



---

---

## **Deliverable D3.13**

---

---

### **Initial multi-cloud native application composite CSLA definition**

---

---

<b>Editor(s):</b>	Majid Salehi Ghamsari, Lena Farid
<b>Responsible Partner:</b>	Fraunhofer
<b>Status-Version:</b>	Final – v1.0
<b>Date:</b>	30/11/2017
<b>Distribution level (CO, PU):</b>	CO

<b>Project Number:</b>	GA 726755
<b>Project Title:</b>	DECIDE

<b>Title of Deliverable:</b>	D3.13 Initial multi-cloud native application composite CSLA definition
<b>Due Date of Delivery to the EC:</b>	30/11/2017

<b>Workpackage responsible for the Deliverable:</b>	WP3 - Continuous Architecting
<b>Editor(s):</b>	Fraunhofer
<b>Contributor(s):</b>	Majid Salehi Ghamsari, Lena Farid (Fraunhofer)
<b>Reviewer(s):</b>	Marisa Escalante (TECNALIA)
<b>Approved by:</b>	All Partners
<b>Recommended/mandatory readers:</b>	WP5, WP4, WP2

<b>Abstract:</b>	This software deliverable will comprise the initial version of a tool to derive composite SLAs from elementary ones. For this, a description formalism will be defined and extended if needed. Range definition for SLA metric values, composition and matching rules will be defined and implemented.
<b>Keyword List:</b>	MCSLA, SLA, SLO, SQO, Editor, Multi Cloud,
<b>Licensing information:</b>	The software is licensed under the Eclipse Public License version 2.0  The document itself is delivered as a description for the European Commission about the released software, so it is not public.
<b>Disclaimer</b>	This deliverable reflects only the author's views and views and the Commission is not responsible for any use that may be made of the information contained therein

---

## Document Description

---

### Document Revision History

Version	Date	Modifications Introduced	
		Modification Reason	Modified by
v0.1	21/08/2017	First draft version TOC and MCSLA data model	FHG
V0.2	21/11/2017	Second draft version, section 3 functional architecture	FHG
V0.3	22/11/2017	Third version, section 3.2 technical description	FHG
V0.4	24/11/2017	4 <sup>th</sup> version, included section MCSLA Concept and Aggregation patterns	FHG
V0.5	25/11/2017	5 <sup>th</sup> version, introduction, conclusion and completed section 4 and 3.2	FHG
V0.6	25/11/2017	Internal quality assurance review	TECNALIA
V0.7	26/11/2017	Review comments addressed	FHG
V1.0	26/11/2017	Ready for submission	TECNALIA

---

## Table of Contents

---

Table of Contents .....	4
List of Figures.....	5
List of Tables.....	5
Terms and abbreviations.....	6
Executive Summary .....	7
1 Introduction.....	8
1.1 About this deliverable .....	8
1.2 Document structure .....	8
2 MCSLA Concept .....	9
2.1 Make up of a MCSLA .....	9
2.2 SLA Aggregation Patterns.....	10
2.2.1 SLA Aggregation Patterns for Availability.....	12
3 Implementation.....	15
3.1 Functional description.....	15
3.1.1 Fitting into overall DECIDE Architecture .....	16
3.2 Technical description.....	16
3.2.1 Prototype Architecture.....	16
3.2.2 Technical Specification .....	19
3.2.2.1 MCSLA Frontend.....	19
3.2.2.2 MCSLA Editor Backend .....	20
3.2.2.3 MCSLA Data Model.....	21
4 Delivery and usage .....	28
4.1 Package information .....	28
4.2 Configuration and Installation instructions.....	28
4.2.1 MCSLA Editor (Frontend).....	28
4.2.1.1 MCSLA Editor UI configuration.....	28
4.2.1.2 Build the UI .....	28
4.2.1.3 Install and Run .....	29
4.2.2 MCSLASService (Backend).....	29
4.2.2.1 MCSLASService Configuration .....	29
4.2.2.2 Build the MCSLASService.....	30
4.2.2.3 Install and Run .....	30
4.3 User Manual .....	30
4.4 Licensing information .....	32
4.5 Download .....	33
5 Conclusions.....	34

5.1 Future work .....	34
References.....	35

---

## List of Figures

---

<b>FIGURE 1.</b> CONCEPTUAL IDEA – MAKE UP OF AN MCSLA.....	10
<b>FIGURE 2.</b> TERM COMPOSITION USING AGGREGATION PATTERNS [2] .....	11
<b>FIGURE 3.</b> BASIC MULTI-CLOUD DEPLOYMENT TOPOLOGY .....	12
<b>FIGURE 4.</b> MULTI-CLOUD REPLICATION DEPLOYMENT TOPOLOGY .....	13
<b>FIGURE 5.</b> DIFFERENT VENDORS TOPOLOGY.....	14
<b>FIGURE 6.</b> COMPONENT DIAGRAM FOR MCSLA EDITOR.....	17
<b>FIGURE 7.</b> SEQUENCE DIAGRAM FOR CREATING AN MCSLA .....	18
<b>FIGURE 8.</b> MCSLA FRONTEND PROTOTYPE IMPLEMENTATION M12 .....	19
<b>FIGURE 9.</b> MCSLA SERVICE BACKEND PROTOTYPE IMPLEMENTATION M12.....	20
<b>FIGURE 10.</b> MCSLA DATA MODEL.....	21
<b>FIGURE 11.</b> MCSLA EXAMPLE IN JSON FORMAT .....	27
<b>FIGURE 12.</b> MCSLA EDITOR UI IN ACTION.....	29
<b>FIGURE 13.</b> MCSLA BACKEND IN ACTION .....	30
<b>FIGURE 14.</b> PATH TO THE PROJECT REPOSITORY AND THE USER NAME.....	31
<b>FIGURE 15.</b> UI CREATE NEW MCSLA.....	31
<b>FIGURE 16.</b> SELECT SLO/SQO/AG AND PUT IN THE MCSLA.....	32
<b>FIGURE 17.</b> EDIT MCSLA PROPERTIES.....	32
<b>FIGURE 18.</b> BUTTON TO SAVE MCSLA.....	32

---

## List of Tables

---

<b>TABLE 1.</b> FUNCTIONALITY COVERED IN M12 IN RELATIONSHIP TO THE ELICITED REQUIREMENTS .....	15
<b>TABLE 2.</b> REST INTERFACES PROVIDED BY THE BACKEND -v1.....	21
<b>TABLE 3.</b> APPLICATION DESCRIPTION MODEL FOR MONITORING THE APPLICATION VIA ITS MCSLA (NESTED ELEMENTS FOR “APP_MCSLA”).....	22
<b>TABLE 4.</b> NESTED ELEMENTS FOR MICROSERVICE_SLAS .....	22
<b>TABLE 5.</b> NESTED ELEMENTS FOR MICROSERVICE_SLO AND MICROSERVICE_SQO .....	22
<b>TABLE 6.</b> NESTED ELEMENTS FOR VIOLATIONTRIGGERRULE .....	23
<b>TABLE 7.</b> NESTED ELEMENTS FOR REMEDY .....	23
<b>TABLE 8.</b> MCSLA METRIC DATA MODEL FOR MONITORING.....	24
<b>TABLE 9.</b> NESTED ELEMENTS FOR EXPRESSION .....	25
<b>TABLE 10.</b> NESTED ELEMENTS FOR PARAMETER .....	25
<b>TABLE 11.</b> NESTED ELEMENTS FOR RULE.....	26

---

## Terms and abbreviations

---

API	Application Programming Interface
CRUD	Create, read, update, delete
CSP	Cloud Service Provider
EC	European Commission
IaaS	Infrastructure as-a-Service
JSON	JavaScript Object Notation
KR	Key Result
MCSLA	Multi-cloud application service level agreement
PaaS	Platform-as-a-Service
QoS	Quality of Service
REST	Representational State Transfer
SaaS	Software as-a- Service
SLA	Service Level Agreement
SLO	Service Level Objective
SQO	Service Qualitative Objective

## Executive Summary

The MCSLA Editor plays a vital role in the DECIDE project as it defines the agreement between the multi-cloud native application developer and the end-user of the app services. Furthermore, it is in a standards-based machine-readable form that allows for other DECIDE tools such as ADAPT to monitor the application and assess whether the expected QoS is guaranteed.

The document at hand describes the very first version of the prototype and the representation of the machine-readable MCSLA definition. The prototype includes a backend and frontend that communicate via a RESTful interface. The prototype allows developers to define in a graphical way the MCSLA.

The document also describes the conceptual work done for the MCSLA. This includes the makeup of the MCSLA and its properties. Furthermore, the functional and technical properties of the prototype are laid out along with the build and installation instructions. A user manual is added to the document to explain the usage of the user frontend.

In the next version of the prototype of the MCSLA Editor will be integrated into the DevOps Framework and will include more functionality, such as support for more aggregation patterns as well as be connected to the interfaces of ACSml.

# 1 Introduction

## 1.1 About this deliverable

The document at hand represents the documentation for the prototype delivered at M12 for the task T3.5 Multi-cloud native application composite SLA description. It also presents concepts that have been defined within this task for the MCSLA definition.

## 1.2 Document structure

This document is divided into four main sections. Section 2 presents the MCSLA concept defined in the project. It describes the makeup of an MCSLA, its properties and the Aggregation Patterns for the different deployment topologies. Section 3 describes the implementation details from a functional and technical perspective and section 4 describes the build and installation instructions as well as the user manual for using the tool.

Finally, at the end of the document section 5 concludes on the outcome of M12 and presents future work to be done.



## 2 MCSLA Concept

It is defined in the DECIDE project that once the multi-cloud native application is implemented and is ready for deployment, i.e. the most optimal deployment topology has been selected; an MCSLA has to be defined. The purpose of the MCSLA is twofold i) it acts as the contract between the end-users and the developer of the multi-cloud native application and ii) it is used for monitoring purposes by ADAPT and will be assessed in runtime to ensure it is being accomplished.

In order for the latter to be realised the task T3.5 of WP3 is responsible for implementing the following two main points:

1. Enable the seamless composition of an MCSLA via an Editor. This should also support the composition of MCSLAs when an application is self-adapted to a new deployment topology.
2. Define a standards-based machine-readable format for an MCSLA in order to be processed by the DECIDE tools.

### 2.1 Make up of a MCSLA

The accumulation of a number of SLAs from different CSPs is defined in the DECIDE project as a multi-cloud native application composite SLA (MCSLA).

A cloud SLA is typically composed of a number of Service Qualitative Objectives (SQO) and Service Level Objectives (SLO) as defined in ISO/IEC 19086-1 [1]. The SLOs and SQOs represent, among others, the non-functional requirements of an application and its underlying infrastructure. We will refer to them as terms.

In the multi-cloud context, the deployment of the micro services of an application takes place on several CSPs. Each CSP contracted shares with the developer an SLA that guarantees an expected quality of service (QoS). Therefore, in a multi-cloud deployment scenario there will be at least two of these agreements. These agreements might differ in their content but might also include same terms (SLOs or SQOs) but with different values.

An MCSLA must therefore act as an aggregator of all terms defined in the various SLAs. If a term occurs in several different SLAs, the values of the term must be aggregated (based on defined mathematical functions). For example, if the SLO Availability occurs in one SLA with a value of 99% and in another with a value of 99%. Then the MCSLA should contain the SLO Availability with the value of 98% (formula is presented in Section 2.2). This is in essence the maximum value for availability that the developer may offer to the end-user, as it is not guaranteed that an outage would take place across all microservices (or CSPs for that matter) at the same time.

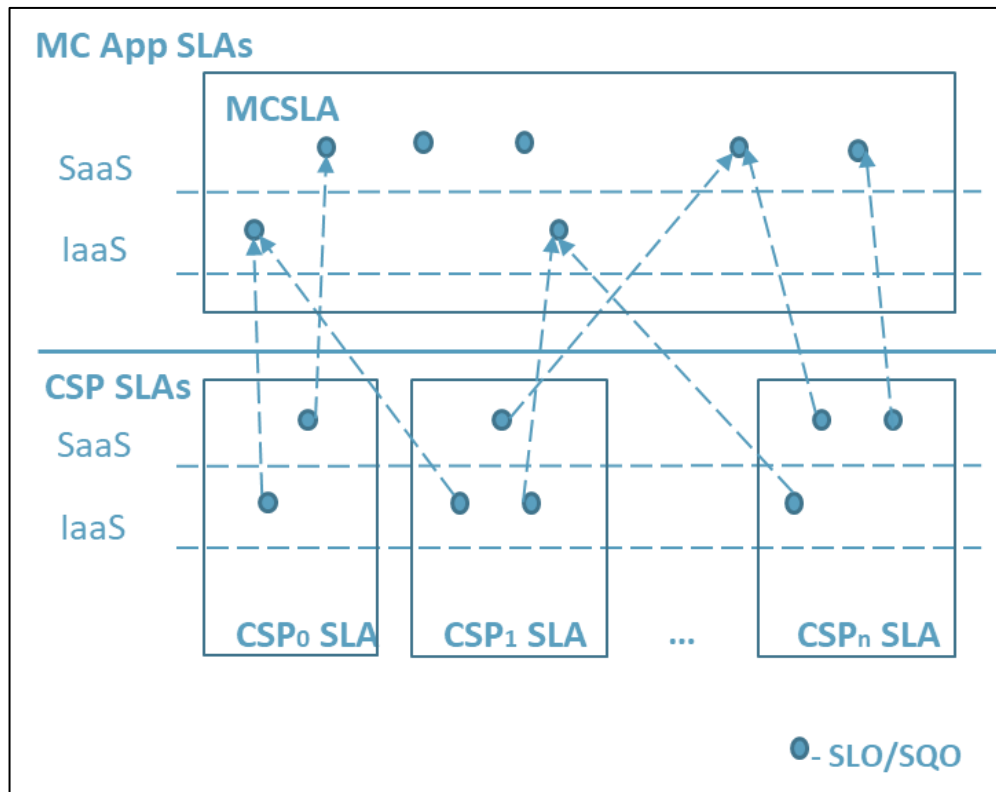
Concerning end-users, the developer may also define application specific terms. These additional terms pertain to the application, are consumer-oriented and not derived from the CSP SLAs. These can be terms the developer needs monitored and/or agreed with the end-user. An example would be the application's response time.

Furthermore, it is important for the MCSLA to reflect the diversity in the contracted SLAs on CSP level and the system hierarchy (IaaS, SaaS) – these need to be consolidated.

Moreover, a SLA term can be hard or soft. This is important for monitoring purposes. Hard terms are to be observed at all time, those declared as soft do not pose a major risk.

Another aspect concerning an MCSLA is re-deployment. Once an MCSLA has been set and communicated to the end-user, certain terms should not be changed. A solution would be to define two layers: an external one, which has to be respected and cannot be changed when a re-deployment should take place, and an "internal" one that collects the SLAs from the various providers where the

application has been or will be deployed. The external SLA is a composition of the SLAs in the internal part, plus the application SLOs. In case of a candidate redeployment involving different services or CSPs, the internal SLAs change accordingly, but their composition should still satisfy the external SLA for the candidate to be acceptable. If no such candidate exists, the adaptation (i.e. re-deployment) fails.



**Figure 1.** Conceptual Idea – Make up of an MCSLA

## 2.2 SLA Aggregation Patterns

In a multi-cloud deployment scenario, a minimum of two microservices are expected to be deployed on different CSPs or on different cloud services of the same CSP. In this case, there will be at least two SLAs contracted for the developer of the multi-cloud native application. As previously stated, these agreements might differ in their content but might include the same terms (SLOs) but with different values.

In order to reduce the complexity of managing a multitude of CSPs, SLAs Aggregation Patterns complemented with an aggregation engine are needed.

An SLA Aggregation Pattern [2] is a mathematical function that computes several terms into one aggregated term.

In [2], they introduce a type which extends the WS-Agreement [3] specification, which labels specific SLA terms with a type in order to be able to calculate and help automate the SLAs.

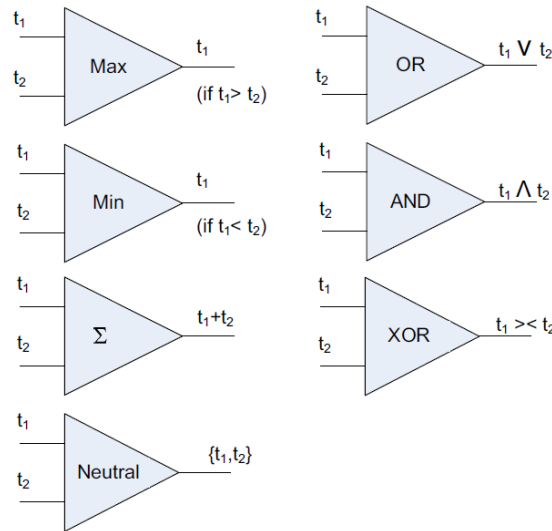
As the DECIDE project is not following the WS-Agreement specification but ISO/IEC 19086<sup>1</sup> (parts 1-4) [1, 4, 5, 6], it may result fruitful to check if such a type is also required there. In any case, these types

<sup>1</sup> Part 2 and 4 are still under development.

can also be internally depicted in ACSml and the MSLA Editor. This will be investigated in the coming iteration of this task.

In [2] seven Aggregation Patterns are defined as types<sup>2</sup>

$$Types = \{sumtype, maxtype, mintype, netural, ORtype, ANDtype, XORtype\}$$



**Figure 2.** Term composition using Aggregation Patterns [2]

Figure 2 depicts the aforementioned types, the following three types are relevant in DECIDE context:

The *sumtype* function (denoted as  $\Sigma$  in Figure 2) defined as

$$\begin{aligned} sumtype &\in Types (\Leftrightarrow sumtype : P(Terms) \rightarrow Terms) \\ sumtype(term1, \dots termn) &= \_n \text{ } i=1 \text{ } termi.terms.value \end{aligned}$$

can be used to calculate terms for storage space, memory, availability and cost in a deployment environment where all microservices are deployed on the same machine. Moreover, it assumes that all microservices will fail simultaneously, which is rarely the case. Therefore, it makes sense to extend this list with an additional type to fulfil DECIDE’s needs in a multi-cloud context.

The *minType* function defined as

$$\begin{aligned} mintype &\in Types (\Leftrightarrow mintype : P(Terms) \rightarrow Terms) \\ mintype(term1, \dots termn) &= \min_{i=1}^n termi.terms.value \end{aligned}$$

is an aggregation function that aggregates a number of terms into one term. The minimum of these terms is picked up and ultimately represents the aggregation of the input terms. Therefore, the only term having the minimum value will contribute to the final term in the MSCLA. A good example is given in [2], which is that for the bandwidth: “In a sequence of activities the activity pertaining to the minimum bandwidth will become the bottleneck for the whole sequence making other activities with higher bandwidth ineffective.” [2]

<sup>2</sup> For the full formalisation please see [2]

The *ORType* function defined as

$$\begin{aligned} ORType &\in Types(\Leftrightarrow ORType : P(Terms) \rightarrow P(Terms)) \\ ORType(term1, \dots termn) &= \_ni=1 termi.terms.value \end{aligned}$$

*ORType* is an aggregation function that aggregates a number of terms into one or more terms. It does so by applying a logical OR function on these terms and the result represents the aggregation of all the input terms. For instance, an application developer who wants to aggregate services of varying qualities but would also like to segregate them under different levels of SLAs, may use the *ORType* aggregation function to fulfil his needs.

An example could be a reseller who buys computational resources of different speeds and qualities from different vendors and aggregates them using *ORType* function so that later, he can offer SLAs of different levels such as gold, silver or bronze, etc. to its consumers. This might prove interesting for developers and will be investigated how it is optimally used in the project.

### 2.2.1 SLA Aggregation Patterns for Availability

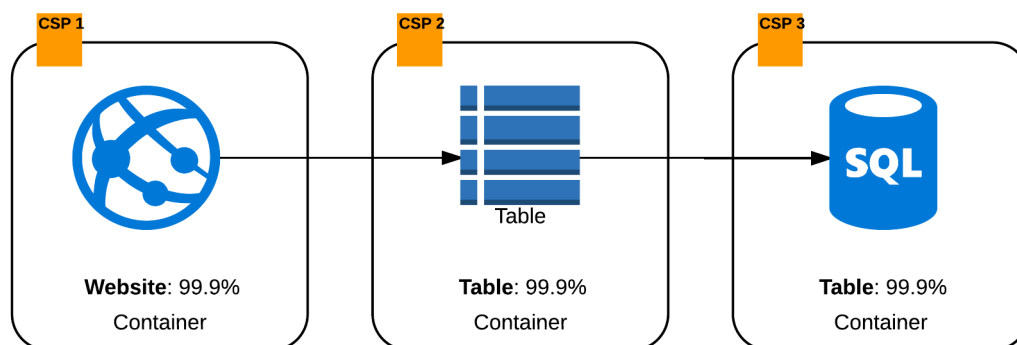
In this section, we look at how to use these Aggregation Patterns for the NFR availability. The following patterns also take into consideration the different deployment topologies. In the next iteration of this deliverable, more Aggregation Patterns will be presented.

Availability is probably the most important single metric that can be used to measure the performance of a service. It shows the time or percentage the service is operational and responding.

The following section gives examples using the three selected Aggregation Patterns for Availability.

#### Aggregated availability the sumType pattern in a basic multi-cloud environment

This example (see Figure 3) for availability includes a web site, a SQL database and table storage. The deployment has taken place on three different CSPs. For the application to function as intended, each of these components must be working. They also each have a 99.9% availability guaranteed in their SLA. It cannot be assumed that the components will fail simultaneously, but at different times. This means that the summation of all terms using the *sumType* function described above is insufficient and would yield a false value for Availability.



**Figure 3.** Basic multi-cloud deployment topology

Therefore, we find it necessary to extend the list of types presented above with the following function to be applied for this deployment topology:

$$mcAvailability(term_1, \dots, term_n) = 100\% - \left( \sum_{i=1}^n (100\% - term_i) \right)$$

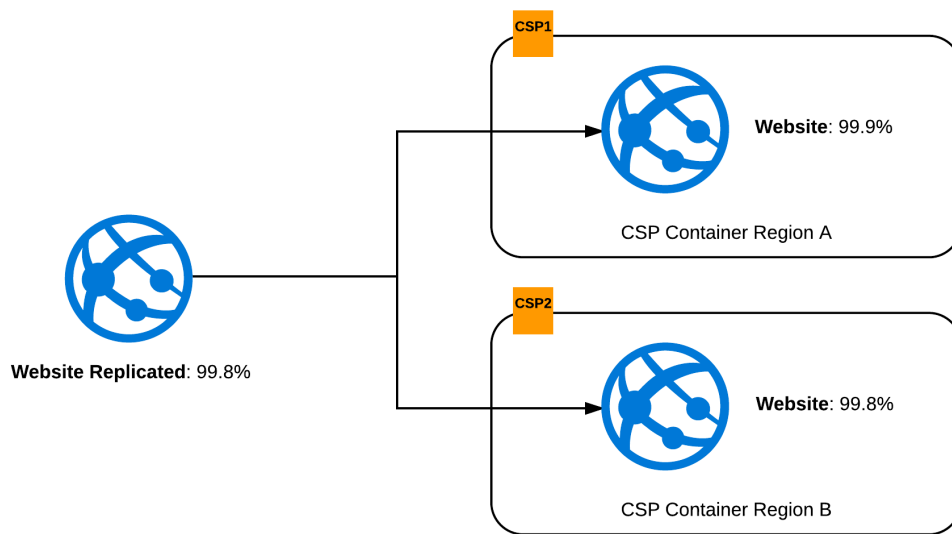
The function takes a number of terms and creates a sum of the “unavailability” of all terms and deducts it from the optimal value for availability.

Example result would be as follows:

$$mcAvailability(99,9\%, 99,9\%, 99,9\%) = 99,7\%$$

#### **Aggregated availability the MINtype pattern in replication deployment topology**

In this example for availability, we have a web site, which is replicated on one CSP in different Regions.



**Figure 4.** Multi-cloud replication deployment topology

In this example, it is only viable to select the minimum term value based on the deployment topology. Otherwise, the availability term in the MCSLA would be false as it cannot be guaranteed.

The function to be applied for this deployment topology is as follows:

$$mintype(term_i, \dots, term_n) = \min_{i=1}^n term_i.value$$

Example result would be as follows:

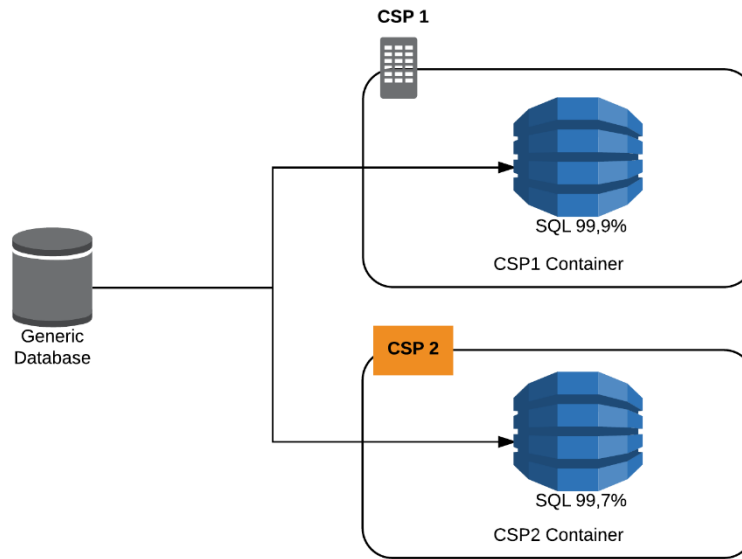
$$mintype(99,9\%, 99,8\%) = 99,8\%$$

#### **Aggregated SLAs uptime the ORtype pattern for service composition from different vendors**

In this example, a generic database accesses two different SQL servers. One enjoys a 99,9% availability and the other only 99,7% availability. The deployment as depicted in Figure 5 is across two different CSPs.

The developer may include, as described before, different plans for the consumers (e.g. bronze, silver, gold) and derive these from the different guaranteed quality for availability. Therefore, the developer

can choose which server is part of which plan by integrating the respective availability value in the MCSLA.



**Figure 5.** Different vendors topology

The function to be applied for this deployment topology is as follows:

$$ORtype(term_1, \dots, term_n) = \bigvee_{i=1}^n term_i.term_s.value$$

Example result would be as follows:

$$ORtype(99,7\%, 99,9\%) = 99,9\%$$

$$ORtype(99,7\%, 99,9\%) = 99,7\%$$

## 3 Implementation

### 3.1 Functional description

The MCSLA Editor provides a tool for the authoring of an MCSLA to be used as a contract between the end-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.

The main functionalities for the MCSLA Editor are as follows:

- F1. Provides supportive means for the developer for the definition of the composite MCSLAs and the corresponding SLOs of the application. This includes:
  - a. Aggregation of the available terms in the various contracted SLAs using defined mathematical formulas mapped to deployment topologies.
  - b. Allowing for editing an existing MCSLA after a re-deployment is recommended whilst respecting the initial SLA
- F2. Provide an interactive user interface for authoring an MCSLA
- F3. The MCSLA is translated into a standards-based machine-readable form that includes a metrics definition.
- F4. The MCSLA is translated into a human readable form.
- F5. Maintain an interface to ACSmI for accessing the contracted SLAs
- F6. Maintain access to the git repository of the Application.
- F7. Storage of the MCSLA file in a git repository to be accessed by the different DECIDE tools.
- F8. Integration of the MCSLA Editor in the DECIDE DevOps Framework.

The MCSLA Editor will be implemented incrementally. The first version (M12) includes the functionalities **F1 (partly)**, **F2**, **F3**, **F4 (partly)**, **F6**, and **F7**. The improvements and finalisation of the previous list of functionalities and the rest (F5 and F7) will be implemented in due course of the project, as the MCSLA Editor is very dependent on the implementation of ACSmI and the DevOps Framework. Moreover, the functionality will evolve during the course of the project. The next release will include more detailed elements also from a usability perspective.

**Table 1.** Functionality covered in M12 in relationship to the elicited requirements

Functionality	Req. ID	Coverage
F1	WP3-CSLA-REQ1	The MCSLA Editor provides the model and CRUD functionality for the file and the mechanism for storing and accessing the MCSLA. Aggregation rules will be integrated into the implementation in Year2.
F2	WP3-CSLA-REQ10, WP3-CSLA-REQ11	The MCSLA Editor is composed of a frontend and backend. The frontend is a web-based tool.
F3	WP3-CSLA-REQ6, WP3-CSLA-REQ7	The MCSLA definition is in machine-readable form and follows the standard ISO/IEC 19086-(1-4) [1] [4].
F5	WP3-CSLA-REQ1	A dummy API has been implemented to facilitate the dependency on ACSmI's interfaces. The interfaces required by the MCSLA Editor have been communicated to the relevant parties in the project. The implementation is based on these and holds a skeleton for accessing ACSmI's API.
F6	WP3-CSLA-REQ1	All the mechanisms for accessing the git repository are in place.

Functionality	Req. ID	Coverage
F7	WP3-CSLA-REQ1	All the mechanisms for storing the MCSLA in the defined application repository are in.

The following list compiles the implemented functionality in M12:

- The MCSLA editor has a web-based UI that allows the user to view all available SQOs and SLOs and select from these terms the ones relevant for the Application. This is done via Combo boxes and drag and drop functionality in the UI. The terms are based on a dummy API and dummy content.
- The MCSLA frontend UI gives information for the user in identifying, where the terms come from (which CSP).
- Any aggregated value of an SLA is removed entirely from the combo box and added to another box specifically dedicated to aggregated terms. The used formula to calculate the value is shown to the user for transparency reasons.
- The MCSLA Editor backend serves the UI and holds the model for the MCSLA, which is based on the ISO/IEC19086-2.
- The MCSLA Editor can read and write to git and can store the MCSLA file in the target application repository.

### 3.1.1 Fitting into overall DECIDE Architecture

The MCSLA editor is crucial for the DECIDE DevOps Framework as it is part of the continuous operation phase and lays the foundation for monitoring the multi-cloud native application as well as the CSPs, which may lead to imperative re-adaptation and re-deployment of the application.

Furthermore, it serves as an interface (UI) through which the developers specify the multi-cloud SLAs agreed with the end-users of the application. The MCSLA editor provides the developer with all possible SLOs and SQOs, which may partly incorporate default values, aggregated values or overwritten values depending on those resulting from the contracted CSPs. This resulting MCSLA serves as the contract between the developer and the end-users of the application.

The tool ADAPT is the main DECIDE tool that is dependent on the output of the MCSLA editor. But also, the MCSLA editor is dependent on the ACSmI as it provides the initial set of SLAs that have been contracted for a multi-cloud deployment scenario. When a re-deployment takes place another round of interactions between the MCSLA editor and ACSmI is required.

Please see **Figure 6**. Component diagram for MCSLA Figure 6 for the interactions between the MCSLA editor and the DECIDE tools.

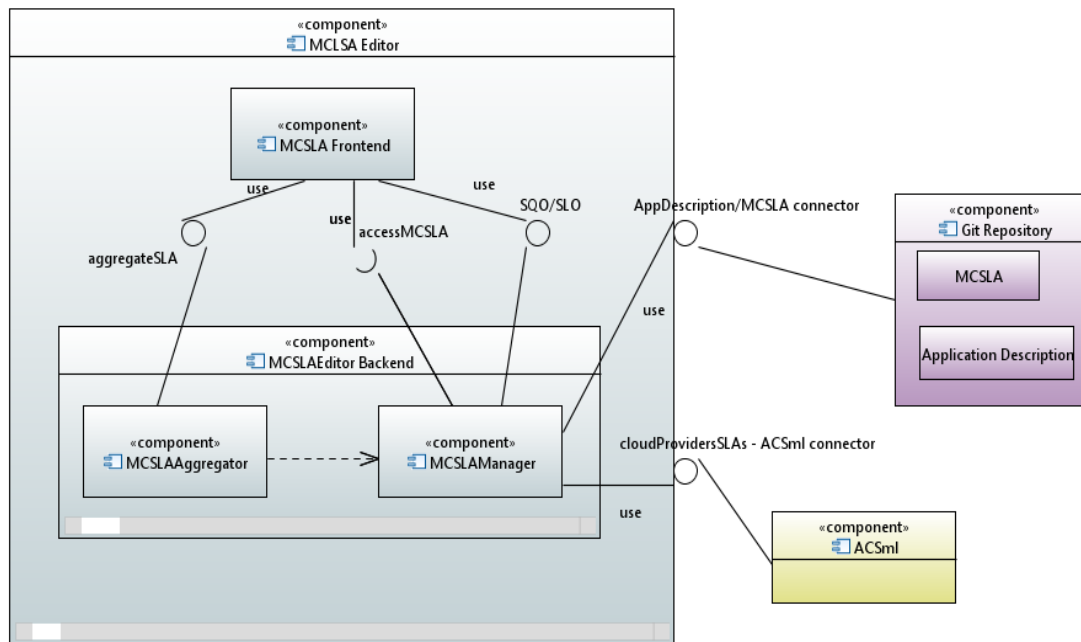
## 3.2 Technical description

This section describes the technical details of the implemented software for the current prototype of the MCSLA editor.

### 3.2.1 Prototype Architecture

The MCSLA Editor is a two-tier architecture represented by the MCSLA frontend and the backend consisting of the MCSLAAggregator and the MCSLAManager. Figure 6 depicts the component diagram for the MCSLA Editor.





**Figure 6.** Component diagram for MCSLA editor

### **MCSLA Frontend**

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

### **MCSLAManager**

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

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

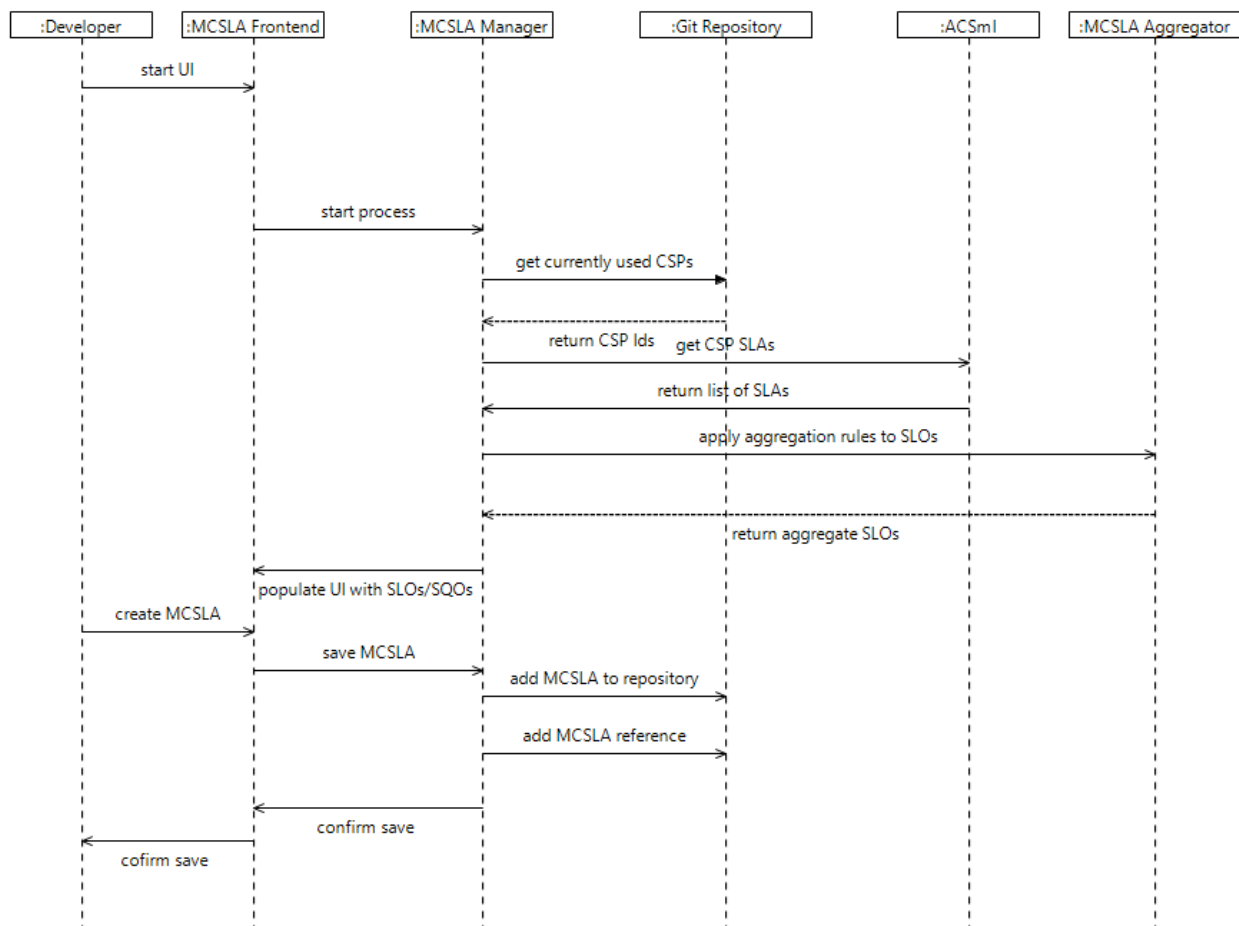
Furthermore, the MCSLA Manager is in charge of storing a tagged version of the MCSLA in the code repository for ADAPT to access and be able to monitor the application.

### **MCSLAAggregator**

The MCSLA Manager serves the MCSLA Aggregator with the SLAs in order 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.

The following sequence diagram (Figure 7) depicts the communication and message exchange that takes place between the MCSLA Editor components, external repositories and DECIDE tools (ACSml).



**Figure 7.** Sequence diagram for creating an MCSLA

The sequences for creating a MCSLA are as follows:

1. The developer starts the *MCSLA Frontend* (GUI); this process calls the *MCSLA Manager* in order to populate the frontend with the necessary values.
2. As long as the MCSLA editor as a whole is integrated into the dashboard, it is clear which *Application Description* is applicable at this stage. The *Application Description* residing in a repository will be accessed via the *MCSLA Manager* to retrieve the currently used deployment topology, i.e. the cloud service Ids.
3. With the cloud service Ids, the *MCSLA Manager* contacts *ACSmI* in order to obtain the contracted SLAs.
4. The *MCSLA Manager* then contacts the *MCSLA Aggregator* to take the necessary measures to aggregate the SLOs defined in each SLA.
5. Once this step is completed, the *MCSLA Manager* populates the frontend with the available SLO/SQOs and their possible values.

The developer then uses the GUI to create the MCSLA, which is in turn saved by the *MCSLA Manager* in the code repository as well as registering it in the *Application Description*.

## 3.2.2 Technical Specification

### 3.2.2.1 MCSLA Frontend

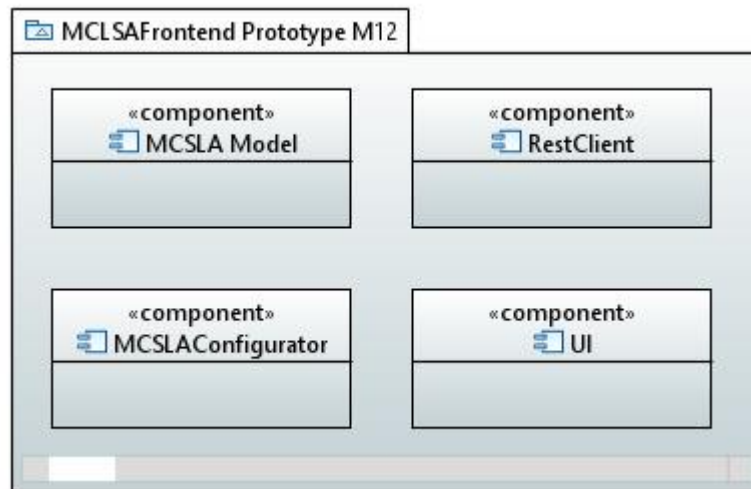


Figure 8. MCSLA Frontend prototype implementation M12

The MCSLA Frontend is implemented using the ZK Ajax Framework 8 [7]. The following are the components implemented for the frontend:

- **MCSLAModel:** holds the logical representation of the MCSLA definition. The MCSLA frontend shares the same data model as the backend to be able to interact with the services. The following MCSLA parts are defined as specified by ISO/IEC 19086-2 [5]: Metric, Expression, Parameter, Remedy, ViolationTriggerRule, UnderlyingMetricRef. The model also includes other parts that are DECIDE specific these are: CSP, MCSLA, MicroservicesSLA, MicroservicesSLO, and MicroservicesSQO. These data structure is described in detail in section 3.2.2.3.
- **MCSLAContriguartion:** The configuration holds the information regarding the DECIDE application project repository in order to access the MCSLAService in the correct context.
- **RESTClient:** is a helper class for accessing the RESTful MCSLAService that provides the MCLSA objects, CSP lists and SLA information.
- **UI:** holds the implementation of the user interface as a Single Page Application and implements all the logic behind CRUD operations in a graphical way as well as the interactive user elements.

### 3.2.2.2 MCSLA Editor Backend

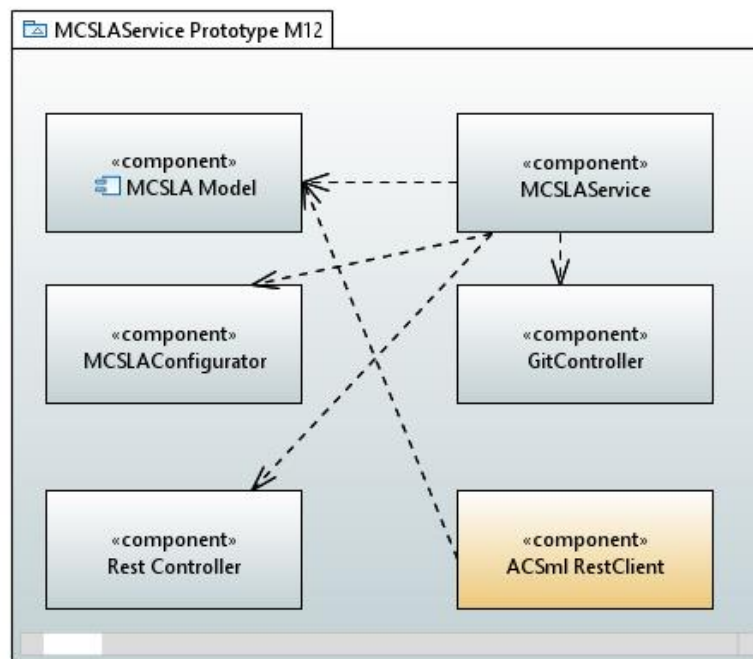


Figure 9. MCSLA Service backend prototype implementation M12

*MCSLService backend* consists of the *MCSLAManager* and *MCSLAAggregator*. The backend is implemented with Spring framework version 3 [8] with Jersey 2.25.1 [9] for supporting RESTful web services in Java. All web services return JSON as a format. The MCSLA backend is a dynamic project using maven, Apache tomcat 8 and JDK 8. The following components are implemented:

- **MCSLService:** is the main implementation of a RESTful interface that the backend provides. It also includes the implementation for the Aggregation Patterns. This component operates on an instance of the MCSLA from the git repository.
- **RESTController:** component defines the restful web services endpoint using Spring Framework.
- **MCSLAModel:** holds the logical representation for the MCSLA definition. following MCSLA parts are defined as specified by ISO/IEC 19086-2: Metric, Expression, Parameter, Remedy, ViolationTriggerRule, UnderlyingMetricRef. The model also includes other parts that are DECIDE specific these are: CSP, MCSLA, Microservices SLA, Microservices SLO, and Microservices SQO. These data structure is described in detail in section 3.2.2.3.
- **GitController:** manages all interactions with the git decide application repository.
- **MCSLAConfigurator:** implements the functionality to manage the decide application projects. It is supplied by properties such as repository URL, username and password.

The **ACSml RestClient** is for accessing the ACSml services. It will be implemented once the services are available from ACSml.

#### 3.2.2.2.1 REST Interfaces

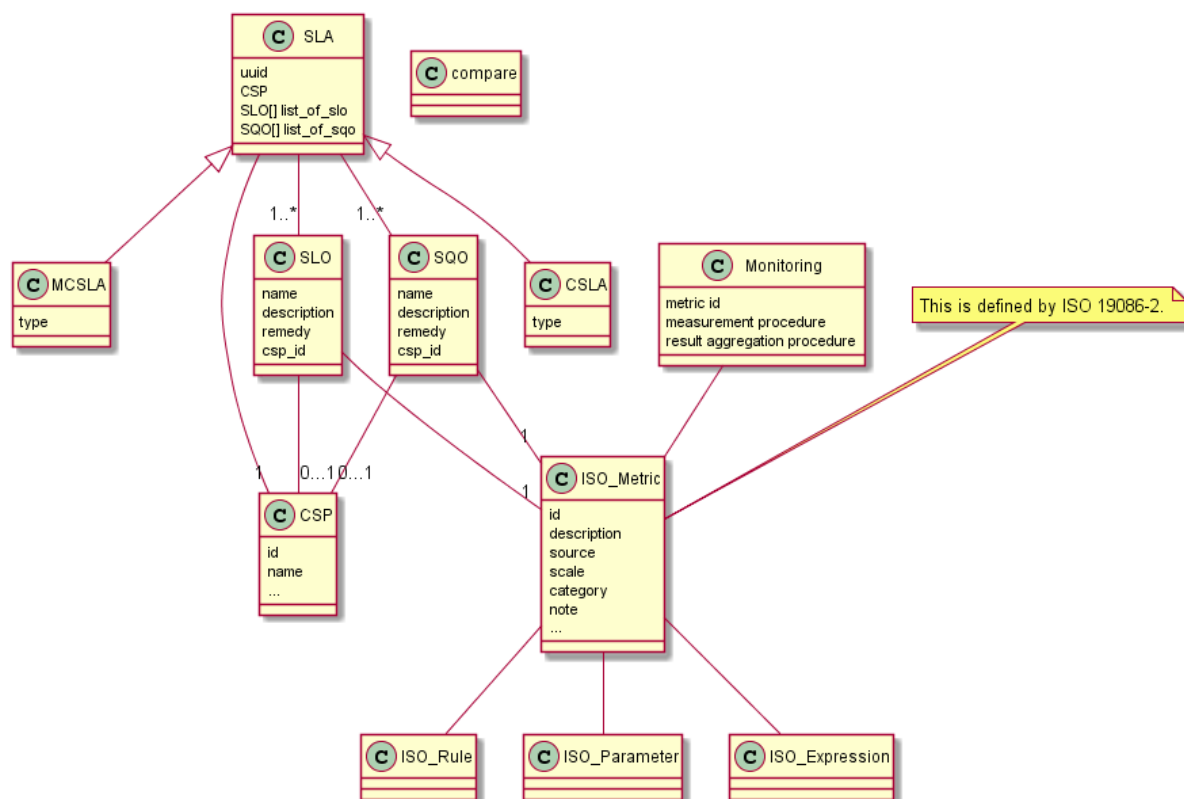
The backend provides the following operations described below in brief. Each operation produces and consumes JSON. The interface documentation will be generated using OPEN API version 3 and available online. The JSON structure can be viewed in section 3.2.2.3.1.

**Table 2.** Rest interfaces provided by the backend -v1

HTTP Verb	URL	Description
GET	/v1/mscla	returns the current MCSLA from the Application Repository as an application/json structure
GET	/v1/mcsclas	returns all available MCSLAs as an application/json structure
GET	/v1/mcsclas/{id}	returns an MCSLA with the given id as an application/json structure
POST	/v1/mscla	Creates a new MCSLA in the application description
PUT	/v1/mcsclas	updates existing MCSLA. Consumes an application/json structure as body
DELETE	/v1/mcsclas/{id}	deletes MCSLA with the supplied id. Expects an id
POST	/v1/aggregate	Method to aggregate the SLOs, takes a list of SLOs as an application/json structure
GET	/v1/config	returns the configuration of the project

### 3.2.2.3 MCSLA Data Model

The data model for an MCSLA is depicted below in Figure 10 and serves as a reference



**Figure 10.** MCSLA Data Model

The following tables describe the MCSLA model for monitoring with a brief description for each field. Table 3 describes the nested elements for the MCSLA. The MCSLA editor is responsible for eliciting this information from the user.

**Table 3.** Application description model for monitoring the application via its MCSLA (nested elements for “app\_mcsla”)

Element Name	app_mcsla		
Description	General information about the MCSLA		
attribute -or- Element	Type	Multiplicity / Default	Definition
id	String	1	Unique Identifier for the MCSLA
description	String	1	This is MCSLA description line.
visibility	String	1	public or private
validityPeriod	Integer	1	The validity period of the MCSLA in days
microservice_SLAs	Microservice_SLAs	1..*	The list of SLAs for each microservice

The following Table 4 describes the fields nested in the microservice\_SLAs field of the MCSLA.

**Table 4.** Nested elements for microservice\_SLAs

Element Name	Microservice_SLAs		
Description	The general information about the SLAs for each microservice		
attribute -or- Element	Type	Multiplicity / Default	Definition
id	String	1	Unique Identifier for the microservice_SLAs
ms_id	String	1	Unique Identifier of the microservice this SLA belongs to
csp_id	String	1	Unique Identifier of the CSP from which the SLA comes from
visibility	String	1	public or private
validityPeriod	Integer	1	The validity period of the SLA in days, should not be higher than that of the MCSLA
microservice_SLO	microservice_SLO	1..*	List of microservice SLOs
microservice_SQO	microservice_SQO	1..*	List of microservice SQOs

The following Table 5 describes the fields nested in the microservice\_SLO and microservice\_SQO fields of microservice\_SLAs.

**Table 5.** Nested elements for microservice\_SLO and microservice\_SQO

Element Name	microservice_SLO and microservice_SQO		
Description	The general information about the slo or sqo defined for a microservice		
attribute -or- Element	Type	Multiplicity / Default	Definition
id	String	1	Unique Identifier for the microservice_SLAs
termName	String	1	Name of the term to which it refers to

<b>value</b>	Integer	1	Term value that should not be violated based on calculation formula
<b>unit</b>	String	1	Term unit
<b>comparisonOperator</b>	String	1	Comparison operator for monitoring the SLO
<b>violationTriggerRule</b>	ViolationTriggerRule	1	The violation Trigger Rule
<b>remedy</b>	Remedy	0..1	The compensation available to the cloud service customer in the event the cloud service provider fails to meet a specified cloud service level objective
<b>metrics</b>	Metrics	1..*	The definition of how to measure the SLO or SLA
<b>violation_report</b>	String	0..1	Indicates where to report violations for this application (optional)

The following Table 6 describes the fields nested in the violationTriggerRule field of microservice\_SLO and microservice\_SQO.

**Table 6.** Nested elements for ViolationTriggerRule

Element Name	ViolationTriggerRule		
<b>Description</b>	The general information about the violation trigger rule		
attribute -or- Element	Type	Multiplicity / Default	Definition
<b>interval</b>	string	1	Indicates the monitoring frequency for each SLO
<b>breaches_count</b>	Integer	1	The count of how many breaches have taken place

The following Table 7 describes the fields nested in the remedy field of microservice\_SLO and microservice\_SQO.

**Table 7.** Nested elements for Remedy

Element Name	Remedy		
<b>Description</b>	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
<b>type</b>	String	1	The type of remedy the cloud service provider will be offering the cloud service customer
<b>value</b>	Integer	1	The value of the type of remedy offered by the cloud service provider
<b>Unit</b>	String	1	The unit for the value offered
<b>validity</b>	Integer	1	The validity period for this remedy

The following table holds the fields (taken directly from ISO/IEC 19086-2 Metric Model [5]) 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	Metric		
Description	The general information about the metric		
attribute -or- Element	Type	Multipli city / Default	Definition
<b>descriptor</b>	String	0..1	A short description of the metric
<b>Id</b>	String	1	A unique identifier for the metric within a context
<b>source</b>	String	1	The individual or organization who created the metric
<b>scale</b>	enumera tedList	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.
<b>note</b>	String	0..1	Additional information about the metric and how to use it.
<b>category</b>	String	0..1	A grouping of metrics with similar expressions, rules, and parameters
<b>expression</b>	Expressio n	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 underlying metrics, parameters, and rules.
<b>parameters</b>	Paramet er	0..*	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.
<b>rules</b>	Rule	0..*	A rule is used to constrain a metric and indicate possible method(s) for measurement.
<b>underlyingM etrics</b>	Metric	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 9 describes the fields nested in the expression field of a Metric.



**Table 9.** Nested elements for Expression

Element Name	Expression		
Description	The expression of the calculation of the <b>Metric</b> and supporting information		
attribute Element	-or- Type	Multiplicity / Default	Definition
<b>Id</b>	String	1	A unique identifier (within the context of the metric) for the expression
<b>expression</b>	String	1	The expression statement written using the ids to represent underlying metrics, parameters, and rules.
<b>expressionLanguage</b>	String	1	The language used to express the metric (i.e. ISO80000 [10])
<b>note</b>	String	0..1	Additional information about the expression
<b>unit</b>	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.
<b>subExpression</b>	Expression	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 10 describes the fields nested in the parameters field of a metric.

**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 Element	-or- Type	Multiplicity / Default	Definition
<b>id</b>	String	1	The unique identifier of the parameter
<b>parameterStatement</b>	String	1	The statement or value of the parameter
<b>unit</b>	String	1	The unit of the parameter
<b>note</b>	String	0..1	Additional information about the parameter

The following Table 11 describes the fields nested in the rules field of a Metric.

**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 Element	-or- 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

#### 3.2.2.3.1 MCSLA JSON Example

MCSLA backend uses Gson Java library that can be used to convert Java Objects into their JSON representation. It can also be used to convert a JSON string to an equivalent Java object. Gson is an open-source project hosted at <http://code.google.com/p/google-gson>.

MCSLA Repo keeps all created MCSLA instances as object and persist these in JSON Format. Figure 11 shows one example of MCSLA in JSON format.

```
{
  mCSLA_id: "915f3e3e-146c-4c34-94c8-3d74781ealcd",
  description: "This is MCSLA dummy description line created from createMCSLAexample method at:2017.11.08 16.22",
  visibility: "public",
  validityPeriod: "this is Validity Period line.",
  microservice_SLAs: [
    {
      microservice_SLA_id: "4d684160-3676-4d1c-ab94-a9cb074badd0",
      __comment: null,
      csp_id: "4561316a-a3b4-4595-8ddd-5768baa2ed6d",
      csp_name: null,
      microservice_id: "fc9bb8d4-9909-4216-822e-24230eac40dd",
      visibility: "public",
      validityPeriod: "5",
      microservice_SQOs: [...],
      microservice_SLOs: [
        {
          microservice_SLO_id: "bf449922-ba74-45bb-95bc-8bb205d9613a",
          __comment: "CPU Utilization will not be less then 95% of the time Over a Day",
          termName: "CPU-utilization-SLO",
          monitoringResult: "result",
          value: "95",
          unit: "percentage",
          calculationFormula: "CPU-utilization LT 95",
          violationTriggerRule: [
            {
              breaches_count: 2,
              viointerval: 30
            }
          ],
          remedy: {
            type: "discount",
            value: 5,
            unit: "%",
            validity: "P1D"
          },
          csp_id: "ca997fa3-09cd-4a85-8cd8-3a71cafe74a9",
          csp_name: "Amazon",
          metrics: [
            {
              metric_id: "CSA_002",
              descriptor: "CloudServiceAvailability",
              scale: null,
              parameter: null,
              expression: null,
              underlyingMetricRef: {
                refid: "TQD_001"
              }
            }
          ]
        }
      ]
    }
  ]
}
```

Figure 11. MCSLA example in JSON format

## 4 Delivery and usage

### 4.1 Package information

The MCSLA Frontend consists of the following packages:

- eu.DECIDEh2020.mcsla.beans contains the implementation Classes for model for the MCSLA definition
- eu.DECIDEh2020.mcsla.editor.service contains the Interfaces and Implementation for parsing the configuration information.
- eu.DECIDEh2020.mcsla.editor.dual\_listbox contains the class DualListbox which implements the UI as a Single Page Application
- eu.DECIDEh2020.mcsla.editor.data contains the class SlasData, a helper class to use the jersey client to build the list of SLOs, SQOs and build the aggregated SLO.
- eu.DECIDEh2020.mcsla.configuration contains the classes for setting the configuration of the decide project.

The MCSLAService Backend consists of the following packages:

- eu.DECIDEh2020.mcsla.beans contains the implementation Classes for model for the MCSLA definition
- eu.DECIDEh2020.mcsla.service.service contains the main implementation of RESTful interface the class MCSLAService. This class operates on instance to the MCSLA Repository.
- eu.DECIDEh2020.mcsla.service.controller contains the class MCSLAController which implements the CRUD functionality for an MCSLA
- eu.DECIDEh2020.mcsla.service.git contains the class GitControl for accessing the git decide application repository
- eu.DECIDEh2020.mcsla.service.configuration contains the classes for setting the configuration of the decide project

### 4.2 Configuration and Installation instructions

#### 4.2.1 MCSLA Editor (Frontend)

The MCSLA editor is also a dynamic project developed using the JAVA language and uses maven, Apache tomcat 8 and JDK 1.8 in eclipse.

##### 4.2.1.1 MCSLA Editor UI configuration

To use the REST web services of the MCSLAService backend you have to configure the UI first. Edit the file *decide.mcsla.editor\src\main\resources\application.properties* and put the main URL to the backend. Or you can provide this information in your user environment. The user environment has highest *priority*.

```
decide.mcsla.srv=http://localhost:8080/decide.mscla.srv
```

##### 4.2.1.2 Build the UI

The project is available via Git repository. Download the source code from here: [https://git.code.tecnalia.com/DECIDE\\_Public/DECIDE\\_Components/tree/M12/DevOpsFramework/MCSLAEditor](https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components/tree/M12/DevOpsFramework/MCSLAEditor).

The project uses Maven as the build tool. So, the only thing to do is to call

```
$> mvn clean install
```

The pom.xml file contains all the required dependencies that the application needs. In order to build the WAR file. You will find the WAR in the target directory.

#### 4.2.1.3 Install and Run

Finally, deploy this application from target folder to a tomcat server and run it.

Once the application is deployed go to <http://localhost:8080/decide.mcsla.editor/>. When you run the application, you will get the following output as depicted in Figure 12. MCSLA Editor UI in action

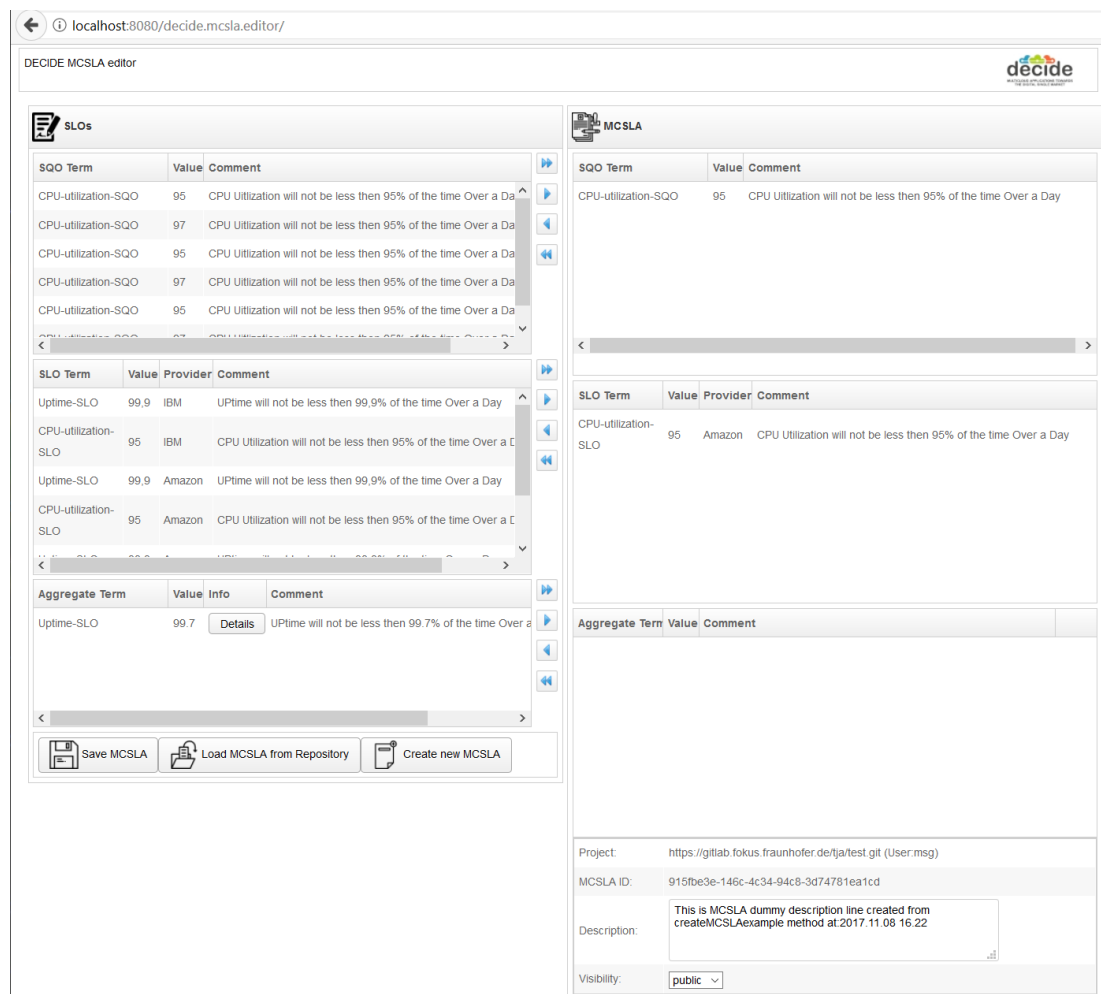


Figure 12. MCSLA Editor UI in action

## 4.2.2 MCSLService (Backend)

### 4.2.2.1 MCSLService Configuration

The current implementation of the MCSLService (backend) needs to set a couple of configuration elements. These can be set in the user environment in the file `decide.mcsla.editor\src\main\resources\application.properties`. The user environment has highest *priority*. In subsequent versions of the project the configuration of the MCSLService will be moved to the DevOps dashboard. The information needed is as follows:

The service needs the git remote path, user and password to your project. Where your Application Description DECIDE.json is located.

```
Logging.level.com.concretepage=INFO
App.git.remote-path=https://gitlab/user/decide.git
App.git.user-name=decideuser
App-git.user-password=decidepassword
```

#### 4.2.2.2 Build the MCSLService

The project is available via Git repository. Download the source code from [https://git.code.tecnalia.com/DECIDE\\_Public/DECIDE\\_Components/tree/M12/DevOpsFramework/MCSLService](https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components/tree/M12/DevOpsFramework/MCSLService).

The project uses Maven as the build tool. So, the only thing to do is to call

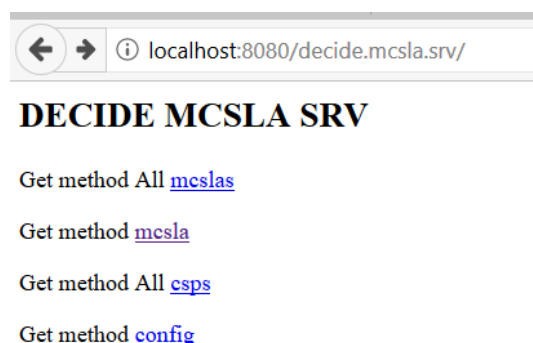
```
$> mvn clean install
```

The pom.xml file contains all the required dependencies that the application needs. In order to build the WAR file. You will find the WAR in the target directory.

#### 4.2.2.3 Install and Run

Finally, deploy this application from target folder to a tomcat server and run it.

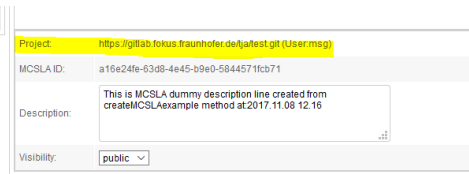
Once the application is deployed go to <http://localhost:8080/decide.mcsla.srv/>. When you run the application, you will get the following output. These are GET methods that you can call within your browser without any REST client. Just click the methods.



**Figure 13.** MCSLA backend in action

### 4.3 User Manual

1. Create new MCSLA will load SLAs pertaining to current deployment configuration and provide an empty MCSLA skeleton.
2. In the UI, the users can see the path to the project repository and the user name. This path can be changed in the MCSLA backend configuration (See Section 4.2.2.1).



Project: <https://gitlab.fokus.fraunhofer.de/test.git> (User: msg)

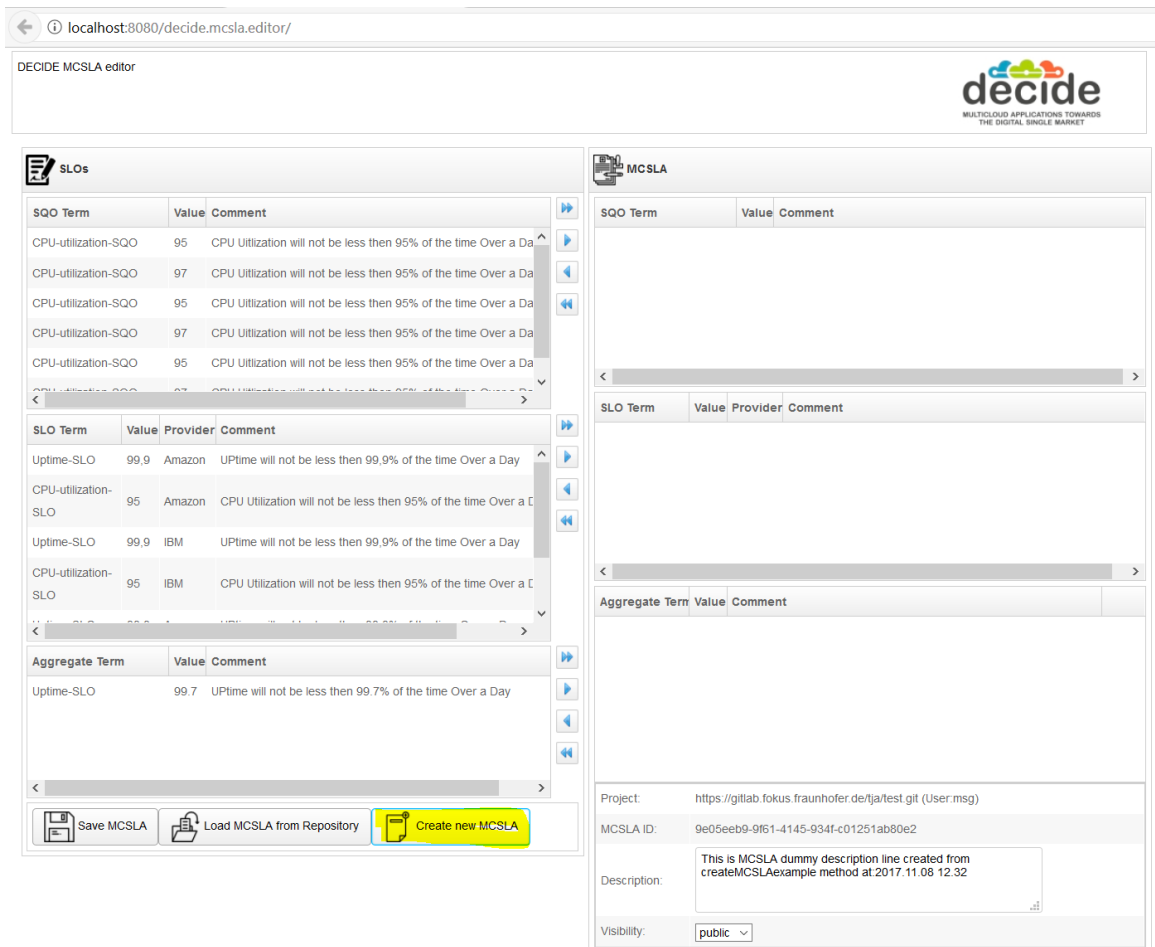
MCSLA ID: a16e24fe-63d8-4e45-b9e0-5844571fcb71

Description: This is MCSLA dummy description line created from createMCSLAexample method at 2017.11.08 12:16

Visibility: public

**Figure 14.** Path to the project repository and the user name

- Aggregate SLO/SQO box is populated with the aggregated values, those that are not aggregated are found in the SQO/SLO boxes with a source field/value. SQO/SLO boxes are controlled lists and hold all possible/supported SQO/SLOs. SLO/SQOs defined already in a CSP's SLA that is part of the deployment config and are not aggregates override their counterparts in the controlled list.



DECIDE MCSLA editor

localhost:8080/decide.mcsla.editor/

**SLOs**

SQO Term	Value	Comment
CPU-utilization-SQO	95	CPU Utilization will not be less than 95% of the time Over a Day
CPU-utilization-SQO	97	CPU Utilization will not be less than 95% of the time Over a Day
CPU-utilization-SQO	95	CPU Utilization will not be less than 95% of the time Over a Day
CPU-utilization-SQO	97	CPU Utilization will not be less than 95% of the time Over a Day
CPU-utilization-SQO	95	CPU Utilization will not be less than 95% of the time Over a Day

SLO Term	Value	Provider	Comment
Uptime-SLO	99.9	Amazon	Uptime will not be less than 99.9% of the time Over a Day
CPU-utilization-SLO	95	Amazon	CPU Utilization will not be less than 95% of the time Over a Day
Uptime-SLO	99.9	IBM	Uptime will not be less than 99.9% of the time Over a Day
CPU-utilization-SLO	95	IBM	CPU Utilization will not be less than 95% of the time Over a Day

Aggregate Term	Value	Comment
Uptime-SLO	99.7	Uptime will not be less than 99.7% of the time Over a Day

Save MCSLA Load MCSLA from Repository Create new MCSLA

**MCSLA**

SQO Term Value Comment

SLO Term Value Provider Comment

Aggregate Term Value Comment

Project: <https://gitlab.fokus.fraunhofer.de/test.git> (User: msg)

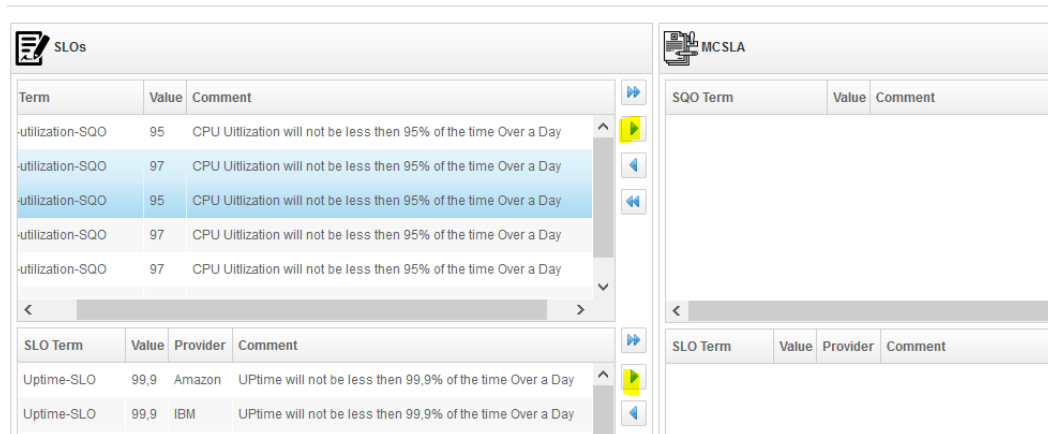
MCSLA ID: 9e05eeb9-9f61-4145-934f-c01251ab80e2

Description: This is MCSLA dummy description line created from createMCSLAexample method at 2017.11.08 12:32

Visibility: public

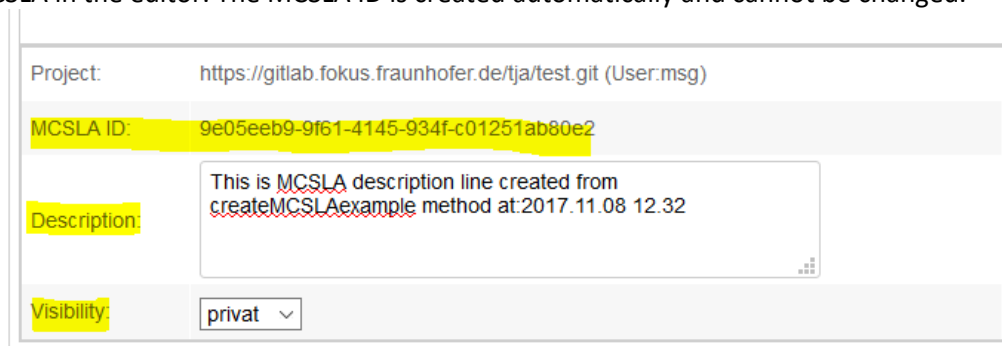
**Figure 15.** UI Create new MCSLA

- The user selects the relevant SLO/SQO /AG from left side and puts these in the list of MCSLA SLO/SQO/AG at the right side.



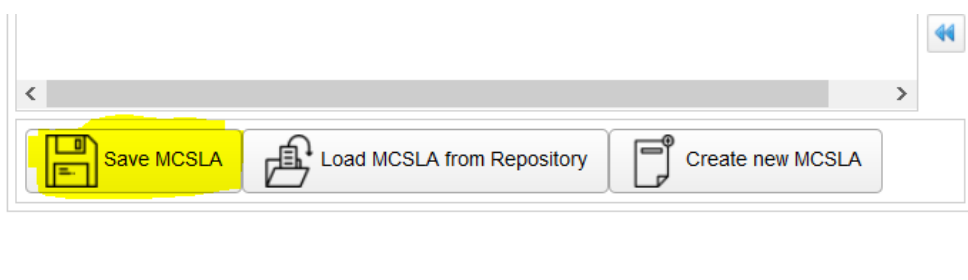
**Figure 16.** Select SLO/SQO/AG and put in the MCSLA

- The user can edit also the description of MCSLA or change the visibility or other properties of MCSLA in the editor. The MCSLA ID is created automatically and cannot be changed.



**Figure 17.** Edit MCSLA properties

- Last step is to save the MCSLA in Application Description. Use “Save” button.



**Figure 18.** Button to Save MCSLA

The end result is a JSON structure integrated in the Application Description file “*DECIDE.json*” in your project repository under attribute app\_mcsla, see Figure 11 for an example.

## 4.4 Licensing information

The source code is licensed under the Eclipse Public License version 2.0.

See <https://www.eclipse.org/org/documents/epl-2.0/EPL-2.0.html>



## 4.5 Download

The source code is available in the EC portal for deliverables, included in the zip file for D3.13.

The first release is available in the DECIDE open git repository, more precisely at the following address:

[https://git.code.tecnalia.com/DECIDE\\_Public/DECIDE\\_Components/tree/M12/DevOpsFramework](https://git.code.tecnalia.com/DECIDE_Public/DECIDE_Components/tree/M12/DevOpsFramework)

## 5 Conclusions

This document presented the MCSLA task and the outcome of several discussions and research. The first outcome has been presented in Section 2 of the document. The main points relevant for DEDICE MCSLA definition are, among others:

- An MCSLA involves different SLOs and SQOs that can be declared as soft or hard and that maintain an unchangeable external and changeable internal structure. The former must be respected during a re-adaption and re-deployment of the application.
- In multi-cloud deployment scenarios SLAs must be aggregated, removing the complexity of managing a multitude of SLAs from different CSPs
- Aggregation patterns are required.

A selection of aggregation patterns have been presented along with the proposition of a custom aggregation pattern that fulfils our needs in terms of aggregating the availability of an application dispersed across several CSPs or cloud services of different CSPs. An important aspect of this pattern is that it takes into account that the dispersed microservice will most probably not fail simultaneously, resulting in a lower availability value than that of an individual microservice.

Furthermore, the functional and technical description of the prototype is detailed. The prototype consists of two main blocks, namely, the frontend and the backend. These components communicate with one another using a restful interface and have been designed to be easily integrated into the DevOps Framework.

The Data Model for the MCSLA has also been presented, it is based on the ISO/IEC 19086 [1] [5] [4] [6] and includes a metric definition for each SLO in order to enable monitoring. An example JSON for an MCSLA has also been given.

Finally, all information related to building, installing and using the prototype has been described in section 4 of this document.

### 5.1 Future work

There is an important part of implementation work that will be included in the next iterations of the prototype. The following is an excerpt of the open issues:

- Integration with ACSml in order to replace the dummy API implementation with that from ACSml
- Integration with the DevOps Framework in order to have a holistic view on all tools and allow the MCSLA tool to be configurable via the UI.
- Improvements to the UI
- Providing the MCSLA in a human readable form.
- Investigation of more aggregation patterns for other NFRs, such as scalability.

Furthermore, regarding the conceptual work for the MCSLA task, the following needs to be investigated in the future:

- Hierarchical structures of SLAs due to sub-contracting and how that affects our implementation.
- Consideration regarding developing an implementation of the ISO/IEC 19086 separate from the tools, as a library to be integrated in different projects
- Inclusion of the types defined in section 2 into the ACSml or MCSLA Editor.

## References

- [1] International Standards Organisation, “ISO/IEC 19086-1: Information technology -- Cloud computing -- Service level agreement (SLA) framework -- Part 1: Overview and Concepts,” 2016.
- [2] I. Ul Haq and E. Schikuta, “Aggregation Patterns of Service Level Agreements,” FIT '10 8th International Conference on Frontiers of Information Technology, Islamabad, Pakistan, 2010.
- [3] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke and M. Xu, “Web Services Agreement Specification (WS-Agreement),” Open Grid Forum.
- [4] International Standards Organisation, “ISO/IEC 19086-3: Information technology -- Cloud computing -- Service level agreement (SLA) framework -- Part 3: Core conformance requirements,” 2017.
- [5] International Standards Organisation, “ISO/IEC 19086-2:Information technology -- Cloud computing -- Service level agreement (SLA) framework -- Part 2: Metric model,” 2017.
- [6] International Standards Organisation, “ISO/IEC 19086-4: Information technology -- Cloud computing -- Service level agreement (SLA) framework -- Part 4: Security and privacy,” 2017.
- [7] “ZK Framework,” [Online]. Available: <https://www.zkoss.org/product/zk/zk8>. [Accessed 26 November 2017].
- [8] “Spring Framework,” [Online]. Available: <https://projects.spring.io/spring-framework/>. [Accessed 26 November 2017].
- [9] “Jersey - RESTful Web Services in Java,” [Online]. Available: <https://jersey.github.io/>. [Accessed 26 November 2017].
- [10] International Standards Organisation, “Standards catalogue - ISO/TC2 - Quantities and units,” [Online]. Available: <https://www.iso.org/committee/46202/x/catalogue/>. [Accessed 26 11 2017].