptually and technically aligned and compatible with others as these may need to interoperate with. To this respect the Application Description (Annex 1) is still being updated as new needs for the tools arise. This annex will be continuously updated and used as the reference for the last updated Application Description schema.

[1] The firstspecification was released on M12. [2]

1 Introduction

1.1 About this deliverable

This document provides the M23 specification of the DECIDE integrated architecture. This architecture collects and describes the main functional key results, tools and components that constitute the DECIDE tool-suite, that is, the comprehensive set of tools created by DECIDE, for developers and operators of multi-cloud applications to apply the DECIDE DevOps extended approach. This document provides an updated version of the DECIDE architecture delivered in M12 [2].

DECIDE Key Results (and related tools and components) are described as functional blocks, including structural and behavioural aspects. The descriptions of the components included in this document aim to provide a general overview of the functionalities of the key results and the interactions between them. The internal representation of the tool (i.e. its internal technical specification) is not addressed in this document but left to further dedicated technical reports (for each tool) that describe them, along with the actual implementation in the different releases [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16].

This overall description aims to prove a complete and correct coverage of the DECIDE extended DevOps approach, providing means (in form of tools and components) to support each phase of the approach.

Special attention is also given to address interoperability between tools, that is, the dependencies between the tools and the technical ways in which those dependencies are managed. Interoperability implies taking care of different dimensions which allows us to identify and define:

- Messages exchange, including compatibility at data content (semantic alignment), data format, serialization format, etc.
- User-driven interaction model, as we foresee most of interoperable situations driven by the end-users of the tools.

This analysis enables to obtain an earlier detection of possible conceptual and technical misalignments (among tool providers), either at conceptual level (i.e. semantics), or at functional and a technical level. Based on this analysis, the document proposes a harmonized conceptual, functional and technical common view that removes these misalignments and enables the specification, by each tool provider, of an interoperable toolset design. In particular, the dependencies amongst tools are identified by detecting the products they consume (as inputs) and produce (as outputs), which would be, in turn, produced and consumed by other tools.

The document also presents and discusses the different possibilities for the deployment of the tools to be implemented in DECIDE and for the whole DECIDE Framework.

In this report, the specification of the Application Description is included as an Annex. This specification describes all the parameters included in the Application Description. The Application Description is a file used during the DECIDE workflow to exchange relevant information about the status of the Application in each step of the workflow. The Application Description is updated and read by every KR in DECIDE, to perform their corresponding actions.

1.2 Innovation of this deliverable

This deliverable introduces the generic architecture of the DECIDE framework and its corresponding KRs. Each of the KRs includes innovative aspects that are expressed in their corresponding deliverables. At general level, this deliverable presents how the DECIDE tools support the DevOps philosophy for the specific case of native multi-cloud applications.

Therefore, this is the main innovative point of the current report. The proposed DECIDE Extended DevOps concept extends the traditional DevOps cycle with new phases (such as continuous architecting, continuous pre-deployment, continuous delivery and continuous adaptation) which aims to support the specific needs of multi-cloud native applications.

1.3 Document structure

This document is structured as follows.

Section 2 provides an overall conceptual and functional introduction to the entire DECIDE tool-suite, an introduction to the DECIDE multi-cloud concept and to the DECIDE proposed extended DevOps approach.

Section 3 provides a detailed description of each DECIDE tool, individually and in the scope of the interactions with other tools in DECIDE tool-suite. The description is both structural -i.e. component dependencies, required and provided interfaces, etc.- and behavioural -i.e. temporal ordered interactions-, attending mainly to interoperability concerns.

Section 4 describes the deployment alternatives for the DECIDE tools, both as individual components and as an ecosystem as a whole.

Section 5 presents the conclusions of the document.

In the Appendix of the document, the current version of the App Description, which is the main mechanism for the interchange of information between tools in DECIDE, is presented.

2 Overview of the DECIDE integrated conceptual architecture

This section sketches the structural view of the DECIDE tool-suite architecture. More elaborated behavioural and structural views of the DECIDE tool-suite architecture will be presented in next sections.

This DECIDE architecture sketch (figure 1) is structured in blocks that correspond to the main DECIDE key results (KRs) and are co-located in the extended DevOps phases for multi-cloud applications as defined in section 2.2, i.e. 1- Architecting, 2- Continuous development and testing, 3- pre-deployment, 4- continuous delivery and 5- Continuous adaptation, to introduce the DECIDE KRs. Some of the KRs (i.e. ACSmI) support several phases of the DECIDE DevOps extended approach. In the figure below two elements are depicted in a different way to stress their singularity:

- *DevOps framework:* DevOps framework is one of the KRs of DECIDE. It is singular because, on one hand, it includes the necessary existing tools for development and integration (i.e. software repository, software development Kit, IDEs, etc.) and on the other hand, it provides the means to integrate the rest of the KRs in a unique stable toolkit (UIs, actions handling, sensitive data management and storage, etc.). More information about the DevOps framework is provided in section 4.2.1.
- *App Description and Application Controller*: Application Description (App Description from now on) is not a DECIDE KR, tool or component as such. It is a file where the actual status of the application is described. This file is used for the different KRs in DECIDE to store/acquire relevant information with respect to the application status needed to the correct operation of the different tools. App Controller is a java library used to access, read, and update information in the App Description. More information about the Application Description is provided in section 5 and in Annex 1. More information about the Application Controller is provided in section 4.3.2.

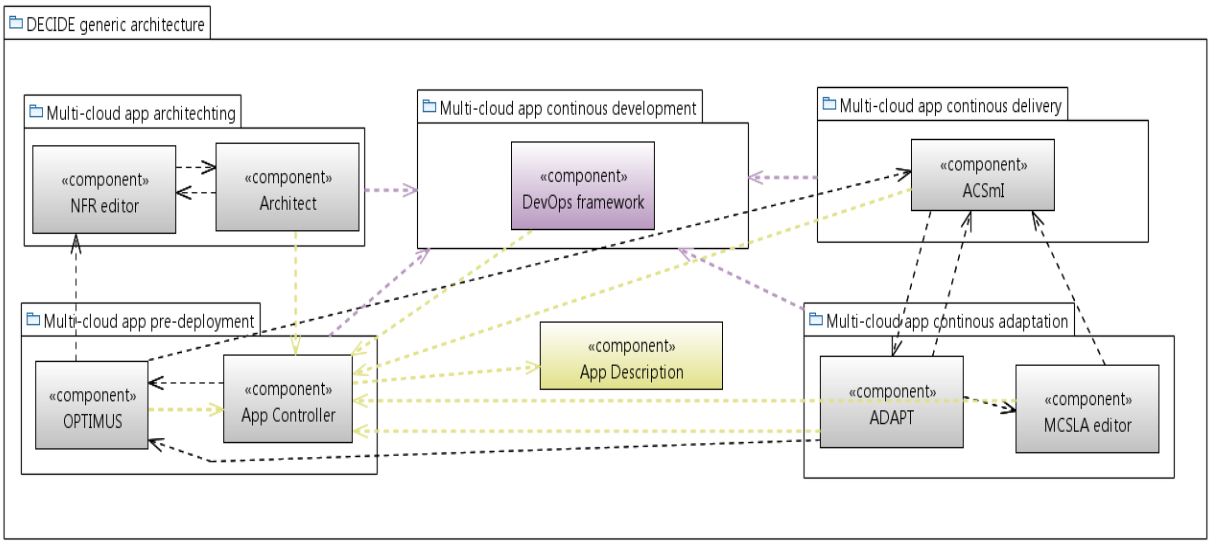


Figure 1. DECIDE integrated generic architecture.

2.1 Multi-Cloud classification

In the context of DECIDE a multi-cloud application is defined as a set of components distributed across heterogeneous cloud resources but that still succeed in interoperating as a single whole.

As described in D2.1 [1], multi-cloud is the use of multiple computing services for the deployment of a single application or service across different cloud technologies and/or Cloud Service providers. This may consist of PaaS, IaaS and SaaS entities in order to deliver an integrated end to end solution

This definition of multi-cloud, when referring to the resources where the different components are deployed, includes services which are in dispersed cloud providers or different cloud platforms (regardless of vendor) [17]:

- Deployment of services across multiple geographically dispersed cloud service providers.
- Deployment of services across different cloud technologies within a single cloud service provider.
- Deployment of services within a single cloud service provider in one technology.

The next figure a depicts the evolution in software development and deployment architectures.

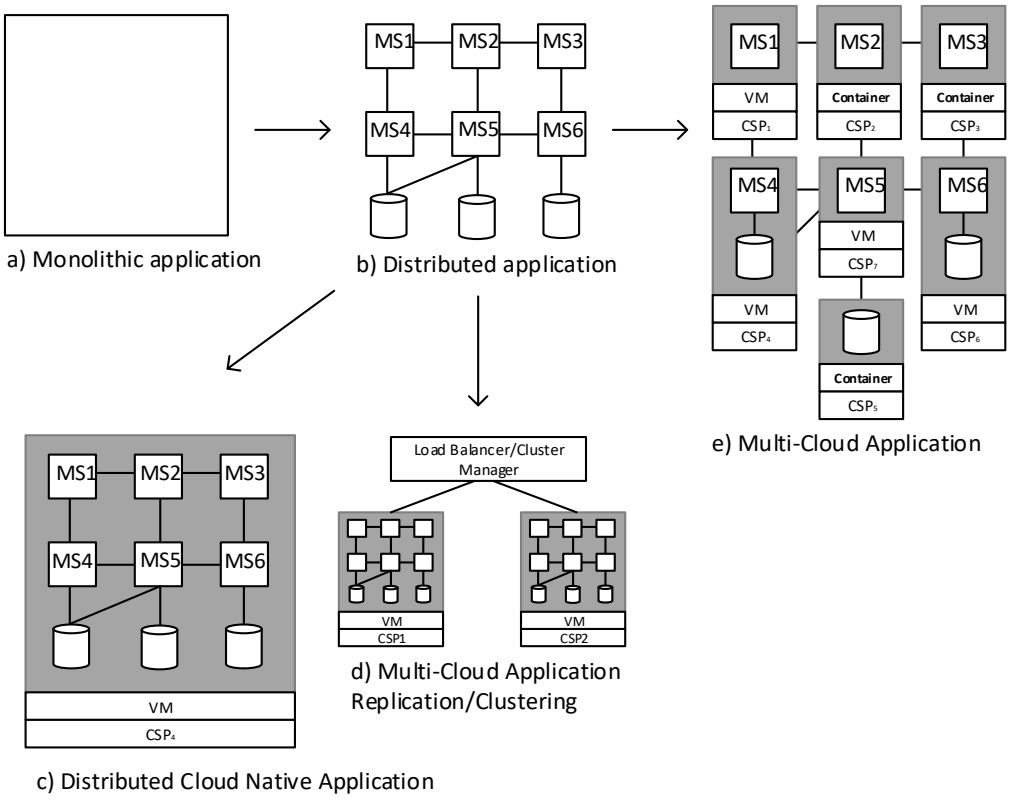


Figure 2. Evolution in software development and deployment architectures

This figure, already presented in D3.1 [18] presents the large difference between cloud native application and multi-cloud native architectures and deployments. It starts off with depicting a monolithic application (a)); b) depicts a distributed application in the traditional sense; c) depicts a microservices-based cloud native application; d) depicts a distributed application replicated or scaled across two CSPs and, finally, d) depicts a multi-cloud application's architecture and deployment as defined by the DECIDE project.

With this strategy[2], i.e. multi-cloud architecture and its deployment, considerations must be made regarding the architectural challenges and decisions that allow an application with its microservices to be seamlessly deployed and adapted across different CSPs.

The challenges that arise when designing a multi-cloud application are listed below, and form the basis for all considerations when designing an application in the context of DECIDE:

- Resilience and portability of the components or microservices of an application; when porting processes across clouds, MTTR must be decreased, and disconnected scenarios and faults have to be avoided. In addition, cost effective deployment of the application, by abstracting from cloud vendor specifics and without having to manually adapt to new interfaces must be given.
- Respecting of the applications defined NFRs.
- The applications components (e.g. microservices) should work together in an integrated manner. Microservices' endpoints must be managed and discoverable in case of switching hosts (IP addresses).
- Just as the portability of microservices, data migration or replication should be easily handled and not pose a problem
- The use of provider specific SaaS and IaaS services, because of each service providers intricacies (e.g. different APIs, data storage), should be possible.
- Dynamic re-configuration of the application properties should be possible.

2.2 DevOps and DECIDE extended DevOps

DevOps is a set of practices that automates the processes between software development (Dev) and IT teams (Ops) so they can build, test, release and deploy software applications more quickly, reliably and continuously. In traditional DevOps approaches, IT roles are merged and communication is enhanced to improve the production release frequency and maintain software quality [19]. The foreseen benefits of the DevOps philosophy include increased trust, faster software releases, automated testing and the ability to solve critical issues quickly.

When applying this philosophy to multi-cloud applications (see previous section) some shortcomings arise. These are caused by the peculiarities when developing, deploying and operating such multi-cloud applications that have not been deeply analysed nor supported by current DevOps solutions [20]:

- Applications need to be responsive to hybrid/multi-cloud model scenarios, in which an application that is executing in a concrete set of cloud services bursts into a new one when the working conditions are not met. This implies that the application architecture shall be re-designed to be "multi-cloud" aware, simplifying the cloud application assembly and the deployment process.
- Means shall be provided to manage and assess cloud deployment alternatives to better support cloud re-deployment decisions. This implies profiling and classifying application components and cloud nodes, as well as analysing and simulating the behaviour of the application to support the deployment decision making process, considering additional factors such as Non-Functional Requirements (NFR), namely performance, availability, localization, cost, or risks associated with the change of cloud resources. Multi-cloud has value only when the right providers are selected, whether public or private (combined into different cloud deployment models), to meet functional and NFR. But the manual selection and combination of those Cloud Services to create the best deployment scenario may imply huge effort, time and knowledge needs.

[2] A pre-requisite for a multi-cloud strategy is a distributed application that is loosely coupled with stateless properties.

- Existing cloud services shall be made available dynamically, broadly and cross border. so that software providers can re-use and combine cloud services, assembling a dynamic and re-configurable network of interoperable, legally secured, quality assessed (against SLAs) single and composite cloud services.

  To overcome these shortcomings, DECIDE project proposes an extension of the "traditional" DevOps approach on both axis: Dev and Ops.

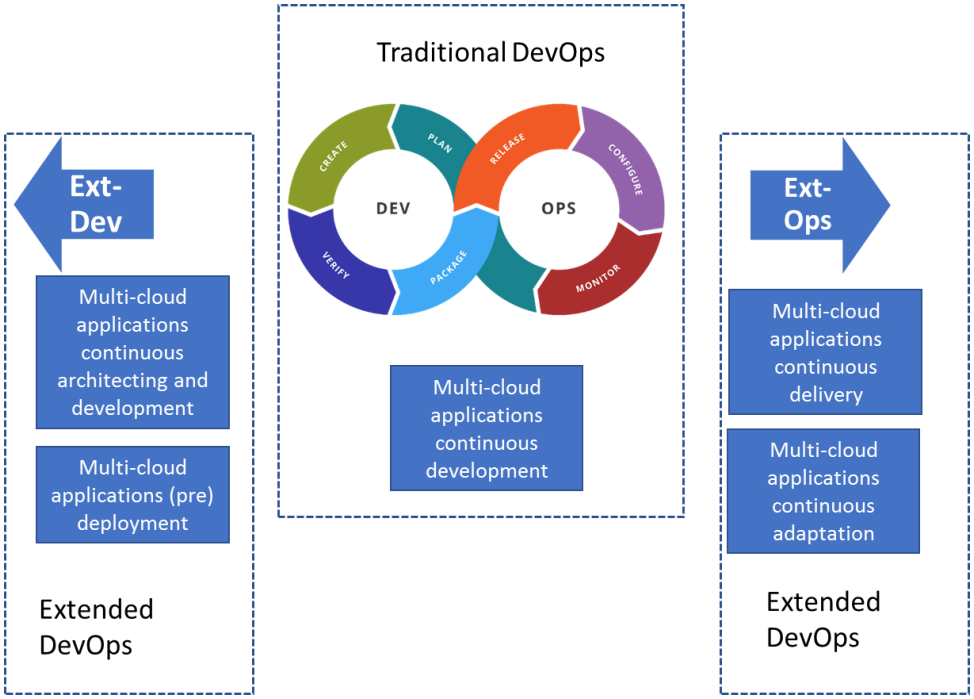


Figure 3. DECIDE extended DevOps approach

The extended DevOps phases and how DECIDE supports them are described in the next sections.

2.3 DECIDE Tools for multi-cloud applications architecting

2.3.1 NFR Editor

NFR editor is the component where the developers can state the Non-Functional Requirements (NFRs) they want to consider during the development and operation of the multi-cloud application. The NFRs can be qualitative or quantitative. In the context of DECIDE action the NFRs fall into the following categories, but they can be extended with new ones when needed (NFR Editor will provide means for this):

- Availability
- Cost
- Location
- Security
- Performance
- Scalability

The selected NFRs that are taken into account during the whole DECIDE process:

1. During the design phase, for proposing the most appropriate architectural design for complying with the selected NFRs

2. In the pre-deployment phase for performing the simulation with the objective of fulfilling the NFRs
3. In the continuous operation phase for monitoring and assessing that the selected NFRs are being fulfilled at run-time

2.3.2 ARCHITECT

The ARCHITECT tool consists of a catalogue of architectural patterns. These patterns serve for the optimization, development and deployment of applications to become multi-cloud aware. The idea is to present the developers with a set of architectural patterns to follow during the application design phase. These patterns are suggested to the developers based on the selected and prioritized NFRs as well as additional properties concerning the application (e.g. the number of micro services and whether they are stateless). The ARCHITECT tool is to be used by the developers at their discretion in the design phase. The ARCHITECT tool is also closely related to the development phase. The suggested patterns provide a description of how these patterns can be applied during the implementation of the code.

2.4 DECIDE Tools for multi-cloud applications continuous development and integration

2.4.1 DevOps framework

The DevOps framework is the integration point for all DECIDE tools and Key Results. It provides five main functionalities:

1. It serves as entry point to DECIDE. A user wishing to utilize the tools will do so through the DevOps framework.
2. It creates the needed resources and information for a DECIDE project, so that the process can be started (folders, git repository location, initial application description file, etc.)
3. It integrates the different tools and KRs. It provides access to them, a UI to check information about the projects (microservices data, metrics, SLAs violations, etc.) and centralizes the UIs of all the tools.
4. It orchestrates the workflow. The DevOps framework will launch the appropriate tool for each phase of the application's lifecycle.
5. It provides generic sensitive data information management functionalities for all the DECIDE tools.

2.5 DECIDE Tools for multi-cloud applications (pre) deployment

2.5.1 OPTIMUS

OPTIMUS deployment simulation tool is responsible for evaluating and optimizing the non-functional characteristics from the developer's perspective, considering a set of provided cloud resources alternatives. OPTIMUS, working with the continuous delivery supporting tool (ACSmI), will provide the best possible deployment application topologies, based on the non-functional requirements set by the developer, automating the provisioning and selection of deployment scripts for multi-cloud applications.

2.5.2 App Controller

The functionality of the Application Controller, as understood by the DECIDE consortium, should reflect the status and state of the application which refers to and connect it with the DECIDE tools, enabling each tool to understand its corresponding fulfilments. The Application Controller manages the application information that is relevant for all the different DECIDE tools.

This functionality has been conceptualised, and solutions for various components and parts of the Framework have been introduced. These are:

- *Application Description* (JSON File) (see annex 1) – is an information model specific to the DECIDE Application and is stored in a repository (e.g. Git) to be accessed by each tool. Each tool shares needed information by pushing and pulling from a dedicated repository. With this solution, an interoperable mechanism has been introduced. Furthermore, no running service is required, which limits a single point of failure and allows the tools to work individually.
- *Application Controller* (see section 4.3.2) is a reusable component that holds the logic for the Application Description, i.e. the model. It also allows reading and writing from the code repository.
- Furthermore, the *Application Controller* component assists in *managing the knowledge regarding the currently used deployment configuration and historical ones*. It keeps records of whether a deployment configuration was successful and if any SLA violations had occurred in the application operation time. With this information, OPTIMUS is able to suggest new and adequate deployment configurations. This is described in more detail in Section 4.3.1.

2.6 DECIDE tools for multi-cloud applications continuous delivery

2.6.1 ACSmI

The Advanced Cloud Service (meta-) Intermediator (ACSmI) provides means to assess continuous real-time verification of the cloud services non-functional properties fulfilment and legislation compliance enforcement. ACSmI is solution-centric, as it can discover services from a range available in a service registry, always making sure that the best combination for the user (i.e. OPTIMUS and ADAPT) is met, while ensuring the integrity and security of the overall ACSmI solution. ACSmI is also able to ensure the governance and overall quality of the service provision to the customer by continuously monitoring the fulfilment of the SLAs, as well as propagating the legislation changes.

2.7 DECIDE tools for multi-cloud applications continuous adaptation

2.7.1 ADAPT deployment and monitoring

The DECIDE ADAPT tool offers the following functionalities: deployment of multi-cloud applications, monitoring of the deployed applications to verify if the declared MCSLA is satisfied or not, and deployment adaptation to cope with identified violations.

ADAPT uses information from the Application Description to generate and apply the scripts for the deployment of the multi-cloud application. ADAPT uses ACSmI as a unified interface to create, monitor and release CSP resources. ADAPT continuously monitors the MCSLA and, in case of a violation, it informs the operator and triggers the redeployment process. Redeployment can be *automatic*, for a low technology risk application, or *subject to operator's confirmation*, for a high technology risk one.

2.7.2 MCSLA Editor

The MCSLA Editor module is part of the continuous operation phase and serves as the user interface (UI) through which the developer can specify the multi-cloud SLAs agreed with the client. The MCSLA Editor provides the developer all the possible SLOs and SQOs, which may partly incorporate default, aggregated or overwritten values, depending on the values resulting from the contracted CSPs. This resulting MCSLA serves as the contract between the developer and the users of the application. Additionally, the MSCLA will be used for monitoring purposes.

3 Alternative workflows

As explained in section 2.2, DECIDE covers the extended DevOps philosophy for multi-cloud applications. This requires supporting a complete workflow from the design of the multi-cloud application to its deployment and continuous adaptation. This complete workflow was introduced in D2.1 section 7 [1] (see D2.1 for a detailed description).

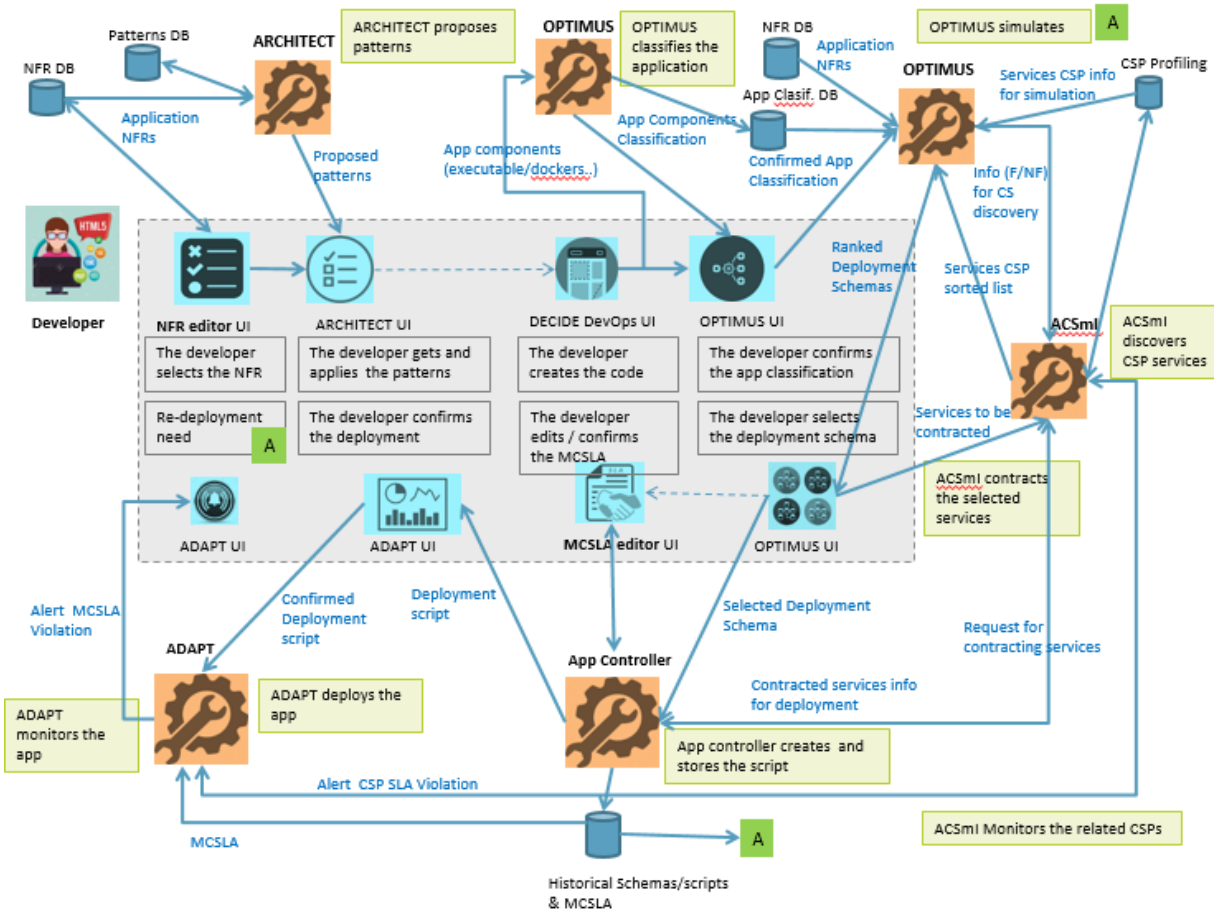


Figure 4. DECIDE workflow **[1]**

This section introduces on one hand the details of the redeployment workflow and on the other hand it presents alternative workflows that can be supported by DECIDE tools. These are sub-workflows that can be supported without executing the entire DECIDE workflow and its corresponding steps. These sub-workflows imply starting the workflow from intermediate points. At this stage three further different entry points have been identified: OPTIMUS, ACSmI and ADAPT.

3.1 Re-deployment workflow

In deliverable D2.1 the whole DECIDE workflow was described. In this section, the details of the re-deployment workflow are analysed so that the tools involved are aware of the required functionalities needed so that the re-deployment workflow can take place.

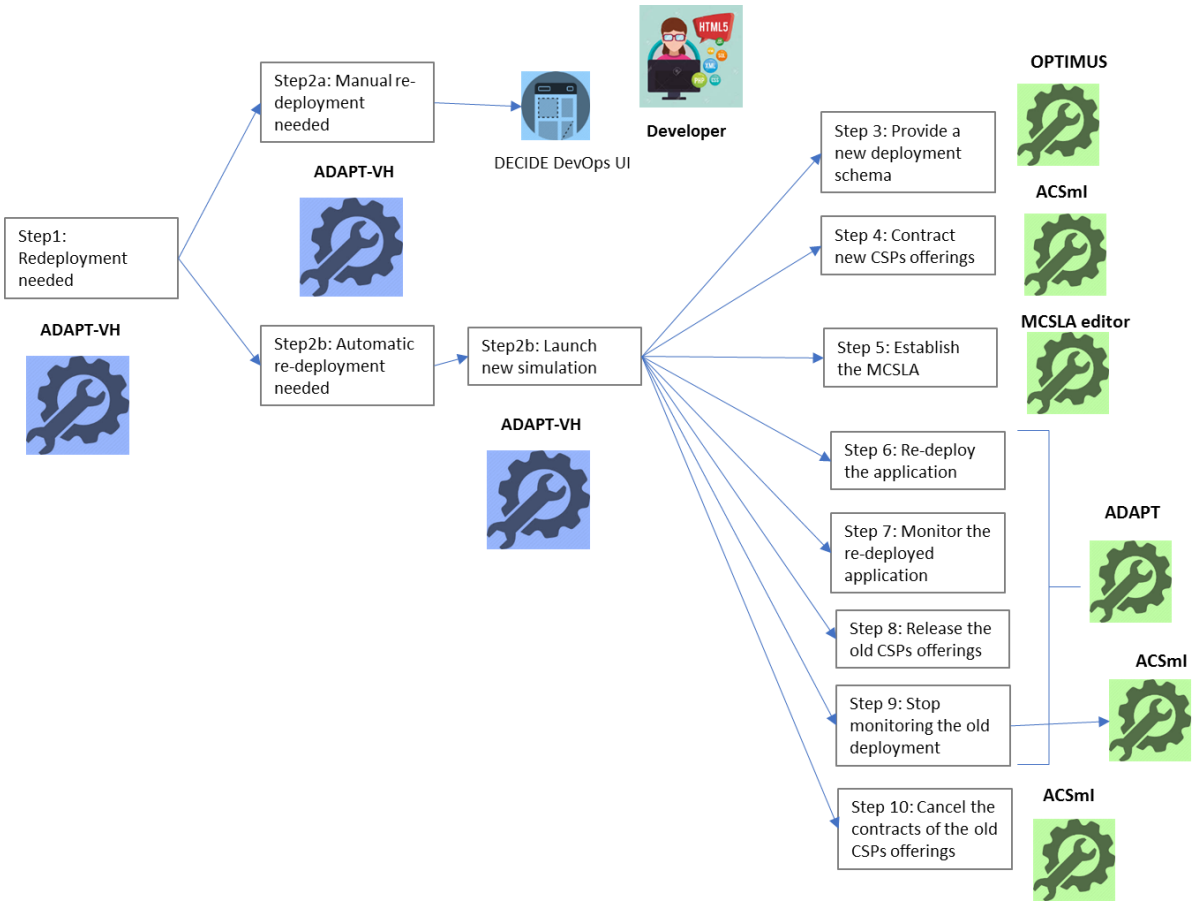


Figure 5. DECIDE re-deployment workflow.

Step 1	A violation that requires a (automatic or not) re-deployment is detected
Involved tools	VH (ADAPT)
Required functionalities	VH needs to discriminate if the occurred violation needs a re-deployment or not. If a re-deployment is needed, VH needs to know if this redeployment should be automatic or handled by the developer.

Step 2a	The re-deployment process should be handled by the developer
Involved tools	VH (ADAPT)
Required functionalities	VH needs to inform the developer requesting for a "manual" redeployment. Then the workflow will be managed by the developer through the DevOps framework.

Step 2b	The re-deployment process should be automatic
Involved tools	VH (ADAPT)
Required functionalities	VH needs to request OPTIMUS for a new simulation. VH needs to monitor the OPTIMUS activities to know when the OPTIMUS activities have been finished.

Step 3	The new deployment schema needs to be written down in the App Description.
Involved tools	OPTIMUS
Required functionalities	OPTIMUS needs to write the deployment schema without a confirmation from the developer. OPTIMUS when then return the status of “simulation finished” to the VH.

Step 4	The new CSP offerings need to be contracted.
Involved tools	ACSmI contracting/VH (ADAPT)
Required functionalities	VH needs to give the flow to ACSmI contracting. ACSmI contracting needs to contract the new CSPs offerings. This process can be automatic or may require information form the developer for the contracting.. Once finished the contracting process the VH should be notified.

Step 5	The MCSLA needs to be established
Involved tools	MCSLA editor/VH (ADAPT)
Required functionalities	VH needs to give the flow to MCSLA editor, and the MCSLA needs to be established. If the old MCSLA can be maintained, the process can be automatic (the MCSLA editor automatically checks if the old one is still valid). If a new MCSLA needs to be defined due to the constraints of the new deployment schema, the developer needs to validate this new MCSLA. Once finished, the VH should be informed.

Step 6	The multi-cloud application needs to be deployed into the new contracted CSPs.
Involved tools	ADAPT DO/VH (ADAPT)
Required functionalities	VH needs to give the flow to ADAPT DO so it can deploy the components in the new CSPs offerings.

Step 7	The multi-cloud application needs to be monitored
Involved tools	ADAPT (DO and monitoring)
Required functionalities	ADAPT DO requests ADAPT mon to start the monitoring.

Step 8	The old CSPs offerings need to be released but only once the application and the data have been successfully ported.
Involved tools	ADAPT (DO)
Required functionalities	ADAPT DO needs to release the old CSPs offerings.

Step 9	The old deployment does not need to be monitored anymore.
Involved tools	ADAPT (DO and monitoring) /ACSmI (monitoring)

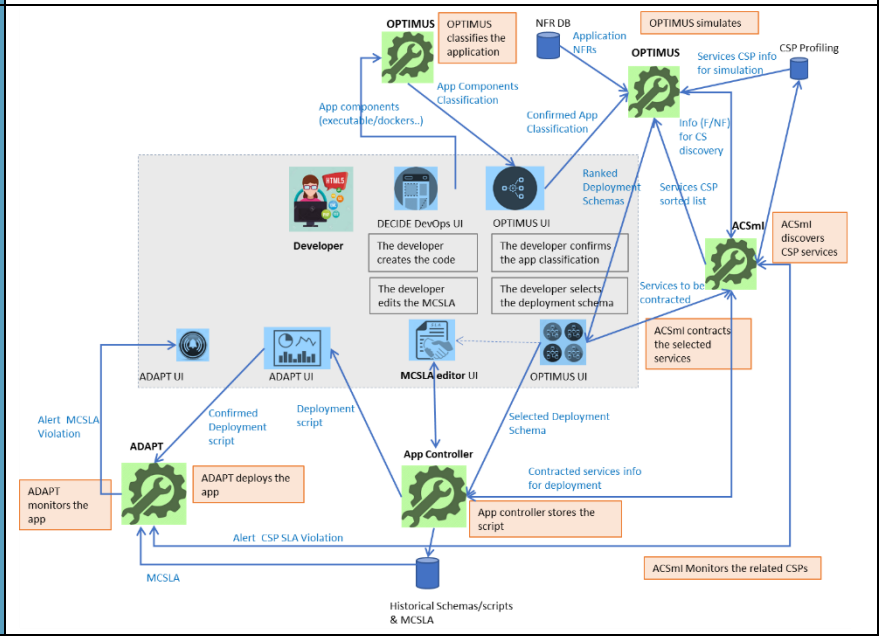
Step 9	The old deployment does not need to be monitored anymore.
Required functionalities	ADAPT DO needs to request to ADAPT monitoring to stop monitoring the old deployment. ADAPT monitoring will request ACSmI monitoring to stop monitoring the old CSPs offerings. Once finished the ADAPT actions, the VH should be informed. Once received the confirmation the VH will inform the developer that the redeployment has been successfully finished.

Step 10	The contracts from the old CSPs need to be cancelled.
Involved tools	ACSmI (billing)/ADAPT (VH)
Required functionalities	VH requests ACSmI to cancel the contracts form the old CSPs offerings. ACSmI needs to cancel the old contracts. The VH needs to be informed that ACSmI has cancelled the contracts.

3.2 Starting the workflow from OPTIMUS

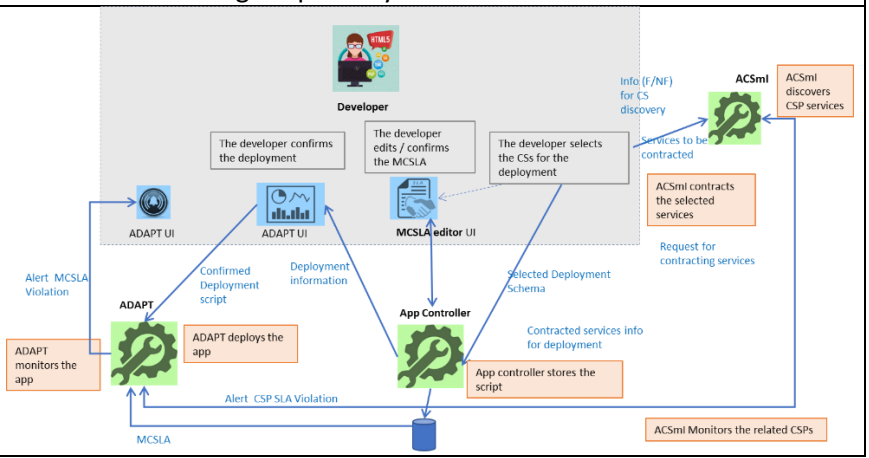
Table 1. Alternative workflow 1: OPTIMUS + ACSmI + ADAPT + MCSLA Editor

Tools involved	OPTIMUS + ACSmI + ADAPT + MCSLA Editor
Initial requirements	The general editor plugin (included in OPTIMUS) needs to create initial data for the DECIDE project, as the DevOps framework does in the complete DECIDE workflow.
Sub-workflow description	This sub-workflow starts in the pre-deployment phase. OPTIMUS recommends the best deployment schema based on the information introduced by the developer and the information contained in ACSmI. Once the developer confirms the deployment schema, the MCSLA is created and established and the different cloud services are contracted through ACSmI. Then the multi-cloud application is deployed by ADAPT and is continuously monitored to assess the working conditions of the application, based on the established MCSLA. ACSmI also monitors the CSP resources contracted.
Issues to be considered	• Credentials management in an Eclipse Plugin. • A DECIDE project needs to be created through the Eclipse plugin. • Link to the git repository.

Tools involved	OPTIMUS + ACSmI + ADAPT + MCSLA Editor
Graphical representation of the sub-workflow	

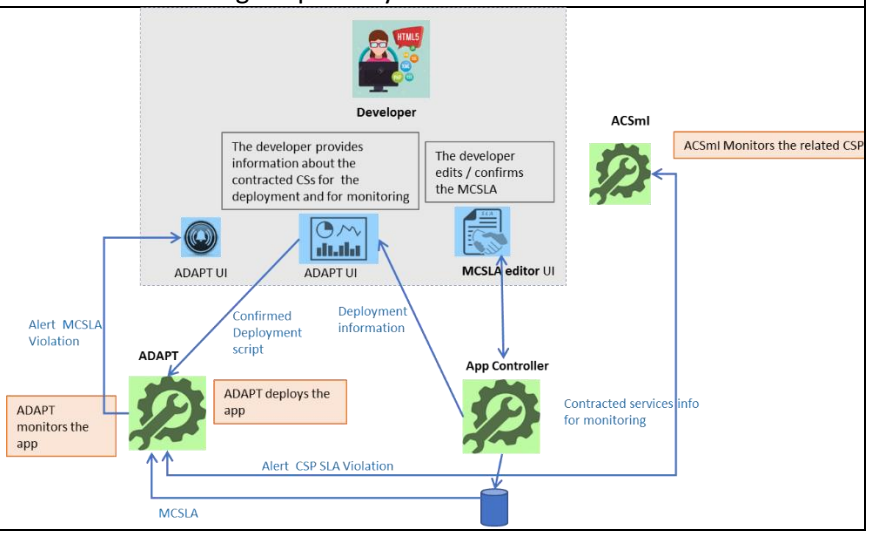
3.3 Starting the workflow from ACSmI

Table 2. Alternative workflow 2: ACSmI + ADAPT + MCSLA Editor

Tools involved	ACSmI + ADAPT + MCSLA Editor
Initial requirements	The initial data for the DECIDE project need to be created (through the DevOps framework). NFRs need to be gathered, introduced by the developer (directly in the application description). ACSmI should provide a UI for the developer to discover cloud services.
Sub-workflow description	The developer establishes the required MCSLA through the MCSLA editor and discovers the required cloud services through the ACSmI. The contracting of the selected services is performed through ACSmI, and ADAPT deploys the application on the contracted services and monitors it based on the established MCSLA. ACSmI also monitors the CSP resources contracted.
Issues to be considered	• A DECIDE project needs to be previously created through the DevOps framework. • Link to the git repository.
Graphical representation of the sub-workflow	

3.4 Starting the workflow from ADAPT

Table 3. Alternative workflow 3: ADAPT + MCSLA Editor + ACSmI monitoring

Tools involved	ADAPT + MCSLA Editor ACSmI monitoring
Initial requirements	There has to be a project so the Project Wizard shall be launched from the DevOps framework. NFRs must be shown to the user to be selected (they will have to be monitored by ADAPT Monitoring), in the DevOps Framework. The user must insert the characteristics and details of the cloud resources where he wants each microservice to be deployed, and the app description should be generated. This step will be manual, so the user shall be able to select the resources where to deploy the components in a human/readably way in the Application Description file. The user must also insert the values of the SLAs of the selected CSPs, and the MCSLA editor should create the composite SLA (step not always necessary) The developer will have to contract the services himself and provide the credentials to ADAPT.
Sub-workflow description	The developer needs to create the DECIDE project through the DevOps framework and select the NFRs. Then he/she needs to introduce the data of the contracted cloud services in ADAPT configuration. Once the information is completed and the MCSLA is composed through the MCSLA Editor, ADAPT will automatically deploy the components and start monitoring them. ADAPT monitoring will also need the metrics from the resources monitored by ACSmI Monitoring.
Issues to be considered	• A DECIDE project needs to be previously created through the DevOps framework. • Link to the git repository.
Graphical representation of the sub-workflow	

4 Detailed DECIDE integrated architecture

4.1 DECIDE tools for multi-cloud applications design and development

4.1.1 NFR editor

The NFR editor is the component that allows the developers of a multi-cloud application to define the non-functional requirements that they want to consider during the development and the operation of the application.

4.1.1.1 Structural description

The NFR editor supports the developer during the "continuous architecting" phase. NFR editor allows to define the relevant NFRs for the multi-cloud application. Based on these defined NFRs ARCHITECT can propose specific architectural pattern. The NFR list contains, for the context of DECIDE, the following categories: availability, cost, location, security, performance and scalability, but it can be extended with new ones when needed.

The NFR editor also supports the developer in the "pre-deployment" phase for detailing the previously selected NFRs. In this step, the NFR editor provides the means for the developer to select the exact values for the NFRs that can be quantified (e.g. cost, location) and apply these values to the components of the multi-cloud application.

The main functionalities of the NFR editor are:

- Provide the available qualitative NFRs, and the means to select them at application level.
- Provide the available quantitative NFRs, and the means to detail their values at component / microservice level.
- Provide the means to store the selected NFRs (qualitative and quantitative).

In the next figure, the sub-components of the NFR editor are presented:

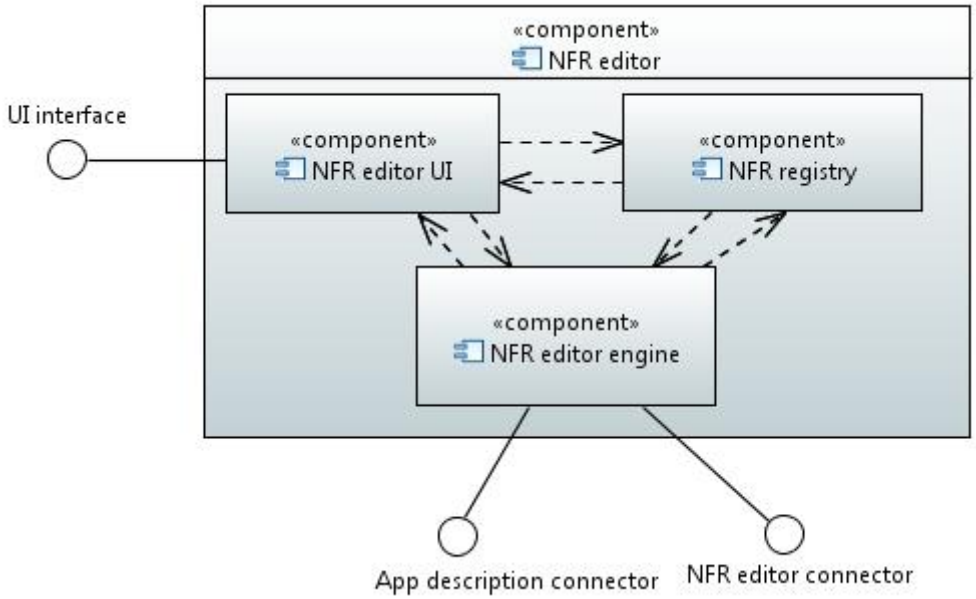


Figure 6. NFR editor component diagram

The NFR editor is composed of the following sub-components:

NFR editor UI

This subcomponent provides the graphical user interface of the NFR editor, used both for selecting the qualitative NFRs at the continuous architecting phase and for detailing the quantitative values for each component. The NFR editor UI is loaded together with the ARCHITECT or OPTIMUS user interfaces.

NFR registry

The NFR registry stores the available NFRs to be loaded by the NFR editor UI. This registry is static, and it contains the available NFRs and their accepted values.

NFR editor engine

The NFR editor engine is the component that manages the different activities to be carried out by the NFR editor. This sub-component gets the requests from ARCHITECT and OPTIMUS and triggers the different sub-components inside the NFR editor. It also stores the values of the selected NFRs in the App Description.

The external interfaces of the NFR editor are:

1. Interface with OPTIMUS and ARCHITECT that accepts requests from those two DECIDE components to select the NFRs
2. Interface with App Description that stores the selected values for the NFRs
3. Interface with DevOps Framework that provides the UI in the integrated DECIDE framework

In the following diagram, the external interfaces of the NFR editor are shown:

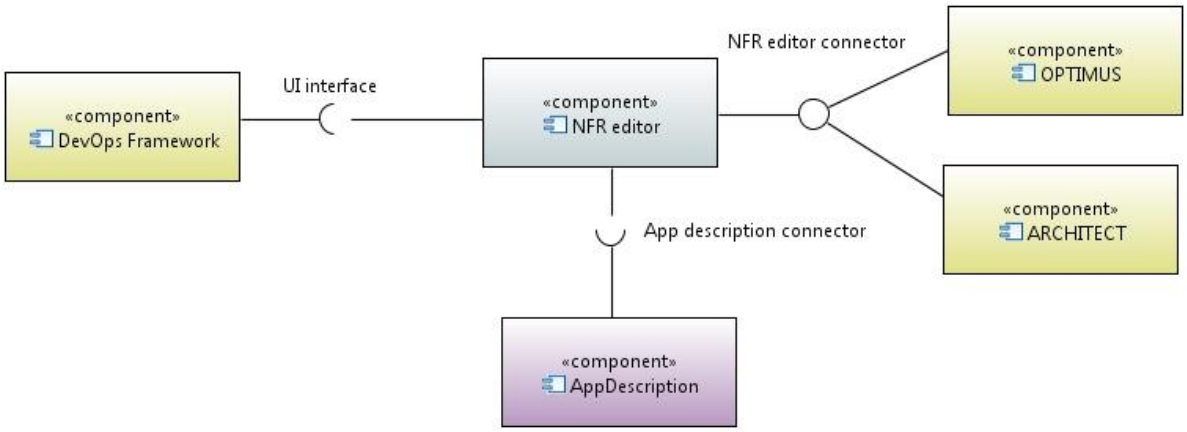


Figure 7. NFR editor external interfaces component diagram

4.1.1.2 Behavioural description

The interaction between the NFR editor and other components in DECIDE as well as the messages they share can be found in the following sequence diagram:

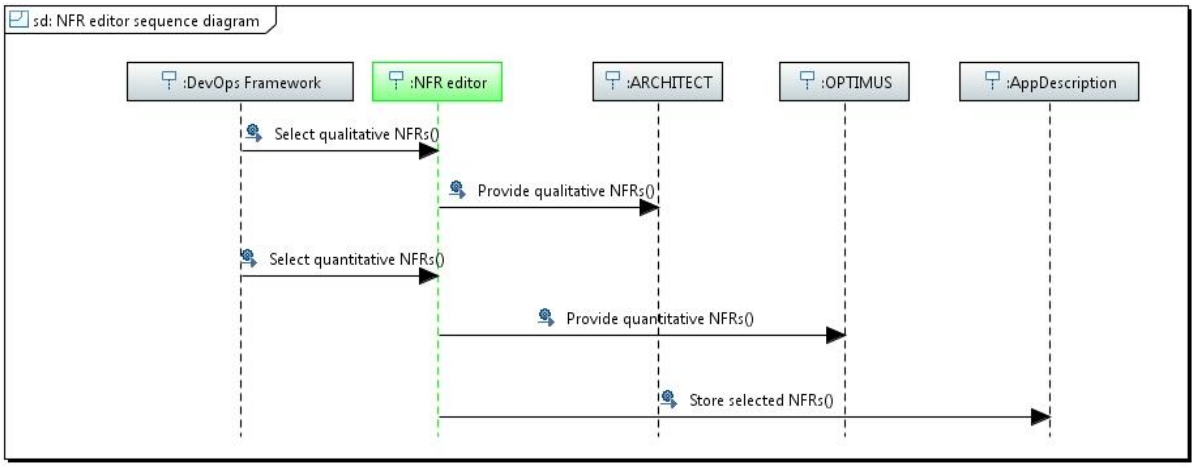


Figure 8. NFR editor sequence diagram.

1. Select qualitative NFRs: The developer through the DevOps Framework UI (or the UI of the NFR editor) selects the corresponding NFRs from the list provided by the NFR editor.
2. Provide qualitative NFRs: ARCHITECT gets the selected NFRs from the NFR editor.
3. Select quantitative NFRs: The developer through the DevOps Framework UI (or the UI of the NFR editor in the Eclipse plugin version) selects the value for each NFR from the list provided by the NFR editor, for each of the micro-services of the multi-cloud application.
4. Provide quantitative NFRs: OPTIMUS gets the selected values for the NFRs from the NFR editor.
5. Store selected NFRs: The NFR editor stores the selected NFRs and its values in the Application Description, so that they can be assessed during the operation phase of the multi-cloud application.

4.1.2 ARCHITECT

ARCHITECT supports the developer with the design of a multi-cloud application and the preparation of the deployment scenarios by providing and suggesting a set of (multi-)cloud design patterns, which should be considered during the application implementation.

4.1.2.1 Structural description

By means of the functional requirements, ARCHITECT is decomposed in several functional blocks and interfaces. The ARCHITECT component has a set of functional requirements [1] that can be summed up in the following functionalities:

- Provide/ recommend to the user (i.e. the developer) architectural patterns based on his/her prioritized NFRs as well as additional information (supplied by the user), with guidelines on how to apply them, to which component this needs to be applied and in which order. This should be performed through a UI.
- Provide a repository of relevant multi-cloud patterns.

Apart from these functionalities, ARCHITECT helps to initiate the development of an application in the context of DECIDE. This includes the creation of the DECIDE project artefacts, mainly consisting of the application description contained in the code repository.

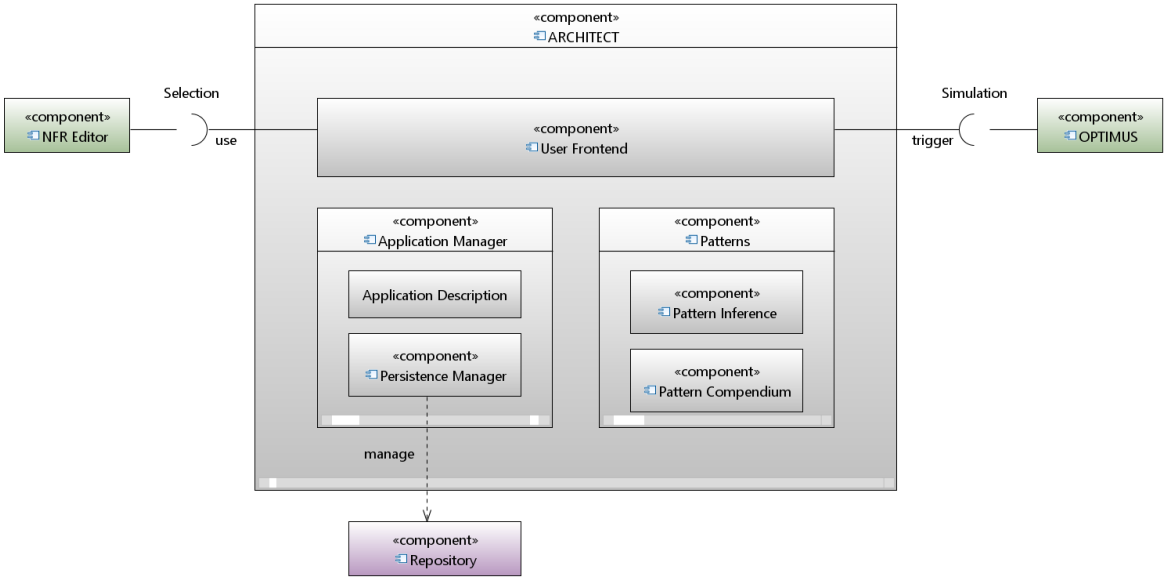


Figure 9 Components of ARCHITECT

ARCHITECT consists of three core elements, as depicted in the figure above. A frontend for user interaction, the application description manager for dealing with the DECIDE project model, and finally the patterns catalogue with the pattern inference engine.

User Frontend

The User Frontend is the workflow-controlling component of ARCHITECT. In the current implementation of DECIDE tool chain, ARCHITECT is integrated as an Eclipse IDE and the User Frontend component provides the mechanism how the ARCHITECT component is plugged in. Its main task is the interaction with the developer by providing the necessary user interface elements to collect and maintain all the application information and to enable the use cases that are described in the next section. An alternative UI is provided in the DevOps framework as part of its Dashboard with the same functionality.

Application Manager

This component is responsible for a convenient abstraction layer for the information model of the DECIDE application. It manages all application information in a transparent manner. That means, it encapsulates and hides the technical details (the application description is encoded and stored as a JSON structure inside a code repository).

Patterns

This element contains a catalogue of patterns, NFRs and their relationships. The contained information can be updated with additional information and patterns over time. The patterns catalogue provides functions that allow the inferring of patterns based on a given set of NFRs and, optionally, the suggestion of some fundamental patterns.

ARCHITECT itself does not provide any external interfaces. The exchange of information between ARCHITECT and OPTIMUS is taking place via the App Description file. However, the *Patterns* component is implemented as an autonomous library and its functionality is offered as a micro-service (Patterns Compendium) that can be accessed by other implementations. This allows an easy integration of ARCHITECT in a polyglot environment. ARCHITECT consumes two interfaces, one from the NFR Editor and the other from the OPTIMUS component.

NFR Editor

ARCHITECT utilizes the NFR Editor for collecting the set of defined non-functional requirements from the application developer. ARCHITECT expects as return value from the editor the list of NFRs that the developer has selected.

OPTIMUS

For a manual triggering of the simulation phase, ARCHITECT should be able to call OPTIMUS. The main artefact transferred is the *Application Description*. Depending on the provided interface of OPTIMUS it can either be referenced through the code repository or be handed over as a parameter in the API method. The result will be returned using the same mechanism. The *User Frontend* and the *Application Manager* may display the result in the current environment in an appropriate way.

4.1.2.2 Behavioural description

Based on the list of the functional requirements [1], several use cases for the developer have been identified as shown in the following figure. These are mainly the creation of a new project, a change of NFRs and a change of selected patterns of an already existing DECIDE project; Finally, the developer or the used CI tool should be able to enter the next DECIDE phase by triggering OPTIMUS for a deployment simulation.

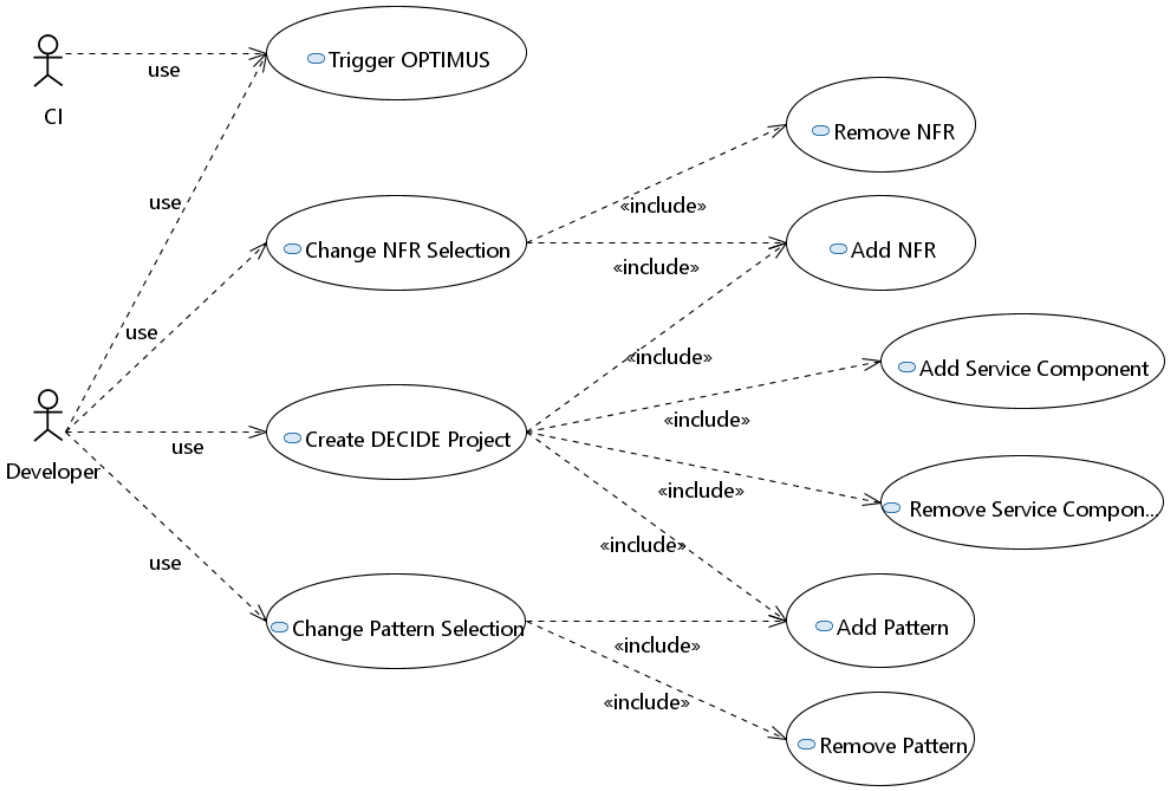


Figure 10. Use Cases of ARCHITECT

The following sequence diagram shows the “Create DECIDE Project” process.

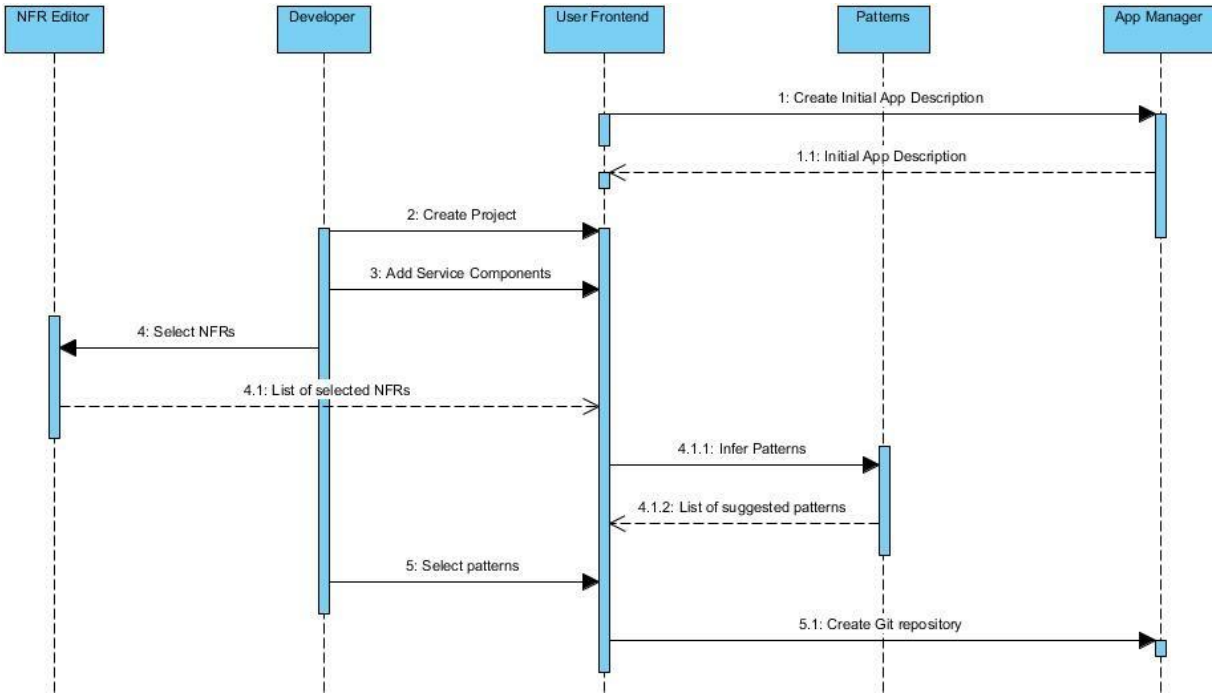


Figure 11 Create new DECIDE Project, sequence diagram.

1. The developer starts the creation of a new DECIDE project.
2. The *User Frontend* requests an initial *Application Description* from the *Application Manager*.
3. The *User Frontend* shows to the user a form that requires the general information about the application, e.g. which micro-services it contains and how they are related to each other and to the application in general.
4. The *User Frontend* shows to the user the NFR Editor, where she can select a set of prioritized NFRs. The NFR Editor returns to *User Frontend* with the selected list of NFRs.
5. Based on the selected NFRs and the additional application information, a list of patterns is suggested to the developer. This list contains both fundamental and inferred patterns.
6. The developer is asked to select any patterns from the catalogue that should or must be applied to the application design.
7. After the developer has selected the patterns for the application, the *User Frontend* finishes the creation process by persisting the final *Application Description* using the *Application Manager*.

4.2 DECIDE tools for multi-cloud applications continuous integration and testing

4.2.1 DevOps framework

4.2.1.1 Structural description

The DevOps framework is the entry point to DECIDE. It allows a user to define a new project and modify its metadata[3], and centralizes de UIs of the different tools and KRs. Besides, the DevOps framework will act as the orchestrator of the DevOps workflow, launching the corresponding tool whenever appropriate.

The following figure shows the components of the framework:

[3] Project metadata captured form the user: i.e number of microservices, git where the project is located, users, etc.

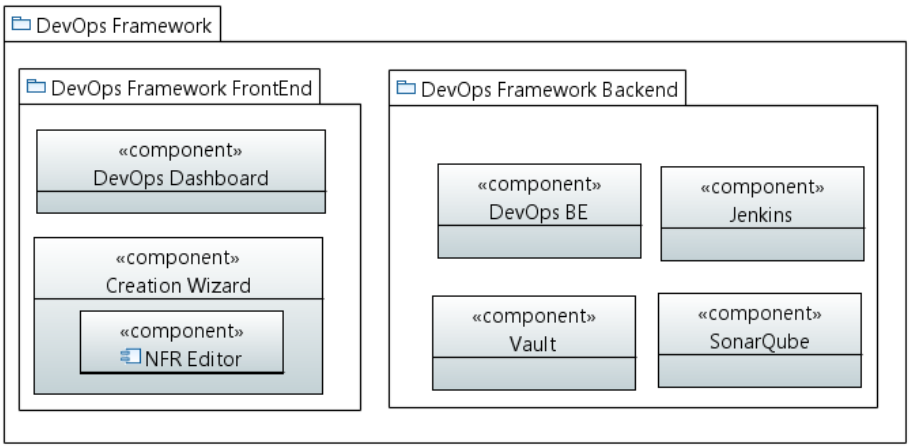


Figure 12. DevOps framework component diagram

Creation Wizard

The Creation Wizard allows a user to introduce basic data about the application. These data include the Git's token and URL where the code is or will be located, the number of microservices, programming language, etc. The Wizard includes an *NFR editor* to indicate the non-functional requirements that the project must fulfil along its life-cycle, and that will be considered by the rest of the tools.

DevOps Dashboard

The Dashboard gives an overview of the status of the DECIDE tools. On one hand, it provides information about the CI/CD tools included in the DevOps Framework: Jenkins and SonarQube. On the other hand, it displays the most relevant data from the different DECIDE tools, such as the selected patterns for ARCHITECT, the result of the simulation for OPTIMUS, and deployment and monitoring information from ADAPT.

DevOps BE

This component is where the intelligence of the DevOps Framework resides. The backend is in charge of administrative tasks, such as user and application management, but is also responsible for triggering and coordinating the different DECIDE tools. Besides, the DevOps BE takes care of reading/writing the Application Description and creating new projects.

Jenkins, SonarQube, Vault

These components correspond to third-party tools that are integrated in the DevOps Framework. Jenkins and SonarQube are CI/CD tools, for automatically triggering builds and for code testing. Vault is a component for securely sharing sensitive data within DECIDE.

The DevOps Framework communicates with the DECIDE tools at two different levels: at a GUI level, it integrates the graphical interfaces of the tools, which, for some tools, is done through iframes, while, for others, the Dashboard builds their GUIs. At information level, the Dashboard shares data with the different tools following two strategies: the Application Description and direct API invocation.

The following figure shows the aforementioned communications:

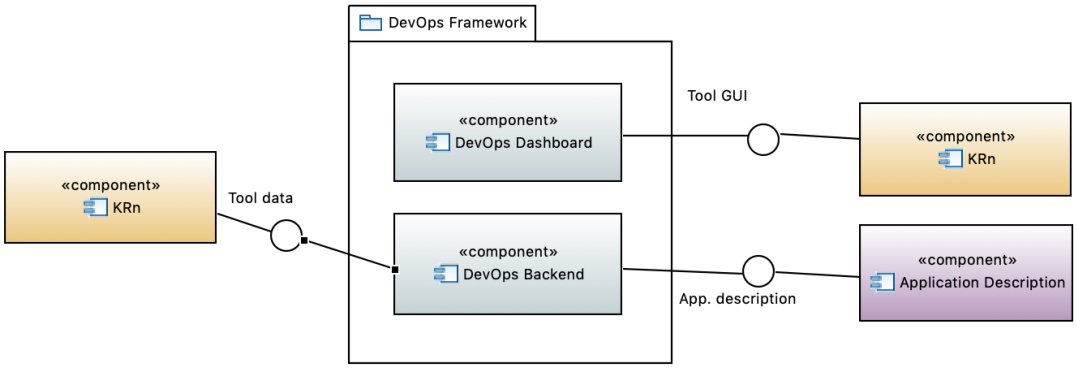


Figure 13. DevOps Framework interfaces diagram

4.2.1.2 Behavioural description

The behaviour of DevOps Framework tool can be described as follows:

1. The DevOps Framework provides a graphical interface for a user to introduce information about the application and its microservices. When that information is complete, it updates the Application Description with it. It also configures the tools for continuous development, integration and testing according to the provided application information.
2. ARCHITECT, based on the Application Description, generates a list of recommended patterns, which then sends back to the DevOps Framework to be displayed.
3. While the development, integration and testing processes are taking place, the DevOps Framework receives the information reported by the corresponding tool and displays them in the UI.
4. It also sends OPTIMUS an order to start a new simulation, either after a direct instruction from the user or as a result of a violation during ADAPT's monitoring process.

The following figure shows these communications:

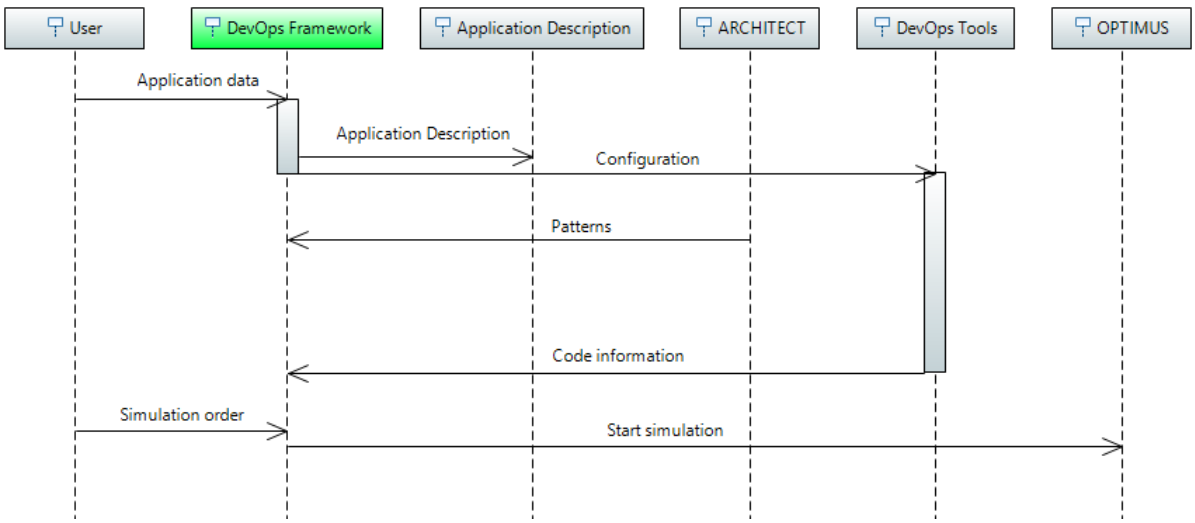


Figure 14. DevOps Framework sequence diagram

4.3 DECIDE tools for multi-cloud applications (pre) deployment

4.3.1 OPTIMUS

4.3.1.1 Structural description

The main functionalities in OPTIMUS are:

- Multi-cloud application classification**.** This functionality will include the classification of the components that form the multi-cloud application (computing, computing IP, storage persistency, storage DB) [9]. For this purpose, the profiling of the multi-cloud application has to be considered as an input. This classification will be based on the information provided by the developer and the information stored in the general applications profiling repository.
- Theoretical deployment generation**.** Once the classification is made, and the NFRs gathered, it will perform a process where it obtains a theoretical schema for the deployment. This schema will be composed of generic types of CSPs, associated to the types set to the micro services. With these generic types of CSPs suitable for the components, a request will be made to the corresponding service of ACSmI. This functionality requires the "CSP modelling" functionality to be available.
- Simulation. The combination of the different possibilities of deployment, considering the theoretical deployment and the sorted list of CSPs (from ACSmI) that suit them, will be ranked in order to select the best ones. The five best schemas will be built and shown to the developer to confirm and select the one desired.

The figure below shows the components and sub-components that will support the listed functionalities:

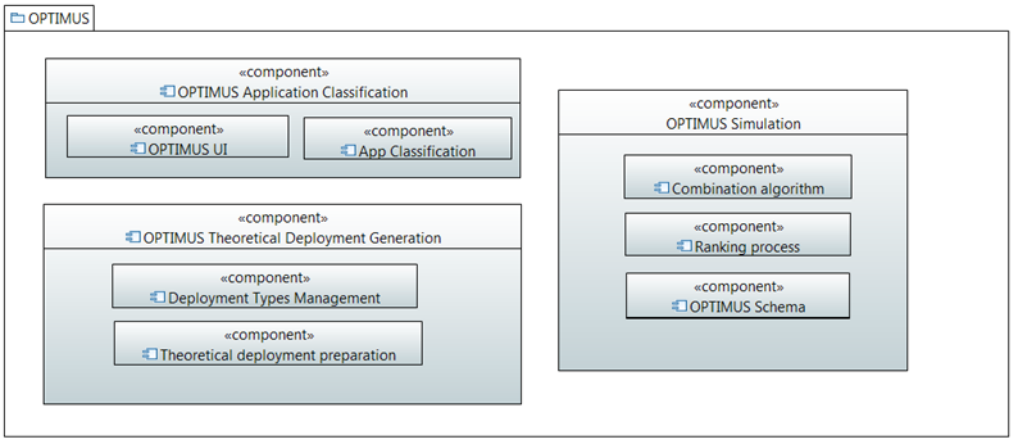


Figure 15. OPTIMUS component diagram

Application classification

The input for this classification task will come from the developer (the UI will show information to him to complete and confirm), and it will be matched with the information stored about types of multi-cloud applications, and the characteristics associated to each of those types. The output will be stored into the *Application description* element.

Theoretical deployment generation

The *Deployment Types* repo will contain information about the micro-services types and the CSPs on which they could be theoretically deployed. The maintenance of this repo will be performed by the *Deployment types management* module.

Taking as input the multi-cloud application classification (micro-services) and the application's NFRs, the theoretical deployment generation component will access the *Deployment Types* repo to obtain the set of CSP offerings that can be used for its deployment. Once it has all the information, it will create a list of possible CSP offerings for each micro-service.

Simulation

The entry for the combination process will be the information about the different possibilities existing for the deployment. The algorithm will perform a combination of all these possibilities, using the different CSP offerings available for each micro-service. These combinations will be sorted from the best of them to the worse. An output with the five best deployments will be created.

The best option for the deployment could be selected and confirmed by the developer through the OPTIMUS UI, and stored into the Application description element as well as into the historical repo managed by the APP CONTROLLER.

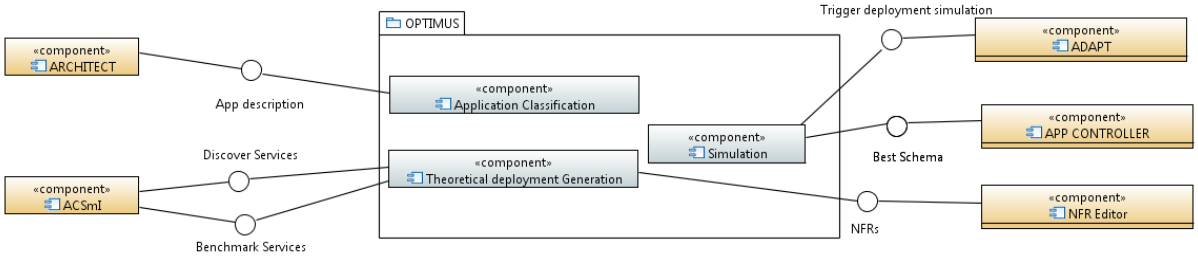


Figure 16. OPTIMUS external interfaces component diagram

4.3.1.2 Behavioural description

The behaviour of OPTIMUS tool, and the interchanged data among the different actors in this part of the DECIDE workflow, is shown by the picture below:

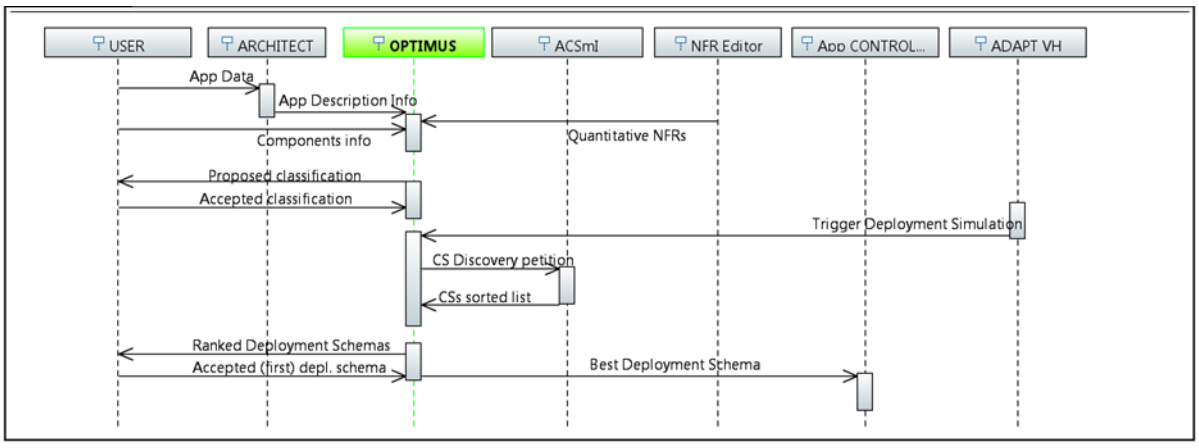


Figure 17. OPTIMUS Sequence diagram

The messages among OPTIMUS and the rest of tools are:

1. App Data: The first information about the application is provided by the user, and requested by the General Editor included in the *ARCHITECT* UI.
2. App Description Info: Once *ARCHITECT* has suggested the most suitable patterns, the related information is stored as part of the Application Description and *OPTIMUS* will have access to it.
3. Quantitative NFRs: The NFRs indicated by the user are an important information to obtain a proper classification and the best theoretical deployment for the application.
4. Components info: The developer or user has already developed the application or has a detailed design about it. *OPTIMUS* requires additional information about the micro-services that the application is composed of.
5. Proposed classification: *OPTIMUS* IU presents the user the results of the application classification.
6. Accepted Classification: The user, through *OPTIMUS* UI, accepts the classification made.
7. Trigger Deployment simulation: When ADAPT VH identifies a violation, a new redeployment simulation is triggered by requesting *OPTIMUS* and sending it the information about the violation
8. CS Discovery petition: *OPTIMUS* asks to *ACSmI* for a list of CSP offerings that fulfill the requirements that the user and the classification process have established.
9. CSP offerings sorted list: *ACSmI* sends *OPTIMUS* a list whose first element is the CSP that best fits the non-functional requirements.
10. Ranked Deployment Schemas: After *OPTIMUS* creates the schemas for the deployment, based on the information sent by *ACSmI*, it will present the five best of them to the user as a ranking (including the NFRs fulfillment for every schema) and he or she will select the schema to be used.
11. Accepted deployment Schema: The schema selected by the user, the best schema presented by *OPTIMUS*.
12. Best Deployment Schema: This output is the global result from *OPTIMUS*. It will be stored into the application description element and into the historical repo of selected schemas by the *App controller*.

4.3.2 Application controller

The Application Controller will assist in managing the intelligence regarding the current and historical deployment topology. The aim of this functionality is to mitigate reusing deployment topologies that were faulty or inadequate in the past.

More detailed information about the implementation of the Application Controller can be found in D3.10 [10].

4.3.2.1 Structural description

The following requirements have been elicited in the project for the Application Controller [1]; these are translated below in Figure 18 as functional components. The requirements can be summed up in the following functionalities:

- Holding the intelligence of the different deployment configurations that the multi-cloud application has had in its operation time. Storing these deployment configurations will allow avoiding those configurations that resulted problematic in terms of security, performance or legal awareness.
- Maintain an interface to OPTIMUS in order to receive the deployment configuration chosen to be stored.

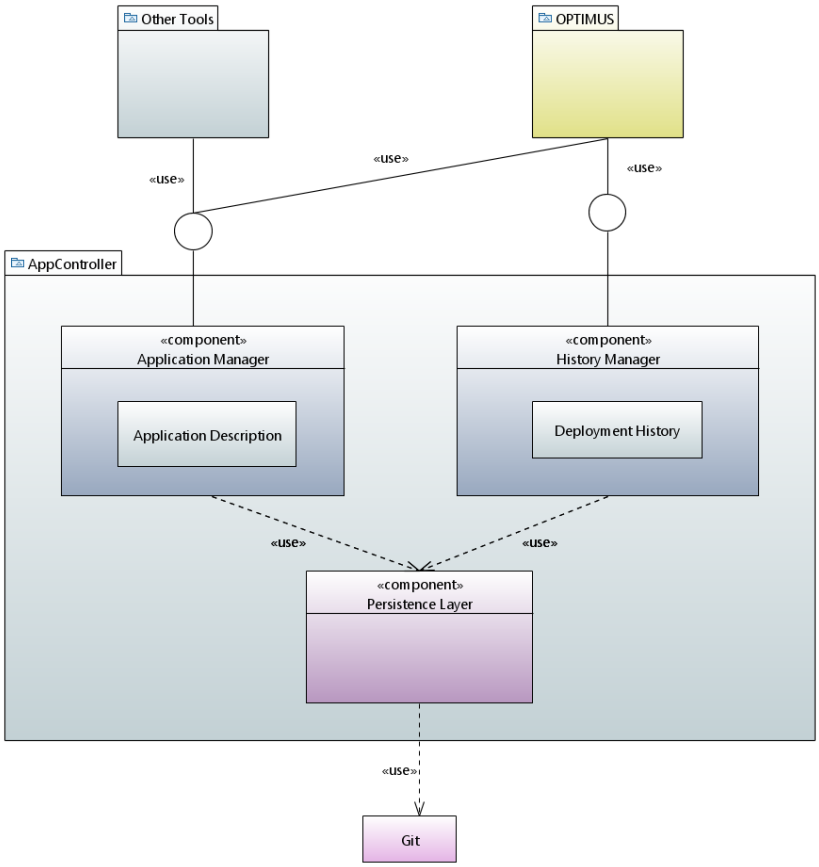


Figure 18. Component Diagram for Application Controller

Furthermore, it stores the deployment history in the code repository where the Application Description is also stored.

The deployment history will include meta-data regarding the deployment configuration, such as time and date of deployment, the current status, information on the microservice, CSP data and information regarding any SLA violations.

4.3.2.2 Behavioural description

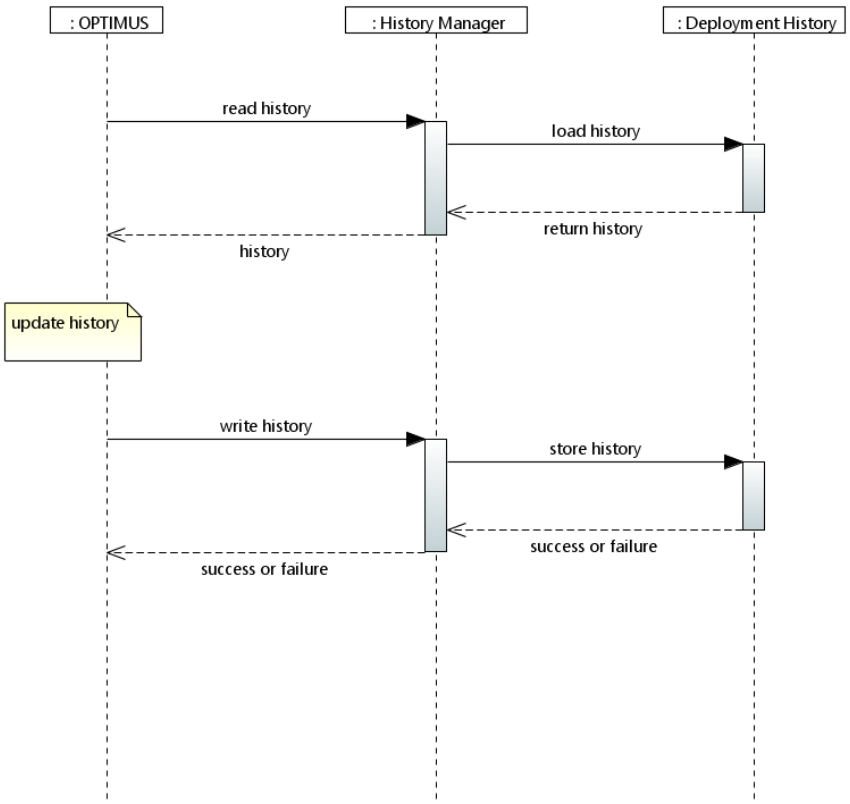


Figure 19. Sequence Diagram for writing and reading deployment configuration

The sequences performed for reading and writing to the deployment history are as follows:

1. Once a successful deployment has taken place, *OPTIMUS* registers the current deployment configuration with the *Application Controller* component. The information to be registered is described above and is specified in the meta-data model (see annex 1 for the meta-model information).
2. The *Application Controller* through the *History Manager* sub-component will update the existing file that holds the deployment history in the code repository. In the case that the file does not exist, the Application Controller component should create it.
3. If an SLA violation takes place, this is reported and the file is updated accordingly (same process as prior step).
4. In the case of a new simulation phase, *OPTIMUS* shall read from the deployment configuration history through the *History Manager* of the *Application Controller* component.
5. The *History Manager* supplies in turn *OPTIMUS* with the information needed in order to evaluate which deployment topology is adequate.

4.3.3 ACSmI

4.3.3.1 Structural description

The Advanced Cloud Service (meta-) intermediator (ACSmI) will provide a cloud services store where discovery, contracting, managing and monitoring different cloud services. ACSmI will provide the means to assess continuous real-time verification of the cloud services non-functional properties fulfilment and legislation compliance enforcement. ACSmI will also provide means for Cloud services endorsement from different CSPs.

These are the main functionalities envisioned:

- Endorse a cloud service into the ACSmI. ACSmI will allow registering services. This can be performed by the CSP itself, by the multi-cloud application operator or by the ACSmI Administrator. The registry of each service should cover the different defined terms to model the CSPs and their services. This will allow the discovery of the services from the service registry.
- Discover and benchmark services. ACSmI provides means to indicate which are the requirements (functional and non-functional) that the discovered services should fulfil. ACSmI will discover, from the services stored in its registry, the most appropriate ones for that set of requirements. Then, from the set of discovered services, ACSmI will prioritize these services in terms of the fulfilled requirements which will be passed to OPTIMUS as a short list. The list will include, additionally, the degree of fulfilment of the NFRs requested by the user.
- Legal compliance. ACSmI provides means to indicate a legal level defined through the assessment of legal relevant aspects when initiating a service, in particular in regard to location of data, data security level, location of the service provider etc. in order to enable the cloud user/developer to assess the legal impact of initializing and operating the proposed service.
- Contract services. This functionality will allow dealing with the activities related to the contracts within the ACSmI. Depending on the type of services and the CSP, ACSmI will manage the contracts in two different ways: 1) ACSmI will facilitate contracting services directly by the user to the provider and 2) ACSmI will manage the contract itself with the provider and the user. In this last case, ACSmI will have mainly two types of contracts. The first one is the contract with the CSP and the other is with the user of the services intermediated by the ACSmI.
- Manage CSPs. This functionality will allow the management of the different connectors to facilitate the contracting of the services and to monitor them. This functionality will be in charge of informing ADAPT with the required information for the deployment of the multi-cloud application through the different contracted services.
- Monitor NFR CSPs and manage the violation alerts. This functionality will monitor the SLA (NFRs) of the service offered by the CSPs to detect any violation of the SLAs. Several metrics will be measured and assessed to detect if a SLA violation occurs. If the CSP occurs ACSmI will carry out three main actions: 1) To inform ADAPT VH that the violation occurs and to provide it with all the required information, 2) to register in the ACSmI the information of the violation in the corresponding services and 3) to alert the CSP.
- Monitor the use and bill the user. This functionality will allow calculating the costs generated by the user for using ACSmI recommended cloud services, and will provide the corresponding invoice. To be able to generate the billing of the contracted services, ACSmI shall monitor the use of the different cloud services.

The high-level architecture of the ACSmI is presented next. The Figure 20 is an updated version of the ACSmI architecture presented in D5.1 “ACSmI requirements and technical design” [17].

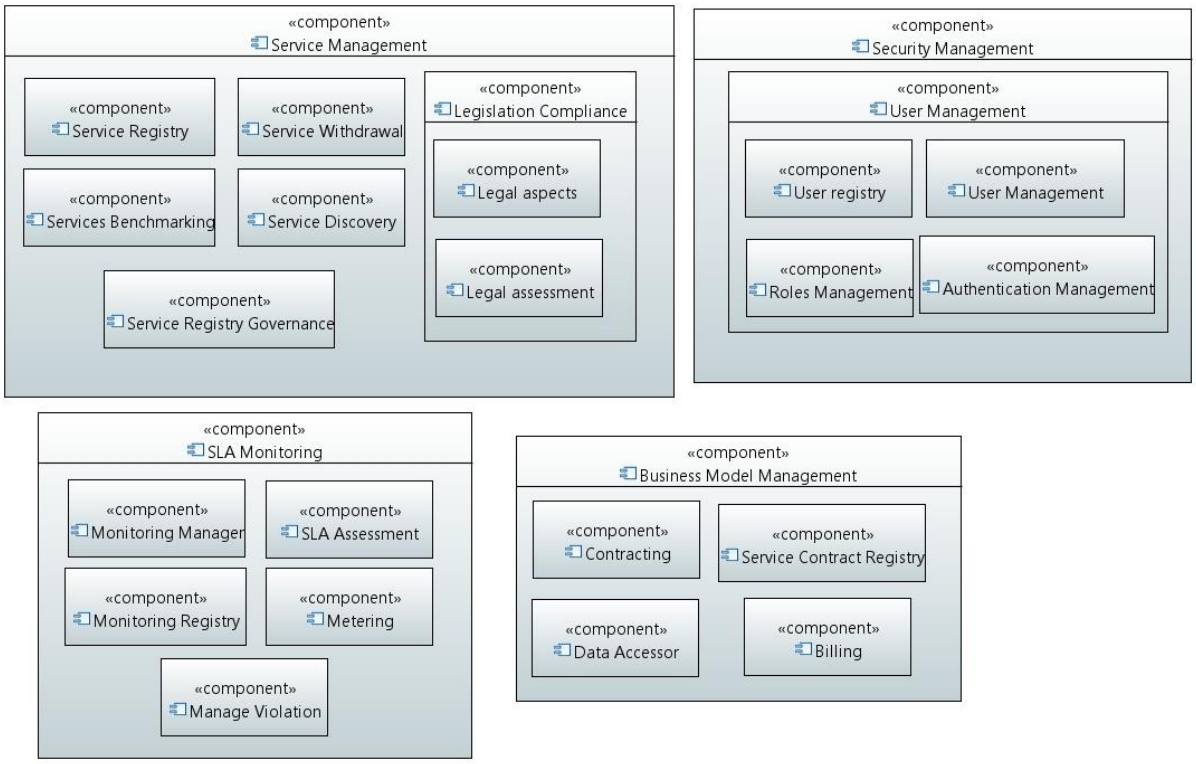


Figure 20. ACSmI High Level Architecture

There are four main components in charge of implementing the core functions of the ACSmI. Next, a high-level description of the main components and their corresponding sub-components is presented.

Service Management (ACSmI Discovery)

This component is in charge of executing and managing all the operations related to the services offered by the ACSmI. Functions like cloud services endorsement, intelligent discovery, legal compliance or service operation are covered by this component and the corresponding sub-components. The sub-modules included in Service Management are:

a. *Service Registry*: The service registry is in charge of registering all the information related to the services offered by ACSmI. The type of information to be registered will be related to the information about the NFRs.
b. *Service Registry Governance*: The Service Registry Governance is responsible for managing the access and update to the service registry.
c. *Service Discovery*: This sub-component is in charge of managing the requests from the developers (OPTIMUS) to discover the services. It gathers and processes the request from OPTIMUS when discovering services in the ACSmI.
d. *Services Benchmarking*: This sub-component will be in charge of comparing the different services and providing a shortlist of services based on the degree of the fulfilment of the requirements (functional and non-functional).
a. *Legislation Compliance:* The legislation compliance is responsible of assessing the level of compliance of the services with respect to the different legislations (i.e. GDPR[4] and Code of Conduct of CSPs). The main functionalities of this sub-module are to record of all the relevant aspects to enable the evaluations of the legal level, and to check if the information collected

[4] General Data Protection Regulation

from the CSPs accomplishes the requirements set by the applicable legislation, as requested by the user when eliciting the NFR.

Cloud service SLA monitoring

This module is in charge of the management of the monitoring of the services in the ACSmI. This module is composed by different sub-modules to perform the corresponding activities:

a. *Monitoring manager:* This sub-component is in charge of managing the different processes and requests that need to be triggered in each of the other sub-components of the component. ADAPT M manager, will launch the following processes:
   - Start/ Stop monitoring
   - Configure metering component depending on the cloud service to be monitored
   - Monitor CSPs violations
b. *Metering* collects metrics and parameters that enables to the SLA Assessment to check if the Cloud services are fulfilling the agreed SLAs.
c. *Monitoring repository* provides, assists, and automates the storage of parameters and values collected by the Metering subcomponent.
d. *SLA Assessment* calculates the metrics based on the values retrieved for the parameters by the Metering sub module and assesses the compliance of the SLA of the contracted services (contracted values vs. real values).
e. *Manage violation.* This sub module is in charge of managing that a contracted service is not fulfilling the SLA. This submodule is responsible to contact ADAPT VH and to record in the ACSmI registry this violation.

Business Model management

This core component is in charge of the execution and management of all the operations related to Service Contracts in the ACSmI. It also performs all the activities related to the financial operations with the different users of the ACSmI. The sub-modules included in this component are:

a. *Contract Manager:* It is in charge of the management of the core functions with respect to the service contracts. It manages mainly two different types of contracts: 1) contracts between the user and the ACSmI and 2) contracts between the CSPs (service providers) and the ACSmI.
b. *Service Contract Registry:* This sub-module stores the different contracts existing in the ACSmI.
c. *Billing*: It is responsible for monitoring and calculating the total values in order to bill the users for the services and to pay the CSPs for the services used.
d. *Data accessor*: It is responsible for allowing to the multi-cloud application operator to get access to the service. ACSmI shall provide the multi-cloud application operator with details of how the access can be obtained and also with the APIs required to contract the services and monitor them in different CSPs.

Security management

This component is in charge of designing and developing the means to guarantee the secure operation of the ACSmI. It includes functionalities such as identity propagation and federated authentication and authorization. This component will manage the user/developer roles in case that the ACSmI is used as a stand-alone component. If ACSmI is used together with the DECIDE framework, it only manages the CSP role. The sub-modules included in this component are:

a. *Roles Manager*: It manages the activities related to the roles in the ACSmI (creation, modification, assignment, deletion).
b. *User Manager*: It manages the activities related to the users in the ACSmI (creation, modification, roles assignment, deletion).
c. *User Registry*: It stores all the information associated to the users of the ACSmI.
d. *Authentication* Manager: This sub-module performs the authentication of the users and manages the access to the different actions/functions of the ACSmI for every user.

A detailed description and design of each component will be covered in the deliverable D5.3 [16] "Intermediate Advanced Cloud Services meta-Intermediator requirements".

The following picture presents the interfaces that ACSmI will have with other DECIDE components.

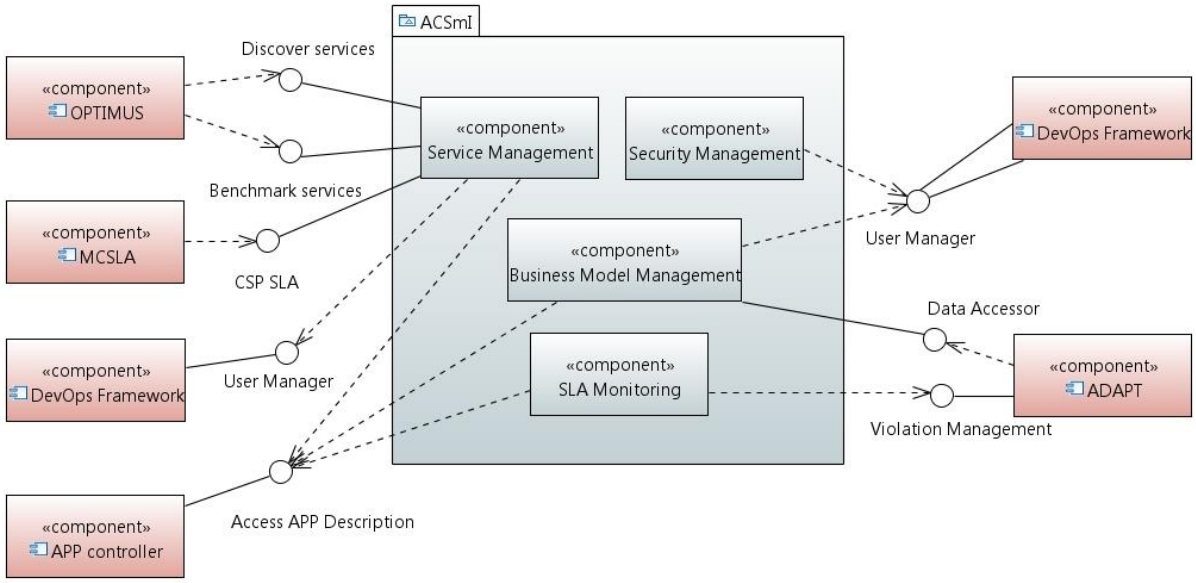


Figure 21. ACSmI external interfaces

The internal interfaces between the ACSmI components will be detailed in the deliverable D5.3 [16] "Intermediate Advanced Cloud Services meta-Intermediator requirements".

4.3.3.2 Behavioural description

The ACSmI behaviour and the interchanged data among the different actors in this part of the DECIDE workflow, is shown in Figure 22.

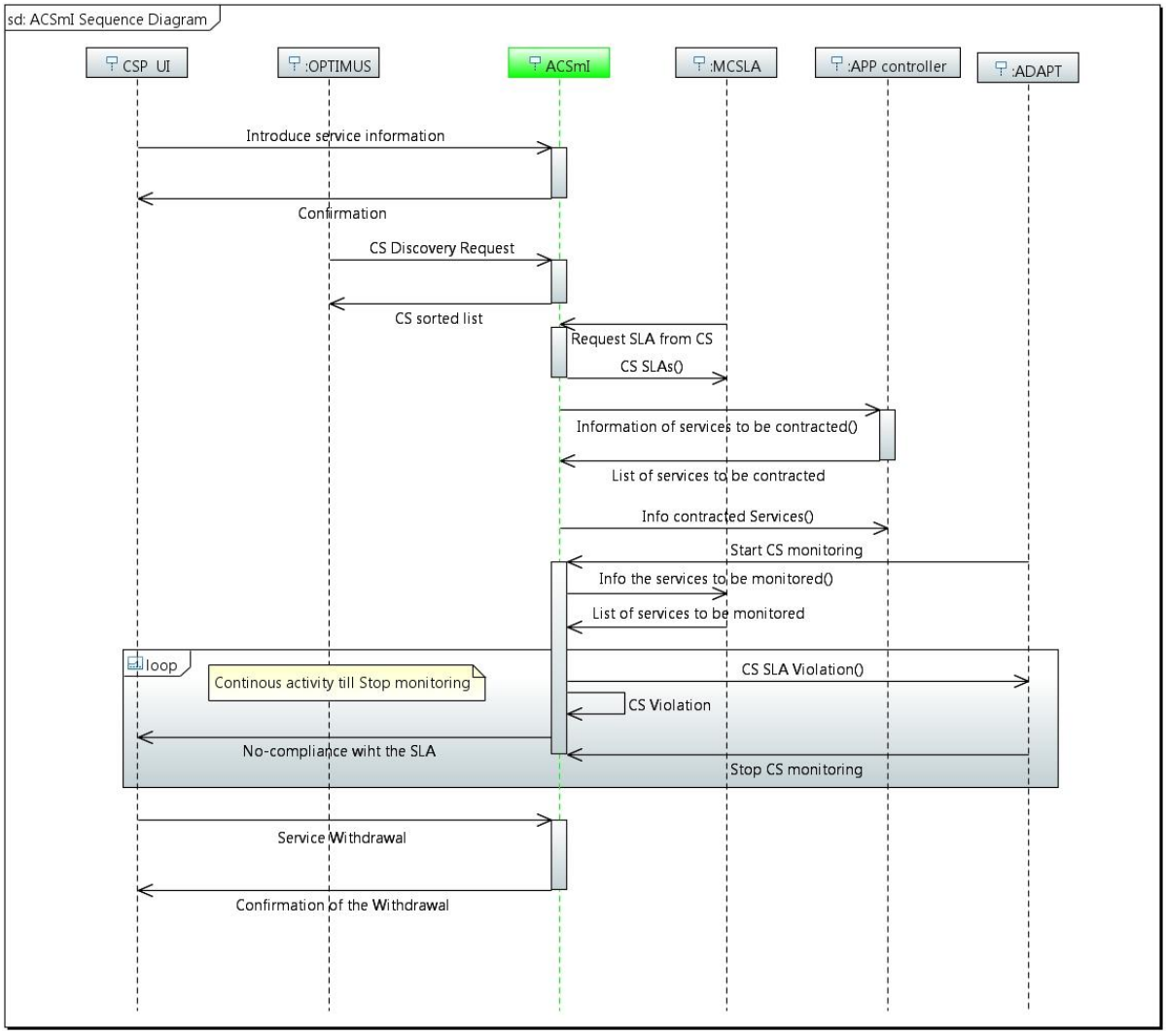


Figure 22. ACSmI Sequence Diagram

Five external components interact with ACSmI:

1. *CSP UI*. Each CSP or the *ACSmI* operator should introduce the information to maintain the *service registry* updated. This component should interact with the *service management* module. This component will also be informed if a non-compliance of the CSP SLA occurred. The CSP will have the possibility to ask for the withdrawal of a service from the service registry.
2. *OPTIMUS* will request the discovery of services that cover the requirements. Once these services are discovered by *ACSmI*, a message with the services discovered will be sent to *OPTIMUS*. This list of services will be sorted according to the level of compliance with the requirements.
3. *MCSLA* requests ACSmI the SLO of the NFRs for each of the services to be contracted, and ACSmI returns this information.
4. *Application Controller*. The *Application Controller* is used by ACSmI to read which are the services to be contracted and it includes the information required by ADAPT. Once the services have been contracted, and in a later stage once ADAPT indicates to start the CS monitoring, ACSmI uses the *Application Controller* to obtain the information on which are the services to be monitored.
5. *ADAPT*. The interaction between *ACSmI* and *ADAPT* can occur from three different ways: 1) ADAPT alerts ACSmI that the application is deployed and it is time to start the monitoring of

Cloud services, 2) ACSmI will inform ADAPT that a violation of a SLA of a cloud service has occurred and 3) ADAPT will inform ACSmI that it is required to stop the monitoring of the cloud services where a concrete application is deployed.

4.4 DECIDE Tools for multi-cloud applications continuous operation

4.4.1 ADAPT

4.4.1.1 Structural description

The main functionalities of ADAPT are the following.

- Deployment of the multi-cloud application. ADAPT generates and applies the scripts to deploy the application's components (containerized microservices) on one or multiple cloud providers, as indicated in the Application Description. The Application Description provides detailed information about the microservices and their containers, about the related CSP resources to be used, and about the MCSLA to be monitored for both the application and the underlying cloud resources. A mandatory prerequisite is that a contract for the needed CSP resources has already been signed.
- Monitoring of the application MCSLA. ADAPT also monitors the status of the deployed multi-cloud based application and verifies that the non-functional requirements and the SLOs are being fulfilled. If a violation of any of the NFRs or SLOs is detected, ADAPT monitoring components will generate the proper actions depending on each situation and context: an alert saying that the working conditions are not met will be sent to the operator, and the "adaptation" process will be launched, through the violation handlers' component.
- Adaptation of the multi-cloud application. If the application is of high technology risk, the operator will have to confirm the following redeployment configuration proposed by OPTIMUS; whereas, in case of low technology risk, the redeployment will be automatic.

The main components of ADAPT are shown in the following Figure along with their external interfaces.

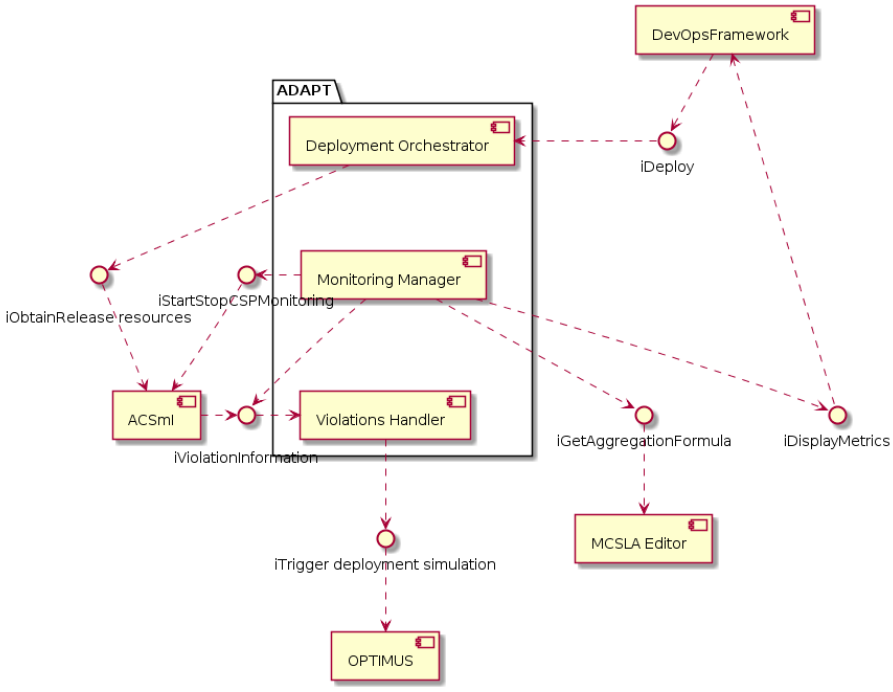


Figure 23. ADAPT Component Diagram and Interfaces

Deployment Orchestrator

The *Deployment Orchestrator* is responsible for orchestrating the deployment lifecycle (deployment, un-deployment, user confirmation, redeployment) for user applications and their components. This component gets all its input information from the Application Description.

Monitoring Manager

The *Monitoring Manager* controls the monitoring functionality for the application, according to its defined (Multi-Cloud) SLA. It identifies and raises application violations. Monitoring information for the CSPs, and related violations, is collected by ACSmI.

Violations Handler

The *Violations Handler* will handle any violation raised either by the *Monitoring Manager* or by ACSmI, regarding respectively the application MCSLA or the CSPs' NFRs. Violation handling may lead both to alerting the operator and to contacting OPTIMUS to trigger a new re-deployment simulation for the application, thus starting a re-adaptation process.

More details on ADAPT architecture can be found in DECIDE deliverable D4.2, to be released at M24 [21] .

4.4.1.2 Behavioural description

The interactions of ADAPT with other tools of the DECIDE framework are shown in the following Figure 24.

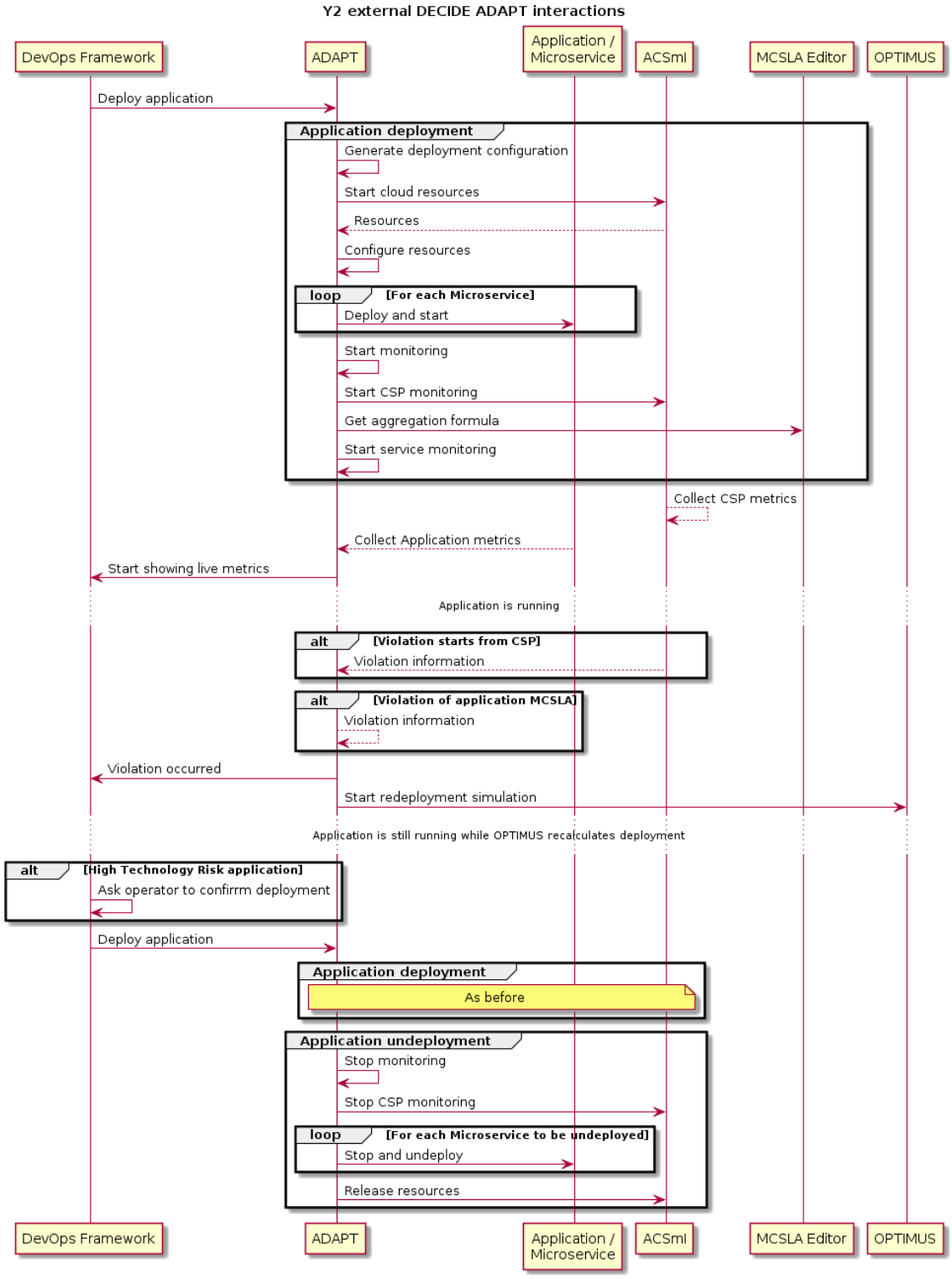


Figure 24. Interactions of ADAPT with other DECIDE tools

The main steps indicated in the sequence diagram of Figure 24 are the following.

1. When ADAPT is invoked to deploy an application, it generates a representation of the deployment configuration generated by OPTIMUS, and included in the Application Description, that can be understood by the used deployment tool (e.g. Terraform).

2. ADAPT contacts ACSmI to start the cloud resources indicated in the Application Description and then configures them.
3. ADAPT deploys and starts each application's microservice
4. ACSmI is then called to initiate CSP monitoring; the proper formulas for aggregating the collected metrics at application level are obtained by calling the MCSLA Editor, and the application service monitoring is then initiated by ADAPT.
5. While the application is running, ADAPT collects application metrics from the microservices and ACSmI from the underlying CSPs
6. Live metrics are shown through the DevOps Framework Dashboard
7. As soon as a violation is identified, the operator is informed and a new redeployment simulation is triggered by contacting OPTIMUS
8. When OPTIMUS finishes recalculating the new deployment configuration, if the application level of technology risk is defined as high, the operator must confirm the new redeployment configuration before executing the actual redeployment. In the case the level of technology is low, no confirmation is needed by the operator and the flow goes automatically to step 9.
9. In case the operator confirms, then ADAPT is invoked again to redeploy the application
10. The new configuration is deployed using the same steps as the initial deployment.
11. The previous configuration is un-deployed, after stopping its monitoring, and the related resources are released through ACSmI.

4.4.2 MCSLA Editor

The MCSLA Editor provides a tool for the authoring of an MCSLA to be used as a contract between the user of the application and the application owner, i.e. developer. Furthermore, the MCSLA is designed in a machine-readable format that describes means to monitor and measure the application's NFRs.

4.4.2.1 Structural description

The requirements elicited for the MCSLA Editor in the project are described in D2.1 [1]. These are translated into functionalities that reside in components as denoted in the component diagram (see Figure 25). The main functionalities for MCSLA Editor are:

- Provide means for the developer, to support him in the definition of the composite MCSLAs (Multi Cloud Service Level Agreement) and the corresponding SLOs (Service Level Objectives) of the application and the dependencies and needs on the underlying (combination of) cloud services in a machine-readable format for the representation.
- Provide means to translate the resulting SLA in machine readable format (based on standards) as well as a human readable format (to be shared with the end-users, i.e. customers).
- Provide a UI (through the DevOps framework) for creating/editing SLAs/MCSLA

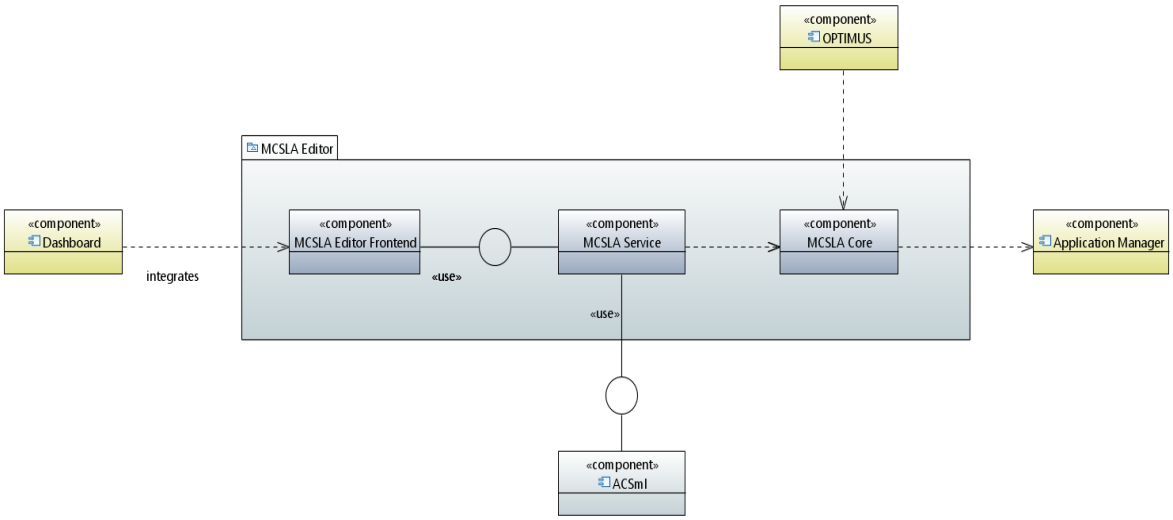


Figure 25. Component Diagram for MCSLA Editor

The MSCLA Editor is a two-tier architecture represented by the *MCSLA Frontend* and the backend consisting of the *MCSLA Service* and *the MCSLA Core library.*

MCSLA Frontend

The *MCSLA Frontend* is a user-facing component that enables the users to create, read, update and delete MCLSAs in a visual and human readable manner. The frontend will be integrated into the DevOps Framework. The Frontend communicates with the backend and uses defined interfaces for accessing available SQOs and SLOs, aggregated values of SLAs as well as existing MCSLAs. Available SLOs and SQOs are based on the ISO Standard 19086 [22] and cover terms that are application specific, rather than just provider specific.

MCSLA Service

The *MCSLA Service* is in charge of managing the MCSLA and holds its logical information model, it communicates with the code repository in order to access the *Application Description* and receive the ids of the cloud providers the multi-cloud application is deployed on.

The *MCSLA Service* uses this information from the *Application Description* to access cloud provider related information via the interfaces provided by ACSmI. This information is in turn used to identify the SLAs (SLOs) that need to be aggregated and represented in the MCSLA.

Furthermore, the *MCSLA Service* is in charge of storing a tagged version of the MCSLA in the code Repository for ADAPT to access and be able to monitor the application. For this the metric aggregation and evaluation is packaged in a separate core library that can be shared between the editor and the ADAPT Monitoring tool.

MCSLA Core Library

The *MCSLA Core library* serves the *MCSLA Service* with the SLA metrics in order for it to accumulate and aggregate the possible values for SLOs depending on the aggregation rules defined in the component.

For each deployment scenario detailed in the *Application Description* a specific aggregation rule is specified and used to aggregate the values.

4.4.2.2 Behavioural description

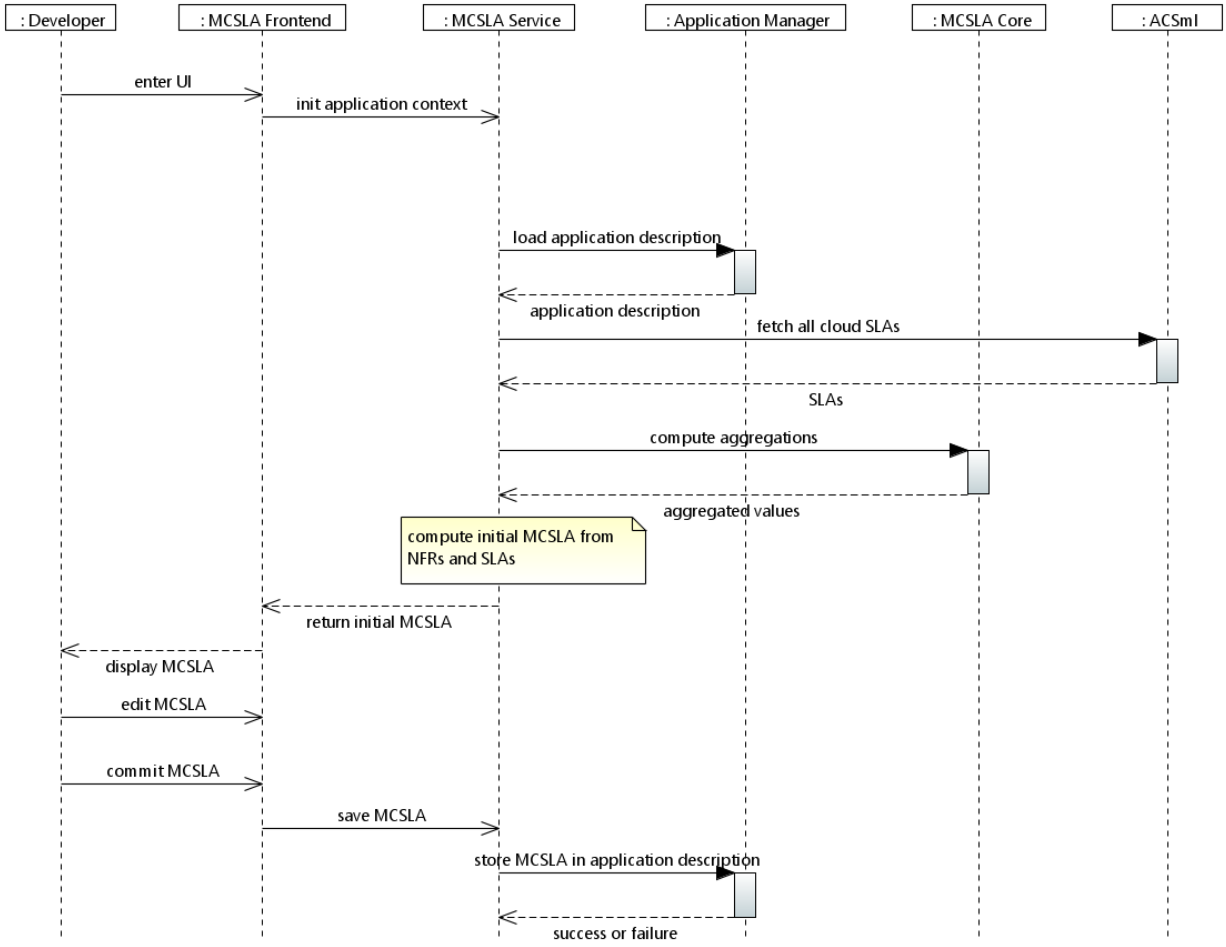


Figure 26. Sequence Diagram for creating an MCSLA

The sequences for creating an MCSLA are as follows:

1. The developer starts the *MCSLA Frontend* (GUI); this process calls the *MCSLA Service* in order to populate the front end with the necessary values.
2. As long as the *MCSLA Editor* as a whole is integrated into the DevOps Framework, it is clear which *Application Description* is applicable at this stage. The *Application Description* residing in a repository will be accessed via the *MCSLA Service* to retrieve the currently used deployment topology, i.e. the CSP Ids.
3. With the CSP Ids, the *MCSLA Service* contacts *ACSmI* in order to obtain the contracted SLAs.
4. The *MCSLA Service* then uses the *MCSLA Core* library to take the necessary measures to aggregate the SLOs defined in each SLA.
5. Once this step is completed, the *MCSLA Server* populates the frontend with the available SLO/SQOs and their possible values.
6. The developer then uses the GUI to create the MCSLA, which is finally saved by the *MCSLA Service* in the code repository as well as registering it in the *Application Description.*

5 DECIDE tool suite deployment

5.1 DECIDE tools deployment options

DECIDE tools deployment follows a containerized approach. Each component is containerized (using Docker [23] technology) and they can be deployed into different resources.

In the integration environment, these containerized tools are deployed into AIMES resources (30-35 GB disk and around 20 containers).

In the following table the different deployments options and technologies for each of the DECIDE KRs is summarized:

Table 4. DECIDE KRs' deployment options

NFR Editor	
Deployment options and technologies used	NFR editor is developed as an Eclipse plugin [24]. This plugin can be used along with ARCHITECT and OPTIMUS KRs (which are also developed as Eclipse plugins). It can be installed and deployed locally (in an Eclipse installation) using the traditional methods for the Eclipse plugins (updatesite, etc) or using the respective Docker where an installation with an Eclipse with the NFR editor installed is included.
ARCHITECT	
Deployment options and technologies used	ARCHITECT is composed of three components: A. The Cloud Patterns: A Java library that contains all patterns and the recommendation algorithm. It can be referenced as maven dependency. B. The Cloud Patterns Compendium: A wrapper microservice for the Cloud Patterns java library providing a convenient REST API. The dashboard depends on this microservice for integrating the ARCHITECT tool. It is containerized and deployable in a Docker runtime. C. The Frontend: An Eclipse plugin that can be installed through the usual eclipse update mechanism. A public accessible update site is currently not available. The eclipse plugin uses the Cloud Patterns java library and not the microservice.
DevOps Framework	
Deployment options and technologies used	The Devops Framework is composed by three main components, the front-end platform, the back-end microservices and the database. A. Front-end: It is developed using Angular 2+ and Typescript technology. The whole component has been containerized using Docker. The client directly consumes the API of the backend service for authentication, and also the microservices provided by the rest of the tools. It also uses HTML, CSS and JavaScript to develop the website. B. Back-end: It provides a set of microservices which are responsible of managing the user authentication process and the communication between tools and the application description. It has been developed in Java using Spring Boot framework, Spring Cloud for microservice communication and

	Spring Security for securing them. This component is also containerized using Docker. C. Database: This database stores all the information regarding to the user, their credentials and associated projects. The technology is a NoSQL database, more precisely MongoDB. On the other hand, there is also an isolated database focused on storing secrets, called Vault. For the deployment of the component, we have created a docker compose configuration file for setting up the environment with all the components, and automate the container creation process.
OPTIMUS	
Deployment options and technologies used	OPTIMUS is composed by two different parts: one as an eclipse plugin with the UI and the classification feature, and other part as a REST service which can be invoked by other tools. The eclipse plugin can be installed locally. The technologies used are java, eclipse and swagger for the REST service.
Application Controller	
Deployment options and technologies used	Application Controller is packaged as a java library. Tools that need this library can reference it as dependency in a maven or gradle based build process.
ACSmI	
Deployment options and technologies used	ACSmI is composed by four functional components with different options for the deployment: A. ACSmI Discovery: This component is deployed using five different containers: a. Frontend. It is responsible to implement the interface (UI and API) to allow introducing the requirements for the discovery. It is developed using JHipster [25] b. Backend. It is responsible to carry out the discovery and the endorsement of the services. It implements the services API to be used by other tools. It is developed using JHipster. c. Registry. This component is totally based on the JHipster registry, and it is responsible to communicate the frontend with the backend. d. MySQL Database structure. e. MySQL Database data provision. B. ACSmI Monitoring: ACSmI monitoring is composed by three subcomponents: a. Monitoring manager (including assessment and violation): Java server (deployed in a Jetty server), containerized using docker. b. Monitoring repository: Influx DB [26] containerized. c. Metering: Telegraf [27], containerized. C. ACSmI Contracting: the component is deployed using one container. It contains a monolith RoR application (Frontend andBackend) and SQLite3 database.

	D. ACSmI Billing: the component is at a late stage of design. It is expected to use the setup similar to ACSmI Contracting: one container with the monolith RoR application and SQLite3 database.
ADAPT	
Deployment options and technologies used	ADAPT is composed of three main components with different nature and deployment options: A. ADAPT DO: ADAPT Deployment Orchestrator (DO) is a microservice packaged into a Docker container. The default deployment is a centralized one, and can be used to deploy multiple applications. However, another deployment option is available too, with one ADAPT DO per application, deployed together with the application components. This per-application deployment option is currently supported by a specific REST call that *deploys the deployer* each time and Application is deployed by operating on a special Application Description which describes ADAPT DO itself. B. ADAPT monitoring: ADAPT monitoring is composed by four sub-components which forms the ADAPT monitoring stack. a. ADAPT monitoring manager: Java server (deployed in a Jetty server), and containerized using docker. b. ADAPT monitoring data storage and aggregation: Influx DB [26] containerized. c. ADAPT monitoring UI: Grafana [28] containerized. d. ADAPT monitoring data collection: Telegraf [27], containerized. ADAPT monitoring is deployed once and used to monitor several applications. It is a centralized component. C. ADAPT VH: It is an intermediate microservice which interacts with ADAPT monitoring manager the ACSmI monitoring, when they detect a new violation in the SLAs. It is developed in Java using the Spring framework. It is also included in a Docker container
MCSLA Editor	
Deployment options and technologies used	The MCSLA Editor is composed of two deployable components and one java library: A. mcsla-core A java library for aggregation end evaluation of service level agreement (SLA) metrics. The tools that need this library can reference it as a dependency in a maven or gradle based build process. B. mcsla-service A microservice that serves as backend. It is containerized and deployable in a Docker runtime. C. mcsla-ui A front end implemented as single page application (SPA) on top of the mcsla-service backend. The component serving this SPA is containerized and deployable in a Docker runtime.

5.2 Information exchange between DECIDE tools

The mechanisms supporting interoperability and information exchange among the different components of the tool-suite are envisioned as follows:

- Information exchange through the **Application Description**: The Application Description is the main mechanism to share information between Key Results, tools and components in DECIDE. The Application Description is a structured JSON file containing all the relevant information about the current status of the multi-cloud application, focusing on the information that is relevant for the different DECIDE Key Results, tools and components. More information about the current version of the App Description is included in the Annex 1.
- Information exchange through the DECIDE **DevOps Framework**: DECIDE DevOps Framework, will also support interoperability among DECIDE Key Results, tools and components. It integrates components at two levels:
    - At **GUI** level, all the KR's graphical interfaces are integrated in the DevOps Framework. For some of the tools, the DevOps Framework builds their GUI with the information obtained calling the tool's API. For some others, the tool provides an iframe that is embedded straight in the DevOps Framework.
    - At **information** level, exchange through direct API invocation. This approach is suitable when a direct message exchange is required between two DECIDE components. Besides, some of the information shared amongst KRs is sensible data, for which the Application Description is not the best mechanism. For this type of data, the DevOps Framework integrates Vault, a component whose goal is enabling secure sharing of secrets.

Interactions between components within the DECIDE tool-suite could be driven by:

- User interactions: When a user requests a DECIDE component. This is done through the UI of the tool itself or the integrated UI (DevOps Framework).
- Task internal transactions: When a DECIDE tool requests information from another DECIDE tool (one component invokes another component, or the invocation is done through the DevOps framework).

6 Conclusions

This document provides a detailed description of the entire DECIDE global architecture, providing a conceptual, functional and interoperable representation.

The combination of DECIDE Key Results and related tools and components supports the extended DevOps approach proposed, from the design of the multi-cloud applications to the operation and monitoring of their working conditions.

The document also provides a deeper analysis, both structural and behavioural of each DECIDE Key Result identifying message exchange dependencies, dependencies through required and exposed interfaces, and the timeline activities conducted by the tools. This in-depth analysis has enabled an earlier identification of potential misalignments between the conceptualization and the technical design of the different tools. These misalignments where identified and addressed during the specification of the first version of this architecture [2], and solved and the solutions proposed translated to the current conceptual and technical design (by their respectively work package tasks) and also to the corresponding prototypes implemented.

This detailed architecture also enabled the final agreement on the work products produced and consumed by each tool, and the agreement of the interoperability requirements between the DECIDE Key results, tools and components.

Additionally, this document addressed the challenge of supporting the deployment of the DECIDE tool-suite and the key results separately. In order to support several approaches (that is the deployment of the tools on their own and also several tools together) the containers approach has been adopted.

The architecture, deployment possibilities and interoperability requirements and mechanisms presented in this document cover the ideas, discussions and summarizes the technical decisions taken by the DECIDE partners until month 23 of the project. The architecture presented in this document will be implemented in the different intermediate and final prototypes of the KR results to be implemented in the different technical Work Packages.

7 References

[1] DECIDE consortium, "D2.1 Detailed requirements specification," 2017.

[2] DECIDE consortium, “D2.4-Detailed architecture v1,” 2017.

[3] DECIDE Consortium, “D2.6-Initial DECIDE DevOps Framework Integration,” 2018.

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

[5] DECIDE Consortium, “D3.10-Initial multi-cloud native application controller,” 2017.

[6] DECIDE Consortium, “D3.11-Intermediate multi-cloud native application controller,” 2018.

[7] DECIDE Consortium, “D3.13-Initial multi-cloud native application composite CSLA definition,” 2017.

[8] DECIDE consortium, “D3.14-Intermediate multi-cloud native application composite CSLA definition,” 2018.

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

[10] Decide Consortium, “D3.10 Initial multi-cloud native application controller,” 2017.

[11] DECIDE consortium, “D4.4-Initial multi-cloud applicatin deployment and adaptation,” 2017.

[12] DECIDE consortium, “D4.5-Intermediate multi-cloud application deployment and adaptation,” 2018.

[13] DECIDE consortium, “D4.7-Initial multi-cloud application monitoring,” 2017.

[14] DECIDE consortium, “D4.8-Intermediate multi-cloud applicatin monitoring,” 2018.

[15] DECIDE consortium, “D5.2-Initial Advanced Cloud Service meta-Intermediator (ACSmI),” 2017.

[16] DECIDE Consortium, “D5.3-Intermediate Advanced Cloud Service meta-Intermediator (ACSmI),” 2018.

[17] DECIDE Consortium, "D5.1 ACSmI requirements and technical design," 2017.

[18] DECIDE Consortium, “D3.1-Initial architectural patterns for implementation, deployment and optmization,” 2017.

[19] L. Riungu-Kalliosaari, S. MakinenSimo, L. . E. Lwakatare, T. Männistö and J. Tiihonen, “DevOps Adoption Benefits and Challenges in Practice: A Case Study,” 2016.

[20] J. Alonso, M. Escalante, L. Farid, M. J. Lopez, L. Orue-Echevarria and S. Dutkowsky, “Towards Supporting the Extended DevOps Approach through Multi-cloud Architectural Patterns for Design and Pre-Deployment,” in *SE-CLOUD 2018*, 2018.

[21] DECIDE Consortium, "D4.2 Intermediate DECIDE ADAPT Architecture," 2018.

[22] International Organization for Standardization, “ISO/IEC DIS 19086-2 Information technology -- Cloud computing -- Service level agreement (SLA) framework -- Part 2: Metric model,” ISO, 2017.

[23] Docker, “Docekr,” [Online]. Available: https://www.docker.com/resources/what-container. [Accessed 22 October 2018].

[24] Eclipse Foundation, “Wiki Eclipse,” [Online]. Available: https://wiki.eclipse.org/FAQ_What_is_a_plug-in%3F. [Accessed 22 October 2018].

[25] JHipster, “JHipster,” [Online]. Available: http://www.jhipster.tech/. [Accessed November 2017].

[26] Influxdata, “InfluxData (InfluxDB): Time Series Database Monitoring,” [Online]. Available: https://www.influxdata.com/. [Accessed October 2018].

[27] Telegraf, "Telegraf is the Agent for Collecting & Reporting Metrics & Data," [Online]. Available: https://www.influxdata.com/time-series-platform/telegraf/. [Accessed November 2017].

[28] Grafana Labs, “Grafana,” [Online]. Available: https://grafana.com/. [Accessed 18 October 2018].

[29] DECIDE consortium, “D3.8-Intermediate DECIDE OPTIMUS,” 2018.

[30] DECIDE Consortium, “D4.1 Initial DECIDE ADAPT Architecture,” 2017.

Annex 1: App Description

This annex includes the current version of the App Description (M24)[5]. As explained in section 4, the App Description is one of the main means in DECIDE for interoperability between the different components.

This version of the App Description, resulting from Partners' research and discussions, includes relevant information for each of the DECIDE components, with a brief description for each field. Please note that the Application Description definition has evolved since the first data model shown in the DECIDE deliverables D2.1 [1], D2.4 [2] and it is still evolving in parallel with the design and implementation iterations.

Table 5. Application Description model

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component
id			String	1	Unique identifier for the Application Description	DevOps Framework
name			String	1	Name of the application	DevOps Framework
consulJoinIp (New)			String	1	Address of the master Consul (service registry) node	TBD: it will be the address of a node running ADAPT
description			String	1	Textual description of the application	DevOps Framework
version			String		Indicates the version number of the app description "schema", for compatibility purposes	DevOps Framework
highTechnologicalRisk			Boolean	1	Indicates if the application has high technological risk: confirmation for (re)deployment is needed	DevOps Framework
jenkinsEndpoint (NEW)			String	1		DevOps Framework
jenkinsToken (NEW)			String (uri)			DevOps Framework
nfrs (NEW)			Array of Objects (any of the	1..*	List of nfrs	DevOps Framework/General editor

[5] This version corresponds to the 31st of October 2018, when this document was last updated. The information of the App Description is evolving and will change as the technical discussions advance.The updated schema for the application description is stored in the private DECIDE software repository: https://git.code.tecnalia.com/decide/AppController/blob/master/src/main/resources/application_description.schema.json

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component
			following: availabilityNfr, performanceNfr, ScalabilityNfr,LocationNfr,CostNfr)			
	availabilityNfr				Note: this line does not indicate a JSON field, but only a possible type of item in the nfr array	
		type (NEW)	String	1..1		DevOps Framework/General editor
		tags	Array of Strings	0..*		DevOps Framework/General editor
		abstractValue (NEW)	Enum	1..1	• Low • Medium • High	DevOps Framework/General editor
		value (NEW)	Number	0..1		DevOps Framework/General editor
		unit	String	0..1		DevOps Framework/General editor
	performanceNfr				Note: this line does not indicate a JSON field, but only a possible type of item in the nfr array	
		type (NEW)	String	1..1		DevOps Framework/General editor
		tags	Array of Strings	0..*		DevOps Framework/General editor
		abstractValue (NEW)	Enum	1..1	• Low • Medium • High	DevOps Framework/General editor
		value (NEW)	Number	0..1		DevOps Framework/General editor
		unit	String	0..1		DevOps Framework/General editor

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component
	scalabilityNfr				Note: this line does not indicate a JSON field, but only a possible type of item in the nfr array	DevOps Framework/General editor
		type (NEW)	String	1..1		DevOps Framework/General editor
		tags	Array of Strings	0..*		DevOps Framework/General editor
		abstractValue (NEW)	Enum	1..1	• Low • Medium • High	
		value (NEW)	Number	0..1		DevOps Framework/General editor
		unit	String	0..1		DevOps Framework/General editor
	locationNfr				Note: this line does not indicate a JSON field, but only a possible type of item in the nfr array	
		type (NEW)	String	1..1		DevOps Framework/General editor
		tags	Array of Strings	0..*		DevOps Framework/General editor
		abstractValue (NEW)	Enum	1..1	• Single Location • Single Country • Cross Country	DevOps Framework/General editor
		value (NEW)	Array of Strings	0..*		DevOps Framework/General editor
	costNfr				Note: this line does not indicate a JSON field, but only a possible type of item in the nfr array	
		type (NEW)	String	1..1		DevOps Framework/General editor
		tags	Array of Strings	0..*		DevOps Framework/General editor
		abstractValue (NEW)	Enum	1..1	• Low • Medium • High	

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component
		value (NEW)	Number	0..1		DevOps Framework/General editor
		unit	String	0..1		DevOps Framework/General editor
schema (NEW)			Array of Objects	1..*	Deployment schema	OPTIMUS
	microservices		Array of Strings	1..*	Microservices id	OPTIMUS
	csId		String	1	Cloud Services id	OPTIMUS
	index		Number	1		OPTIMIUS
microservices			Array of Objects	1..*	List of microservices	DevOps Framework /OPTIMUS
	id		String	1	Unique Identifier for the microservice	DevOps Framework /OPTIMUS
	name		String	1	Human readable name for the microservice	Creation Wizard/OPTIMUS
	classification (NEW)		String	1	Classification specification by OPTIMUS	OPTIMUS
	dependencies (NEW)		Array of strings			DevOps Framework/General editor
	sourceRepository (NEW)		String (uri)	1		DevOps Framework/General editor
	safeMethods (NEW)		Array of strings			DevOps Framework/General editor
	stateful		Boolean	1	Is the microservice stateful or stateless?	DevOps Framework /OPTIMUS
	programmingLanguage		String	1	Type of programming language used for microservice (hint)	DevOps Framework d/OPTIMUS
	containerId (NEW)		String	1		To be defined
	containerRef (NEW)		String	1		To be defined
	tags (NEW)		Array of strings		Tags to relate NFRs with microservices	ARCHITECT
	publicIP		Boolean	1	True if the microservice has a public IP address	OPTIMUS Classification

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component
	endpoints		Array of Objects	1..*	List of URI to access the services and their methods[6]	DevOps Framework
	deploymentorder (NEW)		number	1		DevOps Framework d/OPTIMUS
	infrastructureRequirements		Object	1	Minimum and maximum requirements for Disk and RAM	
		minDisk	Integer	1		OPTIMUS
		maxDisk	Integer	1		OPTIMUS
		minRam	Integer	1		OPTIMUS
		maxRam	Integer	1		OPTIMUS
	detachable_resource		Array of Objects	1…*	list of elements that are going to be used by the microservice as for example external DB services	OPTIMUS Classification
		id(NEW)	String	1		OPTIMUS Classification
		name (NEW)	String	1		OPTIMUS Classification
		db (NEW)	Boolean	1		OPTIMUS Classification
		sql (NEW)	Boolean	1		OPTIMUS Classification
		classification (NEW)	String	1		OPTIMUS Classification
recommendedpatterns (NEW)			Array of Objects	0..*	List of patterns applied to the application	ARCHITECT
	title (NEW)		String	1	Title of the pattern	ARCHITECT
	uriRef (NEW)		String (uri)	1		ARCHITECT
	positivelmpacts (NEW)		Array of Strings	0..*		ARCHITECT
	categories (NEW)		Array of Strings	0..*		ARCHITECT
	selected (NEW)		Boolean	1		ARCHITECT

[6] Port numbers in each URI are those exposed by the microservice, the container can be configured to map them to a different port number

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component
monitoring (NEW)			Object			
	status (NEW)		Boolean	1	Monitoring status. If monitoring is activated, the status is true	ADAPT
	urls (NEW)		Array of Strings (uri)	0..*	Urls with the ADAPT monitoring UI	ADAPT
mcsla (NEW)			Object			
	sla (NEW)		Object	1		
		description	String	1		MCSLA Editor
		visibility	String	1		MCSLA Editor
		validityPeriod	Integer	1		MCSLA Editor
		coveredServices	Array of Strings	0..*		MCSLA Editor
		objectives	Array of Objects [7]	0..*	The list of service objectives as part of this SLA	MCSLA Editor
	csSlas (NEW)		Object		Map between cloud services and Slas. Keys are the cloud service ids	
		description	String	1		MCSLA Editor
		visibility	String	1		MCSLA Editor
		validityPeriod	Integer	1		MCSLA Editor
		coveredServices	Array of Strings	1		MCSLA Editor
		objectives	Array of Objects[8]	1		MCSLA Editor
virtual_machines			Array of Objects	0..*	Description of the VMs that will host the containers	
	id		String	1		ACSmI/OPTIMUS
	cspName		String	1	Name of the CSP providing this VM	ACSmI/OPTIMUS
	cspId		String	1	Internal UUID for the CSP providing this VM	ACSmI/OPTIMUS
	ram		Integer	1	Amount of memory (in GB)	ACSmI/OPTIMUS

[7] This object is described in the next tables

[8] This object is described in the next tables

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component
	cores		Integer	1	Number of cores	ACSmI/OPTIMUS
	storage		Integer	1	Amount of disk space (in GB)	ACSmI/OPTIMUS
	image		String	1	Name of the VM image (identifies also the OS and its version)	ACSmI/OPTIMUS
	vmSoftwareId (New)		String	1	Id of the software resource from the ACSmI registry. Represents the OS and version of the VM (e.g. "Ubuntu 16.04")	ACSmI
	VmResourceId (New)		String	1	The id of the ACSmI VM resource, which represents the underlying CSP that will perform the real provisioning	ACSmI
	vmRegionId (New)		String	1	The id of the "Region" where the VM will run, taken from the ACSmI registry (E.g.: Zrh, US Standard, …)	ACSmI
	instanceTypeId (New)		String	1	The id of the "instanceType" which represents the combination of resources allocated to the vm (e.g. "2 Total cores, 2GB RAM)	ACSmI
	keyPairId (New)		String	1	The id of the keypairs needed to access ACSmI resources (associated to the ACSmI user profile)	ACSmI
	openedPorts		Array of integers			To be defined
	dockerPrivateRegistryIp (New)		String	0..1	IP of a Docker private registry, which will host custom container image prepared by a developer that are not published to the public Docker Hub repository. Used for configuring the VM	Developer
	dockerPrivateRegis		Integer	0..1	Port of the private Docker registry. Used for configuring the VM	Developer

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component
	tryPort (New)					
	dockerHostNodeName (New)		String	1	Name of the Docker node (referenced by the same field in each container definition)	Developer / OPTIMUS
	dockerHostPublicIp (New)		String	0..1	IP address of the Docker node (written by ADAPT DO, used by ADAPT MM)	ADAPT
containers			Array of Objects	1..*	Description of the containers that will host the microservices	
	containerName		String	1	Name of the container	Developer (DevOps Framework)
	imageName		String	1	Name of the container image	Developer (DevOps Framework)
	imageTag		String	1	Tag to identify the container in the registry	Developer (DevOps Framework)
	dockerPrivateRegistryIp		String	0..1	IP of a Docker private registry, which will host custom container image prepared by a developer that are not published to the public Docker Hub repository	Developer (DevOps Framework)
	dockerPrivateRegistryPort		String	0..1	Port of the private Docker registry	Developer (DevOps Framework)
	dockerPrivateRegistryUser		String	0..1	Username to access the private Docker registry	Developer (DevOps Framework)
	dockerPrivateRegistryPassword		String	0..1	Password to access the private Docker registry	Developer (DevOps Framework)
	hostname		String	1	Hostname of the container	Developer (DevOps Framework)
	restart		String	1	Attribute indicating the restart policy for this container (e.g. “always”)	Developer (DevOps Framework)
	command		Array of Strings	0..*	Comma-separated list of commands to be passed to the container,	Developer (DevOps Framework)

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component
					as for the "CMD" Dockerfile specs	
	entrypoint		Array of Strings	0..*	Comma-separated list of commands and parameter to be passed to the container, as for the "ENTRYPOINT" Dockerfile specs	Developer (DevOps Framework)
	dockerHostNodeName		String	1	Name of the VM hosting the container	
	networks		Array of Strings	0..*	This field will trigger the creation of one or more dedicated Docker network(s) on the container to allow two containers to see each other in case it does not exist	
	volumeMapping		Array of Objects	0..*	Mapping of volumes from host paths to container paths	Developer
		hostPath (New)	String	1	Path on the host	Developer
		ContainerPath (New)	String	1	Path on the container	Developer
	environment		Array of Strings	0..*	List of comma-separated KEY=VALUE environment variables to be defined before starting the container, as for the "ENV" Dockerfile specs	Developer
	consulKvProviderNodeName		String	1	Name of the node hosting the Consul Key-Value provider	(TBD: it will be the node running ADAPT)
	addConsulService		Integer	0..1	Specify whether to register the service to a Consul service registry (enables basic health-check)	(TBD: it may be enabled by default)
	addConsulTraefikRules		Boolean (0|1)		Specify whether to add reverse proxy routing rules to the Consul K/V store (based on "Host:" header)	Developer

Field name	Nested Elements	Nested Elements	Type	Multiplicity/ Default	Description	Responsible component
	consulServicePort (New)		Integer	0..1	Port number for the service to be registered in Consul; usually the same as endpoints.port	
	portMapping		Array of Objects	0..*	List of ports to be published by this container	Developer
		hostPort (New)	String	1	Port to be exposed on the host	Developer
		ContainerPort (New)	String	1	Port exported by the container	Developer
	hostMapping		Array of Objects	0..*	Optional mapping to support a reverse proxy	Developer
		Ip (New)	String	1		Developer
		ReferencedContainerName (New)	String	1		Developer
		Host (New)	String	1		Developer
	endpoints		Array of Objects	0..*	List of endpoints for this container	Developer
		protocol	String	1	Typically “http”, but can be something else according to URL syntax	Developer
		port	Integer	1	The port to which the endpoint is bound	Developer
		skipRule	Boolean (0|1)	0..1	Set to 1 to discard the routing rule based on hostname (“Host:” header)	Developer
		containerNameOverride	String	0..1	Overrides the standard routing rule based on hostname; hence, it allows to consider a different hostname for this service	Developer
applicationInstanceId (New)			String	1	Application ID (written by ADAPT DO, used by ADAPT MM)	ADAPT

The following tables describe some of the sub-elements of the Application Description model for *mcsla* elements. Table 2 describes the nested elements for the sub-element *objectives* of the Application Description. The MCSLA Editor is responsible for eliciting this information from the user.

Table 6. MCSLA *objectives* object description (nested elements for “objectives”)

Element Name	Serviceobjectives		
Description			
attribute -or-Element	Type	Multiplicity / Default	Definition
type	String	1	If it is an SLO or an SQO
termName	String	1	The term name of the service objective, e.g. “Availability”
comment	String	1	Some further explanation
violationTriggerRules	Array of Objects	1..*	Tracking of violation events
metrics	Array of Objects	1..*	The list of metrics for this service objective
remedy	Array of Objects	1..*	Some optional remedy

The following table describes the sub-elements nested in the *violationTriggerRule* sub-element of *microservice_SLO* and *microservice_SQO*.

Table 7. Nested elements for *violationTriggerRule*

Element Name	violationTriggerRule		
Description	The general information about the violation trigger rule		
attribute -or-Element	Type	Multiplicity / Default	Definition
violationInterval	number	1	Indicates the monitoring frequency for each SLO
breaches_count	number	1	The count of how many breaches have taken place

The following table holds the fields (taken directly from ISO 19086-2 Metric Model [22]) that are nested within the *metrics* field *of microservice_SLO* and *microservice_SQO*. The MCSLA Editor is responsible for eliciting this information from the user.

Table 8. MCSLA *metric* data model for monitoring

Element Name	metrics

Description	The general information about the metric		
attribute -or- Element	**Type**	**Multiplicity / Default**	**Definition**
descriptor	String	0..1	a short description of the metric
id	String	1	a unique identifier for the metric within a context
source	String	1	the individual or organization who created the metric
scale	String (enumeratedList)	1	classification of the type of measurement result when using the metric. The value of scale shall be “nominal, ordinal, interval, or ratio”. SLOs shall use either the “interval” or “ratio” scale. SQOs shall use the “nominal” or “ordinal” scales.
note	String	0..1	additional information about the metric and how to use it.
category	String	0..1	a grouping of metrics with similar expressions, rules, and parameters
expression	Object	0..1	The expression of the calculation of the Metric and supporting information. An SLO metric shall have an expression while an SQO may or may not have an expression (e.g., specified using natural language). It shall be written using the ids to represent UnderlyingMetrics, Parameters, and Rules.
parameters	Array of Objects	0..*	a Parameter is used to define a constant (at runtime) needed in the expression of an Metric. A

			Parameter may be used by more than one Metric if it is defined using a unique ID within the set of metrics it is used in.
rules	Array of Objects	0..*	a Rule is used to constrain a Metric and indicate possible method(s) for measurement.
underlyingMetrics	Array of Objects metrics / Strings	0..*	a metric element that is used within an expression element to define a variable. The Expression shall use the Underlying Metric id to reference the Underlying metric within the expression.

The following table describes the sub-elements nested in *expression* sub-element.

Table 9. Nested elements for *expression*

Element Name	**expression**		
Description	The expression of the calculation of the **Metric** and supporting information		
attribute -or-Element	**Type**	**Multiplicity / Default**	**Definition**
id	String	1	a unique identifier (within the context of the metric) for the expression
expression	String	1	the expression statement written using the ids to represent UnderlyingMetrics, parameters, and rules.
expressionLanguage	String	1	the language used to express the metric (i.e. ISO80000)
note	String	0..1	additional information about the expression
unit	String	0..1 required when scale is ratio or interval	real scalar quantity, defined and adopted by convention, with which any other quantity of the same kind can be compared to express the ratio of the two quantities as a number.

subExpression	Array of Expression Objects	0..*	an associated element of type element that is used within the expression to define a variable. The Expression shall use the SubExpression id to reference the SubExpression within the expression.

The following table describes the sub-elements nested in *parameter* sub-element.

Table 10. Nested elements for *parameter*

Element Name	**parameter**		
Description	A Parameter is used to define a constant (at runtime) needed in the expression of a Metric. A Parameter may be used by more than one Metric if it is defined using a unique ID within the set of metrics it is used in.		
attribute -or-Element	**Type**	**Multiplicity / Default**	**Definition**
id	String	1	the unique identifier of the parameter
parameterStatement	String	1	the statement or value of the parameter
unit	String	1	the unit of the parameter
note	String	0..1	additional information about the parameter

The following table describes the sub-elements nested in *rule* sub-element.

Table 11. Nested elements for *rule*

Element Name	**rule**		
Description	A Rule is used to constrain a Metric and indicate possible method(s) for measurement. For instance, an "AvailabilityDuringBusinessHour" Metric could be defined with a scope that constrains some piece of a generic "Availability" Metric element that limits the measurement period to defined business hours. A Rule describes constraints on the metric expression. A constraint can be expressed in many different formats (e.g. plain English, ISO 80000, SBVR)		
attribute -or-Element	**Type**	**Multiplicity / Default**	**Definition**
id	String	1	the unique identifier for the rule
ruleStatement	String	1	a constraint on the metric
ruleLanguage	String	1	the language used to express the rule in the ruleStatement
Note	String	0..1	additional information about the rule

The following table describes the sub-elements nested in *remedy* sub-element.

Table 12. Nested elements for *remedy*

Element Name	**remedy**		
Description	The general information about the compensation available to the cloud service customer in the event the cloud service provider fails to meet a specified cloud service level objective		
attribute -or-Element	**Type**	**Multiplicity / Default**	**Definition**
type	String	1	The type of remedy the cloud service provider will be offering the cloud service customer
value	Integer	1	The value of the type of remedy offered by the cloud service provider
Unit	String	1	The unit for the value offered
validity	Integer	1	The validity period for this remedy