ADEPTNESS – Design-Operation Continuum Methods for Testing and Deployment under Unforeseen Conditions for Cyber-Physical Systems of Systems

EUROPEAN COMMISSION

Horizon 2020

H2020-ICT-01-2019

GA No. 871319

Deliverable No.	ADEPTNESS D1.1 (ANNEX A)	
Deliverable Title	ANNEX A Requirements and validation tests	
Deliverable Date	2020-08-31	
Deliverable Type	Report	
Dissemination level	Public	
Written by	IKERLAN	2020-07-22
Checked by	IKERLAN	2020-07-22
Approved by	General Assembly	2020-08-31
Status	First version	2020-09-07

H2020-ICT-01-2019 – 871319 – ADEPTNESS: Design-Operation Continuum Methods for Testing and Deployment under Unforeseen Conditions for Cyber-Physical Systems of Systems

Acknowledgement

The author(s) would like to thank the partners involved with the project for their valuable comments on previous drafts and for performing the review.

Project partners

1 – MGEP – Mondragon Goi Eskola Politeknikoa – ES

2 – ORO – Orona S. Coop – ES

3 – UES – Ulma Embedded Solutions S. Coop – ES

4 – SRL – Simula Research Laboratory – NO

5 – BT – Bombardier Transportation Sweden – SE

6 – IKL – Ikerlan S. Coop – ES

7 – EGM – Easy Global Market SAS – FR

8 – MDH – Maelardalens Hoegskola – SE

9 – TUW – Technische Universitaet Wien – AT

Disclaimer:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871319.

Document Information

Additional author(s) and contributing partners

Name	Organisation
Goiuria Sagardui, Aitor Arrieta, Gorka Olalde	MGEP
Aitor Agirre, Blanca Kremer	IKL
Haris Isakovic	TUW
Franck Le Gall	EGM
Shaukat Ali, Asma Swappa	SRL
Wasif Afzal	MDH
Odei Astuday, Rakel Alvarez	UES

Document Change Log

Name	Date	Comments
V0.1	2020-05-20	Initial draft
V1.0	2020-09-07	First version

Exploitable results

Exploitable results	Organisation(s) that can exploit the result
There are not exploitable results in this document.	

CONTENTS

1 PURPOSE OF THE DOCUMENT

This document contains the requirements of the Adeptness toolchain and methods, alongside with their validation tests.

2 ADEPTNESS FRAMEWORK REQUIREMENTS

2.1 Stakeholder Requirements

2.1.1 Bombardier

The purpose of this section is to elicit Bombardier's requirements regarding ADEPTNESS toolchain.

2.1.1.1 Scope of the system

TCMS (Train Control and Management System) consists of TCMS Devices and the TCMS Application Software. The scope of the system is limited to TCMS Application Software layer.

2.1.1.2 BT Stakeholder Requirements

2.1.1.2.1 General

ID	Requirement Type	Description	Rule
SH_BT_1	Functional	The adeptness toolchain shall enable execution of both similar and different test cases in different levels of simulation/validation (MiL, SiL, HiL).	Mandatory
SH_BT_2	Functional	The adeptness toolchain shall enable mechanisms for oracle calculations to decide if test cases are successful or not in different levels of simulation/validation (MiL, SiL, HiL).	Mandatory
SH_BT_3	Functional	The adeptness toolchain shall allow for automatic generation of unforeseen situations to validate the system in different levels of simulation/validation (MiL, SiL, HiL).	Mandatory
SH_BT_4	Functional	The adeptness toolchain shall make the outcome of test executions traceable to relevant design or requirements artefact.	Optional

2.1.1.2.2 Model, Software and Hardware In-The-Loop

2.1.1.2.2.1 Monitoring

ID	Requirements Type	Description	Rule
SH_BT_MON_1	Functional	Monitoring data from MiL/SiL/HiL test executions shall be available through logs	Mandatory

2.1.1.2.2.2 Deployment Environment

ID	Requirements Type	Description	Rule
SH_BT_DEP_1	Functional	The simulator at MiL/SiL/HiL shall be the domain-specific simulator in use at the in-house project	Mandatory
SH_BT_DEP_2	Functional	The environment setup of MiL/SiL/HiL shall follow the project-specific configuration: I/O list etc.	Mandatory
SH_BT_DEP_3	Functional	The input to the test cases at the functional level shall be the stimuli triggering the execution of a defined functionality	Mandatory
SH_BT_DEP_4	Functional	The oracles shall be able to validate requirements related to specific TCMS functionality and non-functional requirements	Mandatory
SH_BT_DEP_5	Functional	The oracles shall be activated by an ordered and timed sequence of test input/stimulus	Mandatory
SH_BT_DEP_6	Functional	The results of test executions shall be provided to the validation service	Mandatory

SH_BT_DEP_7	Functional	The result of test execution shall follow the in-house taxonomy (pass/fail/invalid etc.)	Mandatory
SH_BT_DEP_8	Non-Functional	The TCMS down time shall be as close to 0 sec as possible	Mandatory
SH_BT_DEP_9	Functional	The deployment service shall provide support for Windows OS	Mandatory
SH_BT_DEP_10	Functional	The deployment service shall provide support for specific PLC hardware requirements at MiL/SiL/HiL	Mandatory
SH_BT_DEP_11	Functional	The deployment service shall have access to install and inspect the binaries and sources for deployment at MiL/SiL/HiL	Mandatory
SH_BT_DEP_12	Functional	The deployment service shall trigger validation of successful deployment at MiL/SiL/HiL	Mandatory
SH_BT_DEP_13	Functional	The deployment service shall meet the pre-conditions for deployment at MiL/SiL/HiL	Mandatory
SH_BT_DEP_14	Functional	The deployment service shall have rollback capability	Mandatory
SH_BT_DEP_15	Functional	The deployment service shall provide for setting up HiL simulation using cloud-based solution	Mandatory

2.1.1.2.2.3 Deployment Plan

ID	Requirement Type	Description	Rule
SH_BT_DEP_16	Functional	A test plan shall specify the simulation environments for the specific in-house project	Mandatory
SH_BT_DEP_17	Functional	A deployment plan shall specify the environment for each simulation level and a monitoring plan shall describe mechanisms for data monitoring from test logs	Mandatory

2.1.1.2.2.4 Monitoring Tool and Deployment Requirements

ID	Requirement Type	Description	Rule
SH_BT_DEP_18	Functional	An automation server shall be used to automate the deployment, monitoring and validation plans	Mandatory
SH_BT_DEP_19	Functional	The repository shall be integrated with requirement management tool (e.g. DOORS) and shall be able to connect to a configuration management tool (e.g. svn or git or RTC (Rational Team Concert))	Mandatory
SH_BT_DEP_20	Functional	The capacity of the repository shall be at least 16 TB	Mandatory
SH_BT_DEP_21	Functional	The integration service shall allow for the execution of a test plan at different simulation levels	Mandatory

2.1.1.2.2.5 Validation Requirements

ID	Requirement Type	Description	Rule
SH_BT_DEP_22	Functional	The validation service shall enable execution of test cases across MiL, SiL, HiL simulation	Mandatory
SH_BT_DEP_23	Functional	The validation service shall enable parallel execution of test cases to speed up testing	Mandatory
SH_BT_DEP_24	Functional	The validation service shall enable reuse of test cases across MiL, SiL, HiL simulation	Mandatory
SH_BT_DEP_25	Functional	The validation service shall enable decision on outcome of each test step (oracle calculation)	Mandatory

2.1.2 Orona

2.1.2.1 Scope of the system

Orona's activities are focused on the design, manufacturing, installation, maintenance, and modernisation of elevators, escalators, and moving ramps and walkways. An elevator installation is a complex CPS composed by a set of elevators that interact to provide service to passengers with the goal of minimising the Average Waiting Time (AWT) and, more recently, also taking into account other criteria such as energy consumption, transport capacity, or overall transit time.

Nowadays, over 250.000 elevators worldwide use Orona's technology. Most of the new functionality in lift traffic groups is provided by software. A single lift has more than 500,000 lines of code, which have to be parametrised to the building characteristics, increasing the complexity of the launching of a new release.

2.1.2.2 ORO Stakeholder Requirements

2.1.2.2.1 General

ID	Requirement Type	Description	Rule
SH_ORO_GR_2	Functional	The analyst shall be able to register and manage new requirements and bugs	Mandatory

SH_ORO_GR_3	Functional	The validator shall be able to specify a validation plan for a dispatcher version at SiL and a Traffic Master at HiL/Operation	Mandatory
SH_ORO_GR_4	Functional	The validator shall be able to specify a deployment for a validation in SiL, HiL or operation	Mandatory
SH_ORO_GR_21	Functional	The analyst shall be able to specify a deployment for a version of the Dispatcher in Operation	Mandatory
SH_ORO_GR_5	Functional	The validator shall be able to specify the monitors for a validation	Mandatory
SH_ORO_GR_9	Functional	The validator shall be able to specify testcases: level of criticality, inputs, and initial status of the Lift group (both optional), criteria to start/finish evaluation and evaluation with respect to the expected results	Mandatory
SH_ORO_GR_10	Functional	The validator shall be able to specify evaluation criteria and the detection of certain conditions based on data	Mandatory
SH_ORO_GR_11	Functional	Evaluation of expected results shall assess about 1) Expected Quality of service: AWT, JT 2) Expected sequence data 3) Expected logical expressions of data	Mandatory

SH_ORO_GR_12	Functional	The validator shall be able to manage the start/finish of an evaluation based on: 1) Identification of a traffic profile: UpPeak,DownPeak, LunchPeak, Interfloor 2) Expected sequence monitored data: sequential status of lifts, CAN/UDP frames 3) Logical expressions of monitored data 4) Passenger injection and simulation Starts/Ends	Mandatory
SH_ORO_GR_13	Functional	The validator shall be able to define test cases that use the data from CAN, Ethernet, code instrumentation or previously persisted data	Mandatory
SH_ORO_GR_15	Functional	The validator shall reuse the same configuration of the evaluation components in Elevate, in the hybrid validation environment and in operation	Mandatory
SH_ORO_GR_16	Non-Functional	The validator shall be able to execute Critical evaluation components in windows and in Linux, in every dispatcher cycle (less than 500 msc.)	Mandatory
SH_ORO_GR_17	Functional	The validator shall reuse the same specification of start/finish criteria for the evaluation, either in Elevate, the hybrid validation environment and in operation.	Mandatory

SH_ORO_GR_18	Non-Functional	The validator shall be able to execute start/finish criteria in windows and in Linux, in every dispatcher cycle (less than 500 msc.)	Mandatory
SH_ORO_GR_19	Functional	The validator shall be able to automatize the validation either in Elevate, in a hybrid simulation environment and in operation. Such validation includes e.g. the triggering criteria for a validation in SiL/HiL and operation	Mandatory
SH_ORO_GR_20	Functional	Deployment shall be done automatically (e.g. as part of an integration test)	Optional

2.1.2.2.2 Software-In-The-Loop

At SiL, the Dispatcher (traffic algorithm) must be considered for design-operation continuous methods.

2.1.2.2.2.1 Monitoring

ID	Requirement Type	Description	Rule
SH_ORO_SIL_MON_1	Functional	The validator shall monitor the status of lifts (floor, position, etc.) and landing calls from Elevate. Data monitored in Elevate shall be obtained from CAN bus in real installations	Mandatory
SH_ORO_SIL_MON_2	Functional	The validator shall monitor information from the dispatch related to the assignment process and cost calculation and information about status of lifts and calls	Mandatory

SH_ORO_SIL_MON_6	Functional	The validator shall monitor data that requires elaboration from Elevate and Dispatcher. Logical data to be monitored shall include passenger estimation, AWT, JT, energy calculation, traffic profile	Mandatory
SH_ORO_SIL_MON_7	Functional	The validator shall be able to monitor operational data, including a timestamp, a trace type and the name, type and value of the data	Mandatory
SH_ORO_SIL_MON_8	Non-Functional	The validator shall be able to monitor and record all data coming from the actual application. That is, with enough throughput to monitor a 125Kb CAN bus loaded at 80%	Mandatory
SH_ORO_SIL_MON_9	Non-Functional	The validator shall be able to monitor the system without affecting its correct behaviour, i.e. in a non-invasive way: not more than a 20% of overhead for 100 traces per second	Mandatory
SH_ORO_SIL_MON_10	Functional	The validator shall be able to store 1Tb of monitoring data	Mandatory
SH_ORO_SIL_MON_11	Functional	Monitored data shall be persisted for further analysis	Mandatory
SH_ORO_SIL_MON_13	Functional	The validator requires a Replay mode that input data from previously stored data (SH_ORO_SIL_MON_11)	Mandatory

2.1.2.2.2.2 Deployment

ID	Requirement Type	Description	Rule
SH_ORO_SIL_DEP_1	Non-Functional	The validator shall require deployment of test artifacts to Windows machines, as far as Elevate runs in Windows	Optional

2.1.2.2.2.3 Validation

Elevate uses an elvx file that provides information about the building and lift installation.

Elevate uses as an input a passenger file.

ID	Requirement Type	Description	Rule
SH_ORO_SIL_VAL_1	Functional	The dispatcher is a dll and an xml file with the parametrisation	Mandatory
SH_ORO_SIL_VAL_2	Functional	The validator uses Elevate to test the dispatcher	Mandatory
SH_ORO_SIL_VAL_5	Functional	The validator requires to execute Elevate to execute the testcases	Mandatory
SH_ORO_SIL_VAL_6	Functional	The validator requires up to two synchronised instances of Elevate running to execute testcases that compare two versions of the dispatcher	Mandatory

2.1.2.2.3 Hardware-In-The-Loop

At HiL, the dispatcher algorithm is integrated in the traffic master.

2.1.2.2.3.1 Monitoring

ID	Requirement Type	Description	Rule
SH_ORO_HIL_MON_1	Functional	The validator requires to monitor all the variables of the CAN bus, from the algorithm, and from the access control system	Mandatory
SH_ORO_HIL_MON_4	Non-Functional	The validator shall be able to monitor the system without affecting its correct behaviour, i.e. in a non-invasive way: the period of the NMT messages sent by the dispatcher should not exceed more than a 10% for 100 traces per second	Mandatory
SH_ORO_HIL_MON_5	Functional	The developer shall reuse the same instrumentation code of the dispatcher in SiL and HiL	Mandatory
SH_ORO_HIL_MON_7	Functional	The validator shall reuse the same monitoring specification format for HiL, SiL and operation	Mandatory

2.1.2.2.3.2 Deployment

ID	Requirement Type	Description	Rule
SH_ORO_HIL_DEP_1	Functional	The validator shall be able to deploy test artifacts to Linux machines, as far as the traffic master runs in Linux	Optional

2.1.2.2.3.3 Validation

ID	Requirement Type	Description	Rule
SH_ORO_HIL_VAL_1	Functional	The validator executes the Testcases in HiL in a hybrid simulation environment where the controllers and the communications are real and the rest of the elements are simulated (in real-time)	Mandatory
SH_ORO_HIL_VAL_2	Functional	The Traffic master is compiled for Windows and Linux and specified as a version and an ini file with the parametrisation	Mandatory
SH_ORO_HIL_VAL_3	Functional	The validator shall configure the controllers before executing a testcase	Optional
SH_ORO_HIL_VAL_4	Functional	The hybrid simulation environment shall be configured with the information of the elvx file and the configuration of the CPS virtual and real nodes	Mandatory
SH_ORO_HIL_VAL_5	Functional	The passenger list used in SiL shall be injected at HiL	Mandatory
SH_ORO_HIL_VAL_6	Functional	A tool to inject passenger lists to the hybrid environment shall be provided	Mandatory
SH_ORO_HIL_VAL_7	Functional	The testcases require to write CAN frames to Orona Bus	Mandatory
SH_ORO_HIL_VAL_8	Functional	The testcases require to write UDP frames to Orona Bus	Mandatory

SH_ORO_HIL_VAL_9	Functional	The validator shall be able to set certain status of the lift group in the Hybrid testing environment before launching a testcase	Optional
SH_ORO_HIL_VAL_10	Functional	The validator shall execute Injectors for passengers, CAN/UDP frames to inject inputs to the SUT	Optional
SH_ORO_HIL_VAL_11	Non-Functional	The validator shall be able to execute a validation non-invasively, without impacting the operation of the CPS	Mandatory

2.1.2.2.4 Operation

2.1.2.2.4.1 Monitoring

ID	Requirement Type	Description	Rule
SH_ORO_OP_MON_1	Non-Functional	The validator shall be able to remote monitor the system over Internet	Optional

2.1.2.2.4.2 Deployment

ID	Requirement Type	Description	Rule
SH_ORO_OP_DEP_1	Non-Functional	The validator and the analyst shall be able to securely and remotely deploy into real installation	Mandatory

2.1.2.2.4.3 Validation

ID	Requirement Type	Description	Rule
SH_ORO_OP_VAL_3	Non-Functional	When testing in operation, the validator shall not be able to inject inputs to the CPSoS	Mandatory

2.1.2.2.4.4 Recovery

ID	Requirement Type	Description	Rule
SH_ORO_OP_REC_1	Functional	The analyst shall be able to roll back a deployment to a previous stable version in case of failure	Mandatory
SH_ORO_OP_REC_2	Functional	The analyst shall be able to configure the system in such a way that it starts recording data (for lab analysis) in case of failure	Mandatory
SH_ORO_OP_REC_3	Functional	The analyst shall be able to configure a recovery based on results of evaluations	Mandatory
SH_ORO_OP_REC_4	Functional	The analyst shall be able to configure the system in such a way that it starts a new validation in case of failure	Mandatory
SH_ORO_OP_REC_5	Non-Functional	The analyst shall be able to execute critical recovery functions in Windows and in Linux	Mandatory

SH_ORO_OP_REC_6	Non-Functional	The analyst shall be aided by a tool capable of computing the logic in less than 500 msc. (a dispatcher cycle)	Mandatory
SH_ORO_OP_REC_7	Non-Functional	The analyst shall be able to execute a recovery non-invasively, without impacting the operation of the CPS	Mandatory

2.2 Adeptness System Requirements

The purpose of this section is to collect Adeptness system requirements. For that aim, each partner has written down what would they do in order to satisfy Adeptness proposal and stakeholder requirements.

2.2.1 System Description

As described in the ADEPTNESS project proposal, the objective of Adeptness system is to develop a workflow to speed-up the software release of CPSoS in operation while guaranteeing its reliability. To this end, the ADEPTNESS system will propose autonomic design-operation continuum methods supported by tools that will combat the fragmentation in methods between development and operation. These methods will reduce the gap from development to operation (continuous delivery) and from operation to development (continuous feedback to development) [Ref. 1].

2.2.2 Definitions, Acronyms and Abbreviations

This subsection describes the definitions, acronyms and abbreviations used in the definition of different Adeptness system level requirements:

Monitor types: There are several conceptually different monitors that will be provided by the monitoring service. These types of monitors are considered:

1) *Physical monitor*: monitors physical data that is captured from the physical world through sensing (e.g. from a temperature sensor, from a CAN bus, etc.).

2) *Application monitor*: monitors data that is internal to the application. It can be monitored only by code instrumentation (e.g. internal variables of an algorithm).

3) *Logical monitor*: provides logical data by processing data that comes from different sources, either aggregating or deriving it from other physical, application or logical monitors (e.g. average temperature, computed from a physical monitor that provides instant temperature).

Validation types: there are several types of validation that shall be provided:

1) Comparison of expected Quality of service with respect to actual values.

2) Expected sequence of different types of monitored data

3) logical expressions of monitored data

Start/finish detection: there are different start/finish checks that shall be provided:

1) Expected sequence monitored data

2) Logical expressions of different types of monitored data

3) Events of tools: Simulation Start/End

Recovery actions shall include:

1) Launch new validation plan

2) Launch new monitoring plan

3) Launch new deployment plan

4) Send an alarm

2.2.3 System Requirements

This section collects Adeptness system level requirements divided into two subsections: Functional Requirements and Non-Functional Requirements.

2.2.3.1 Functional Requirements

2.2.3.1.1 Monitoring Subsystem

ID	Description	Rule	Author	Satisfies Requirement
SYS_MON_1	The Adeptness toolchain shall provide a configurable monitoring service that is pluggable to a CAN fieldbus	Mandatory	IKL	SH_ORO_GR_13 SH_ORO_HIL_MON_1
SYS_MON_2	The Adeptness toolchain shall provide a configurable monitoring service that is pluggable to Ethernet	Mandatory	IKL	SH_ORO_GR_13 SH_ORO_HIL_MON_1

SYS_MON_3	The Adeptness toolchain shall provide a mechanism to persist operational data logs	Mandatory	IKL	SH_ORO_SIL_MON_11 SH_ORO_SIL_VAL_5
SYS_MON_4	The Adeptness toolchain shall support monitoring of application internal variables (*application monitors*) through code instrumentation	Mandatory	IKL	SH_ORO_GR_13 SH_ORO_SIL_MON_1 SH_ORO_SIL_MON_2 SH_ORO_HIL_MON_5 SH_ORO_HIL_MON_1
SYS_MON_5	The Adeptness toolchain shall provide a monitoring service that can be customized for different stakeholder specific data formats over different communication buses (e.g. a specific data format protocol over MQTT or over CAN). Note: This monitoring service shall be configurable to monitor specific physical and application variables (physical monitors and application monitors) in a specific communication bus and in an specific (stakeholder specific) data format.	Mandatory	IKL	SH_ORO_SIL_MON_3 SH_ORO_SIL_MON_5 SH_ORO_SIL_MON_6 SH_ORO_HIL_MON_1 SH_ORO_HIL_MON_7 SH_ORO_HIL_MON_9 SH_BT_MON_1 SH_BT_DEP_17
SYS_MON_7	The monitored data shall include a timestamp, the variable name, the variable type, and the variable value	Mandatory	IKL	SH_ORO_SIL_MON_7
SYS_MON_8	The monitoring subsystem shall monitor all types of monitors described in Section 4.2.2 (Physical/Application/Logical)	Mandatory	IKL	SH_ORO_SIL_MON_3 SH_ORO_SIL_MON_5 SH_ORO_SIL_MON_6

SYS_MON_9	The Adeptness toolchain shall provide a configurable monitoring service that is pluggable to a file that contains previously stored data (the data persisted in SYS_MON_3)	Mandatory	IKL	SH_ORO_SIL_MON_13
SYS_MON_10	The Adeptness toolchain shall provide a DSL to specify a monitoring plan that is valid for both at SiL, HiL and operation levels	Mandatory	IKL	SH_ORO_GR_5
SYS_MON_11	Adeptness toolchain shall run monitorization subsystem on-demand.	Mandatory	UES	SH_ORO_SIL_MON_7
SYS_MON_12	Adeptness toolchain shall support data exchange between monitorization and validation subsystems.	Mandatory	UES	SH_ORO_GR_10
SYS_MON_13	The Adeptness toolchain shall provide a DSL for monitoring	Mandatory	IKL	SH_ORO_GR_5
SYS_MON_14	Monitoring information shall be exchanged through a standardised data model and API (i.e. NGSI-LD)	Mandatory	EGM	SH_ORO_OP_MON_1
SYS_MON_15	The monitoring information model should be extensible (i.e. using linked data principles)	Mandatory	EGM	SH_BT_DEP_2 SH_BT_DEP_17 SYS_MON_1 SYS_MON_2 SYS_MON_7

2.2.3.1.2 Deployment Subsystem

ID	Description	Rule	Author	Satisfies Requirements
SYS_DEP_1	The Adeptness toolchain shall provide a DSL to specify a deployment plan that is valid for both at SiL, HiL and operation levels	Mandatory	IKL	SH_ORO_GR_4
SYS_DEP_2	The Adeptness toolchain shall be able to execute the deployment plan	Mandatory	IKL	SH_ORO_GR_20 SH_BT_DEP_17 SH_BT_DEP_1 SH_BT_DEP_8 SH_BT_DEP_9 SH_BT_DEP_10 SH_BT_DEP_11 SH_BT_DEP_13 SH_BT_DEP_14 SH_BT_DEP_15
SYS_DEP_3	The Adeptness toolchain shall be able to verify the success of the deployment plan	Mandatory	IKL	SH_ORO_GR_20 SH_BT_DEP_17 SH_BT_DEP_12
SYS_DEP_4	The deployment plan shall define the allocation of heterogeneous software components (test artifacts, SUT, etc) to hardware nodes	Mandatory	IKL	SH_ORO_GR_4
SYS_DEP_5	The deployment subsystem shall be invocable from the rest of the subsystems that conform the Adeptness toolchain	Mandatory	IKL	SH_ORO_GR_19

SYS_DEP_6	The Adeptness toolchain shall provide a DSL for deployment	Mandatory	IKL	SH_ORO_GR_4 SH_ORO_GR_21
SYS_DEP_7	The system service bus (broker) shall allow both query and subscription mechanisms (NGSI-LD suggested)	Mandatory	EGM	SH_BT_DEP_15 SH_ORO_OP_MON_1
SYS_DEP_8	The deployment plan shall specify the MiL, SiL and HIl to be provisioned	Mandatory	EGM	SH_BT_1
SYS_DEP_9	The adeptness system shall provide a configurable way to provision MiL, HiL and SiL	Mandatory	EGM	SH_BT_1 SH_ORO_GR_4

2.2.3.1.3 Validation Subsystem

ID	Description	Rule	Author	Satisfies Requirement
SYS_VAL_1	The Adeptness toolchain shall provide tools to inject CAN and UDP frames to stimulate the SUT	Mandatory	MGEP	SH_ORO_HIL_VAL_7 SH_ORO_HIL_VAL_8
SYS_VAL_2	The Adeptness toolchain shall be able to launch executables and microservices	Mandatory	MGEP	SH_ORO_HIL_VAL_10 SH_ORO_SIL_VAL_5 SH_ORO_SIL_VAL_6
SYS_VAL_3	The validation microservice shall be configured to collect the data from different types of monitors	Mandatory	MGEP	SH_ORO_GR_13
SYS_VAL_4	A mechanism to semi-automatically set the status of the system in HiL shall be provided	Mandatory	MGEP	SH_ORO_HIL_VAL_9 SH_ORO_HIL_VAL_3 SH_ORO_HIL_VAL_4

SYS_VAL_5	The Adeptness toolchain shall provide a DSL to specify a validation plan that is valid for either SiL, HiL and operation. The validation plan shall include a deployment plan and a monitoring plan, a suite of testcases and global validation for the test suite	Mandatory	MGEP	Research
SYS_VAL_15	The Adeptness tool chain shall provide a DSL for continuous validation	Mandatory	MGEP	Research
SYS_VAL_6	The DSL for continuous validation shall allow the specification of CPSoS testcases: criticality, inputs and initial status of the system (optional), criteria to start/finish evaluation and validation with respect to the expected result	Mandatory	MGEP	SH_ORO_GR_9 SH_ORO_GR_14 SH_ORO_GR_10
SYS_VAL_7	The Deployment Plan in the Validation Plan shall include all the components required to execute a Validation Plan: simulators, injectors, files, so, dll, monitors, validation software, etc. And shall define the allocation of heterogeneous software components (test artifacts, SUT, etc) to hardware nodes	Mandatory	IKL	SH_ORO_GR_3 SH_BT_DEP_2 SH_BT_DEP_16
SYS_VAL_8	The Validation Plan shall include a monitoring plan with the configuration of the monitors required for the validation	Mandatory	IKL	SH_ORO_GR_8
SYS_VAL_9	The DSL for continuous validation shall allow the	Mandatory	MGEP	SH_ORO_SIL_VAL_2

	specification of SUT and testing environment in SiL and HiL			SH_ORO_SIL_VAL_1 SH_ORO_HIL_VAL_1 SH_ORO_HIL_VAL_2
SYS_VAL_10	The Adeptness toolchain shall provide a common interface to support reusability of tests inputs at all levels (HiL, SiL)	Mandatory	MGEP	SH_ORO_HIL_VAL_6 SH_BT_1 SH_BT_DEP_24
SYS_VAL_11	A tool to transform inputs from simulators to field buses shall be provided	Optional	MGEP	SH_ORO_HIL_VAL_5
SYS_VAL_12	Interface of the validation components and of the detection of start/finish criteria shall be valid for SiL, HiL and Operation	Mandatory	MGEP	SH_ORO_GR_15 SH_ORO_GR_17
SYS_VAL_13	Software that detects start/finish criteria and validation software in critical evaluations shall be executable in windows and Linux over ARM in soft real time	Mandatory	MGEP	SH_ORO_GR_18 SH_ORO_GR_16
SYS_VAL_14	Different types of validation of expected results shall be provided	Mandatory	MGEP	SH_ORO_GR_11 SH_BT_DEP_25 SH_BT_2 SH_BT_DEP_4 SH_BT_DEP_5
SYS_VAL_15	Different types of detection of criteria to start/finish an evaluation shall be provided	Mandatory	MGEP	SH_ORO_GR_12

SYS_VAL_16	Evaluation in HiL and Operation shall be performed non-invasively, without impacting the operation of the CPS	Mandatory	MGEP	SH_ORO_HIL_VAL_11
SYS_VAL_17	The validation microservice shall not inject inputs in Operation	Mandatory	MGEP	SH_ORO_OP_VAL_3
SYS_VAL_18	The validation toolchain shall execute a Validation Plan in SiL, HiL and Operation	Mandatory	MGEP	SH_ORO_GR_19 SH_BT_DEP_22 SH_BT_DEP_3
SYS_VAL_19	The test artifacts generator shall generate test artifacts for SiL, HiL and Operation	Mandatory	MGEP	SH_ORO_GR_19 SH_BT_DEP_22 SH_BT_DEP_6 SH_BT_DEP_7
SYS_VAL_20	Adeptness toolchain shall run validation subsystem on-demand.	Mandatory	UES	SH_BT_DEP_22
SYS_VAL_21	Adeptness toolchain shall support data exchange between monitoring and validation subsystems.	Mandatory	UES	SH_ORO_GR_10
SYS_VAL_22	Adeptness toolchain shall analyse the test execution operational data to obtain a verdict	Mandatory	UES	Research
SYS_VAL_23	Adeptness toolchain shall send validation report data to developer and validator	Mandatory	UES	Research

SYS_VAL_24	The validation management interface should allow selecting sub-part of the validation plan to be executed	Optional	EGM	SH_BT_DEP_16 SH_BT_DEP_21 SH_ORO_GR_3
SYS_VAL_25	The validation system shall regularly report its progress state to the monitoring interface for user information purposes	Mandatory	EGM	SH_BT_4 SH_BT_MON_1
SYS_VAL_26	The validation system shall trigger the execution of defined functions through test data or stimuli.	Mandatory	MDH	SH_BT_DEP_3
SYS_VAL_27	The validation system shall setup parallel machines to speed up MiL/SiL testing.	Mandatory	MDH	SH_BT_DEP_23

2.2.3.1.4 Traceability Subsystem

ID	Description	Rule	Author	Satisfies Requirement
SYS_TRC_2	The DSL for traceability shall be able to understand OSLC standard	Mandatory	UES	SH_BT_DEP_19
SYS_TRC_3	The DSL for traceability shall allow updating lifecycle artifacts from OSLC reports	Mandatory	UES	SH_BT_DEP_19
SYS_TRC_4	The DSL for traceability shall allow notifying lifecycle artifact changes	Mandatory	UES	SH_BT_DEP_19
SYS_TRC_5	Adeptness toolchain shall process OSLC reports	Mandatory	UES	SH_BT_DEP_19

SYS_TRC_6	Adeptness toolchain shall update lifecycle artifacts from OSLC reports	Mandatory	UES	SH_BT_DEP_19 SH_BT_4
SYS_TRC_7	Adeptness toolchain shall allow notifying about lifecycle artifact changes	Mandatory	UES	SH_BT_DEP_19 SH_BT_4
SYS_TRC_8	The traceability microservice shall get validation, oracle, and monitoring microservices data to generate test validation report	Mandatory	UES	SH_ORO_GR_10

2.2.3.1.5 Recovery Subsystem

ID	Description	Rule	Author	Satisfies Requirement
SYS_REC_1	The Adeptness toolchain shall be able to launch executables and microservices (deployment, validation, monitoring)	Mandatory	MGEP	SH_ORO_OP_REC_1 SH_ORO_OP_REC_2 SH_ORO_OP_REC_4
SYS_REC_2	The ADEPTNESS toolchain shall provide a DSL (to specify rules and recovery actions) for recovery	Optional	MGEP	SH_ORO_OP_REC_3
SYS_REC_3	The Adeptness toolchain shall provide a common interface to gather information/results from the oracles and the uncertainty module	Mandatory	MGEP	Research

SYS_REC_4	Recovery functions shall be executable in windows and Linux over ARM	Mandatory	MGEP	SH_ORO_OP_REC_5
SYS_REC_5	The recovery microservice shall be non-invasive regarding the actual application	Mandatory	MGEP	SH_ORO_OP_REC_7
SYS_REC_6	The toolchain shall provide a recovery service in soft real time	Mandatory	MGEP	SH_ORO_OP_REC_6
SYS_REC_7	The Adeptness toolchain shall provide a common interface to gather information/results from the monitoring module	Mandatory	MGEP	SH_ORO_OP_REC_6

2.2.3.1.6 Uncertainty Situation Detection Subsystem

ID	Description	Rule	Author	Satisfies Requirement
SYS_UNC_1	The Adeptness toolchain shall launch executables and microservices for uncertainty situation detection at design and operation time respectively	Mandatory	SRL	SH_BT_3
SYS_UNC_2	Different types of heuristics shall be applied to determine an uncertainty situation has been detected	Mandatory	SRL	Research. SH_ORO_GR_11
SYS_UNC_3	Adeptness toolchain shall support uncertainty detection system and use active or passive learning methods on CPSS at operation and design time.	Mandatory	SRL	SH_ORO_GR_19

SYS_UNC_4	Existing Adeptness toolchain shall be an input to the learning model of uncertainty detection subsystem.	Mandatory	SRL	SH_ORO_GR_10
SYS_UNC_5	The uncertainty detection subsystem shall produce and store operation logs for future use, analysis and inspection	Mandatory	SRL	Research
SYS_UNC_6	The uncertainty situation detection service shall produce a digital twin of it's capabilities and learn on the toolchain behaviour at operation and update on the capabilities	Mandatory	SRL	SH_ORO_GR_10
SYS_UNC_7	The toolchain shall support deployment of real-world scenarios based on learned model on virtual infrastructure	Mandatory	SRL	Research, SH_ORO_GR_2
SYS_UNC_8	The toolchain shall allow software update for the subsystem if required	Mandatory	SRL	Research, SH_ORO_GR_2
SYS_UNC_9	The toolchain shall permit configuration and optimization of applied techniques, heuristic algorithms/methods if required by the subsystem	Mandatory	SRL	Research, SH_ORO_GR_2

2.2.3.1.7 CI/CDE/CD Architecture support

ID	Description	Rule	Author	Satisfies Requirement
SYS_CICD_1	The Adeptness toolchain shall support the registration and management of requirements and/or bugs. Note: Both can be opened and will follow a specific lifecycle (Active, Closed, Resolved)	Mandatory	IKL	SH_ORO_GR_2
SYS_CICD_2	The Adeptness services shall be used through a well stablished API (either synchronous or asynchronous)	Mandatory	IKL	Research
SYS_CICD_3	The Adeptness platform shall provide a management console to (1) monitor the status of the platform itself and (2) manage the different functionalities of the platform	Optional	IKL	Research
SYS_CICD_4	The Adeptness toolchain shall provide an integrated multi-stage pipeline for development, deployment, monitoring and verification.	Mandatory	TUW	SYS_REC_1 SH_BT_DEP_19 SH_BT_DEP_20 SH_BT_DEP_21
SYS_CICD_5	The Adeptness toolchain shall have a system for the management CI/CDE/CD infrastructure. It would include ability to manage monitoring, deployment, validation, uncertainty detection, and recovery.	Mandatory	TUW	SYS_CICD_3 SYS_REC_1

SYS_CICD_6	The Adeptness toolchain shall provide a build server with virtual environment capabilities for cross-compiling.	Mandatory	TUW	SH_BT_DEP_18
SYS_CICD_7	The Adeptness toolchain should be released as a single installation package.	Optional	TUW	
SYS_CICD_8	A method for unique identification of software and hardware components shall be provided	Mandatory	ORO	SH_ORO_GR_20
SYS_CICD_9	The CI/CDE/CD infrastructure should support different target operating systems	Mandatory	ORO	SH_ORO_SIL_DEP_1 SH_ORO_HIL_DEP_1

NOTE: Some of the requirements above have comments.

** SYS_CICD_4: Ability to change stages of the pipeline allow us interoperability and flexibility.*

** SYS_CICD_5: Corelates and overlaps with SYS_CICD_3 I propose merging these two reqs.*

2.2.3.2 Non-Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
SYS_NF_1	Testcases shall be performed non-invasively, without impacting the operation of the CPS	Mandatory	ORO	SH_ORO_SIL_MON_9 SH_ORO_HIL_MON_4 SH_ORO_OPE_VAL_1

SYS_NF_2	The Adeptness CAN monitoring service shall support a minimum data rate of 125 Kb/s	Mandatory	ORO	SH_ORO_SIL_MON_8
SYS_NF_3	The monitoring subsystem shall be able to store data up to 1 Tb	Optional	MGEP	SH_ORO_SIL_MON_10
SYS_NF_4	Adeptness toolchain shall allow integrating different lifecycle management tools using OSLC standard.	Mandatory	UES	Research
SYS_NF_5	Adeptness toolchain shall allow integrating the Adeptness workflow.	Mandatory	UES	Research
SYS_NF_6	Adeptness toolchain shall allow integrating the Adeptness lifecycle management and development methods	Mandatory	UES	Research
SYS_NF_7	Each Adeptness toolchain tools shall provide a rootservice document for discovering server discovery capabilities	Mandatory	UES	Research
SYS_NF_8	Adeptness toolchain tools shall establish Consumer/Friend relationship using OAuth to allow servers interaction	Mandatory	UES	Research

SYS_NF_9	Each Adeptness toolchain tool shall create project area artifact container associations to enable linking between different OSLC resources.	Mandatory	UES	Research
SYS_NF_10	Each Adeptness toolchain tool may define specific integration requirements of each lifecycle application to get the integrations working	Optional	UES	Research
SYS_NF_11	Each Adeptness toolchain tool shall define a TRS (Tracked Resource Set) provider	Optional	UES	Research

2.2.3.2.1 Configuration

ID	Description	Rule	Author	Satisfies Requirement
SYS_NF_Con_1	The adeptness toolchain shall use a repository for the deployable artifacts	Optional	IKL	Research

2.2.3.2.2 Reliability

ID	Description	Rule	Author	Satisfies Requirement
SYS_NF_Rel_1	Adeptness toolchain shall define certain reliability assessment criteria for all the involved subsystems in both design and operation time	Optional	SRL	

2.2.3.2.3 Regulatory

ID	Description	Rule	Author	Satisfies Requirement
SYS_NF_Reg_1	Adeptness toolchain shall allow the traceability in the whole lifecycle of CPSoS from initial development to operation and recommission using OSLC standard	Mandatory	UES	Research

2.3 STO Level Requirements

2.3.1 STO 1.1 Microservice Based Architecture and Interfaces Requirements

The objective of STO1.1 is to design a novel CPSoS architecture for adaptive deployment in CPSoS using microservice and container-based paradigm. This framework will provide embedded microservices with generic functions (such as messaging clients, alerts, rule engines, etc.) and used by the new specific functionalities for design-operations continuum.

2.3.1.1 MICROSERVICE BASED ARCHITECTURE AND INTERFACES REQUIREMENTS

2.3.1.1.1 Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO1.1_1	The Adeptness toolchain shall provide one mechanism to add reqs/bugs and send notification to the involved stakeholders (validation team, development team)	Mandatory	IKL	SYS_CICD_1
STO1.1_2	The Adeptness toolchain shall provide one mechanism to receive notifications from a version control tool (e.g. commits)	Mandatory	IKL	SYS_CICD_1

STO1.1_3	The Adeptness toolchain shall provide one mechanism to execute scripts or processes (pipeline steps). This functionality could be provided by an automation server (e.g. compile/deployment/validation (test executor)	Mandatory	IKL	SYS_CICD_4
STO1.1_4	The Adeptness toolchain shall provide a repository for software artifacts & config files artifacts	Mandatory		SYS_DEP_2
STO1.1_5	The Adeptness toolchain shall provide a deployment mechanism with an interface that allows to load a deployment plan.	Mandatory	IKL	SYS_DEP_2
STO1.1_6	The Adeptness toolchain shall provide a deployment mechanism with an interface that allows to start the execution of its loaded development plan.	Mandatory	IKL	SYS_DEP_2
STO1.1_7	The Adeptness toolchain shall provide service(s) that allow to monitor the CPSoS	Mandatory	IKL	SYS_MON_1 SYS_MON_2
STO1.1_8	The Adeptness toolchain shall provide service(s) that allow to validate the CPSoS	Mandatory	IKL	SYS_VAL_2
STO1.1_9	The Adeptness toolchain shall provide service(s) that allow the recovery of the CPSoS	Mandatory	IKL	SYS_REC_6

STO1.1_10	The Adeptness toolchain shall provide service(s) that allow uncertainty detection	Mandatory	SRL	SYS_UNC_1
STO1.1_11	The Adeptness toolchain shall provide service(s) that allow automated test generation.	Mandatory	IKL	SYS_VAL_10 SYS_VAL_18
STO1.1_12	The Adeptness toolchain shall provide service(s) that allow traceability of CPSoS artifacts.	Mandatory	IKL	SYS_TRC_8
STO1.1_13	The Adeptness toolchain shall provide continuous feedback on test progress to the operator	Mandatory	EGM	SYS_VAL_25
STO1.1_39	The Adeptness toolchain shall provide feedback on deployment progress	Mandatory	IKL	SYS_DEP_3
STO1.1_14	The Adeptness services shall provide a well-known interface through a synchronous mechanism (e.g. REST API)	Mandatory	IKL	SYS_CICD_2
STO1.1_15	The Adeptness services shall provide event mechanism through an asynchronous publish subscribe mechanisms and a well-defined data format	Mandatory	IKL	SYS_CICD_2
STO1.1_16	The Adeptness services shall allow their configuration through files	Optional	IKL	SYS_MON_9

STO1.1_17	The Adeptness services shall provide a RESTful API to start/stop its functionality	Mandatory	IKL	SYS_VAL_2 SYS_MON_11
STO1.1_18	The Adeptness services shall provide a RESTful API to allow its health checking	Mandatory	IKL	SYS_CICD_3
STO1.1_20	The monitoring service shall publish asynchronous topics (e.g. MQTT) to be consumed by the validation service	Mandatory	IKL	SYS_MON_7 SYS_MON_12
STO1.1_21	The monitoring service shall publish asynchronous topics (e.g. MQTT) to be consumed by the uncertainty service	Mandatory	IKL	SYS_VAL_7
STO1.1_35	The monitoring service shall provide a RESTful API to be configured as defined in the monitoring plan.	Mandatory	IKL	SYS_VAL_8 SYS_VAL_3
STO1.1_38	The monitoring service shall publish asynchronous topics (e.g. MQTT) to be consumed by the recovery service	Mandatory	IKL	SYS_REC_7
STO1.1_23	The validation service shall subscribe to the topics published by the monitoring service	Mandatory	MGEP	SYS_VAL_3

STO1.1_24	The validation service shall be able to publish asynchronous topics to be consumed by the uncertainty service and recovery service	Mandatory	MGEP	SYS_UNC_4
STO1.1_25	The uncertainty service shall subscribe to the topics published by the monitoring service	Mandatory	SRL	SYS_MON_5 SYS_VAL_3
STO1.1_26	The uncertainty service shall subscribe to the topics published by the validation service	Mandatory	SRL	SYS_MON_10 SYS_VAL_3 SYS_VAL_5
STO1.1_27	The uncertainty service shall invoke the RESTful API of the recovery service	Mandatory	SRL	SH_ORO_OP_REC_1 SH_ORO_OP_REC_2
STO1.1_28	The uncertainty detection shall configure recovery microservice with required rules to specify the recovery actions to be launched.	Mandatory	SRL	SYS_REC_3
STO1.1_29	The uncertainty detection shall provide an interface to connect to test oracles, validation services and recovery microservices	Mandatory	SRL	SYS_VAL_10 SYS_VAL_12
STO1.1_30	The recovery service shall subscribe to the topics published by the validation service	Mandatory	MGEP	SYS_REC_3

STO1.1_31	The recovery service shall subscribe to the topics published by the uncertainty service	Mandatory	MGEP	SYS_REC_3
STO1.1_32	The recovery service shall provide a RESTful API to be configured by the uncertainty service	Mandatory	MGEP	SYS_REC_3
STO1.1_34	The recovery service shall subscribe to the topics published by the monitoring service	Optional	MGEP	SYS_REC_3

2.3.1.1.2 Non-Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO1.1_40	The Adeptness services that compose the toolchain shall be communicated across Ethernet	Mandatory	IKL	SYS_CICD_4
STO2.1_41	The Adeptness services shall be executed in different platforms and operating systems	Mandatory	IKL	SYS_CICD_9
STO2.1_42	The Adeptness services shall be deployed independently	Mandatory	IKL	SYS_CICD_4
STO1.1_43	The Adeptness services shall have a loosely coupled with the other services	Mandatory	IKL	SYS_CICD_4

2.3.2 STO 1.2 Deployment Modelling Requirements

The objective of STO1.2 is to define a modelling language for deployment to help engineers specify the deployment workflow, which will consider the tight interaction between the CPSoS and its environment (STO1.2). This language will enable to specify the nodes to be updated, under which status of the

component/CPS/CPSoS, rollback mechanisms, etc. The specification within the modelling language will be transformed into tasks and scripts for continuous deployment tools. The deployment will be done both in the virtual and real infrastructure of the CPSoS, enabling to deploy any type of software artefact (e.g., control software, monitors, test oracles, etc.).

2.3.2.1 Definitions, Acronyms and Abbreviations

Deployment plan: Set of actions necessary to be able to have deployed, in the appropriate hardware node, the indispensable artifacts to carry out the validation of the SUT.

DSL: Domain Specific Language

Deployable element: component that shall be deployed in a target node. Includes executables, libraries, test artifacts and in general any type of file susceptible of being deployed in a target node

Target Node: device that is part of the hardware infrastructure

Step: The deployment plan is composed by several steps. Each step contains an action, a precondition and a postcondition (e.g. result notification)

Action: Two types of actions shall be supported: (1) deploy and (2) launch. Deploy: download a specific artifact to a target. Launch: execute the artifact

2.3.2.2 Deployment Modelling Requirements

2.3.2.2.1 Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO1.2_1	The DSL shall support the specification of the deployment plan.	Mandatory	IKL	SYS_DEP_1 SYS_DEP_6
STO1.2_2	The deployment plan shall be specified through a DSL/Schema XML	Mandatory	IKL	SYS_DEP_1 SYS_DEP_6
STO1.2_3	The DSL shall allow the definition of the target nodes			SYS_DEP_4
STO1.2_4	The DSL shall allow the definition of the deployable elements (executables, libraries, test artifacts, etc.)	Mandatory	IKL	SYS_DEP_4

STO1.2_5	The DSL shall allow to allocate each deployable element to a target node	Mandatory	IKL	SYS_DEP_4
STO1.2_6	The DSL shall allow the specification of conditions to trigger the deployment. The condition events that trigger the deployment shall be produced by monitors. For instance, a monitor could fire an event when certain action is performed in the CI/CD orchestrator, such as e.g. a code commit or a direct user action. Other conditions that could be considered are e.g. a test pass/fail event or a notification sent by a monitor that is recording operational data.	Mandatory	IKL	SYS_DEP_5
STO1.2_7	The DSL shall allow the configuration of event notifications associated to the result of the deployment (success or fail).	Mandatory	IKL	SYS_DEP_3

2.3.3 STO 1.3 Deployment Orchestation Requirements

STO 1.3 describes a virtual CPSoS deployment orchestration. A software artefact deployment service requires set of tools and methods that ensure software artefacts are released properly as deployable images, the physical and logical network of deployment target is maintained, deployment targets are in correct state for deployment.

2.3.3.1 Definitions, Acronyms and Abbreviations

Step: The deployment plan is composed by several steps. Each step contains include an action, a precondition and a postcondition (e.g. result notification).

Action: Two types of actions shall be supported: (1) deploy and (2) launch. Deploy: download a specific artifact to a target. Launch: execute the artifact

Artifact: any type of element to be downloaded to a target node. These artifacts can be both executable and non-executable (libraries, configuration files …)

2.3.3.2 Deployment Orchestration Requirements

2.3.3.2.1 Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO1.3_1	The deployment system shall execute the deployment plan. This deployment plan shall be written according to the DSL specified in STO1.2	Mandatory	IKL	SYS_DEP_1 SYS_DEP_2
STO1.3_2	The deployment system shall verify the success of the execution of the deployment plan.	Mandatory	IKL	SYS_DEP_3
STO1.3_3	The deployment system shall notify the result of the execution of the deployment plan. To notify this result, the deployment system shall raise an event following a well-known interface (e.g. json over MQTT)	Mandatory	IKL	SYS_DEP_3
STO1.3_4	The deployment system shall subscribe to the deployment triggering events described in the deployment plan	Mandatory	IKL	SYS_DEP_5
STO1.3_6	The Adeptness toolchain shall provide a mechanism that allow the deployment of all the necessary artifacts not only to set up a system but also to observe and validate it (i.e. SUT, tests artefacts & config files)	Mandatory	IKL	SYS_DEP_1
STO1.3_7	The deployment mechanism shall be able to be configured with multiple steps, each performing a specific action	Mandatory	IKL	SYS_DEP_2

STO1.3_8	The deployment mechanism shall provide the traceability of each of its steps and actions through notification events, it will also create trigger notification events for the rest of the toolchain depending on the monitoring status.	Mandatory	IKL	SYS_DEP_3
STO1.3_9	Two types of actions shall be supported: (1) deploy and (2) launch	Optional	IKL	SYS_DEP_2
STO1.3_10	The deployment system shall use a repository for picking up the deployable artifacts.	Mandatory	IKL	SYS_NF_Con_1
STO1.3_19	The deployment orchestration system shall be controlled by other components in the CI/CD toolchain via API.	Mandatory	TUW	SYS_DEP_2
STO1.3_20	The deployment orchestration system shall be platform agnostic or support a set of potential platforms.	Mandatory	TUW	SYS_DEP_4
STO1.3_21	The deployment system will have a self-check mechanism that will notify CI/CD management about potential outages.	Mandatory	TUW	SYS_DEP_3
STO1.3_22	The deployment system shall support different payload formats.	Mandatory	TUW	SYS_DEP_2
STO1.3_23	The deployment system shall support runtime updates.	Optional	TUW	TBD
STO1.3_24	The deployment system shall support extensions for a platform specific end target deployment.	Optional	TUW	SYS_DEP_2

STO1.3_25	The deployment system shall support secure communication channels towards deployment targets.	Mandatory	TUW	TBD
STO1.3_26	The deployment system shall support fog/edge extension platform to facilitate deployment on devices without direct Internet communication capabilities.	Optional	TUW	SYS_DEP_2
STO1.3_27	The deployment system shall provide authentication service for the deployment artefacts.	Optional	TUW	TBD

NOTE: Some of the requirements above have comments.

** STO1.3_20: With it is implied complete software and architecture ambiguity.*

2.3.4 STO 2.1 Monitoring Microservice Requirements

STO 2.1 relates to the Monitoring Microservice that will provide observation and data acquisition capabilities on hardware and software components of a CPSoS. Monitoring can be performed on synchronously on runtime capturing data in parallel to the operation of the system, or asynchronously where the data is stored first and analysed later. Monitoring service is in close relation to other service such as continuous validation, deployment, recovery mechanisms and unforeseen situation detection service.

2.3.4.1 Definitions, Acronyms and Abbreviations

Monitor types: There are several conceptually different monitors that will be provided by the monitoring service. These types of monitors are considered:

(1) Physical monitor: monitors physical data that is captured from the physical world through sensing (e.g. from a temperature sensor, from a CAN bus, etc.).

(2) Application monitor: monitors data that is internal to the application. It can be monitored only by code instrumentation (e.g. internal variables of an algorithm).

(3) Logical monitor: provides logical data by processing data that comes from different sources, either aggregating or deriving it from other physical, application or logical monitors (e.g. average temperature, computed from a physical monitor that provides instant temperature).

2.3.4.2 Monitoring Microservice Requirements

2.3.4.2.1 Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO2.1_1	The service shall offer an interface to plug-in different physical network connectors (e.g. CAN, Ethernet…). ***Depending on the stakeholder, different fieldbuses must be monitored, and thus, different physical connectors developed.***	Mandatory	IKL	SYS_MON_1 SYS_MON_2
STO2.1_2	The service shall offer an interface to plug-in different stakeholder specific data parsers over a specific physical network connector. For the same connector (e.g. CAN), different stakeholder specific data protocols may be defined).	Mandatory	IKL	SYS_MON_5
STO2.1_3	The monitoring service shall be configured to extract logical data from persisted information or from physical drivers.	Mandatory	IKL	SYS_MON_1 SYS_MON_2 SYS_MON_9
STO2.1_4	The monitoring service shall provide a message-event interface (subscription mechanism) to be consumed by the other services.	Mandatory	IKL	SYS_MON_6
STO2.1_5	The monitoring service shall provide an interface to check its health	Mandatory	IKL	SYS_CICD_2
STO2.1_6	The service configuration shall include the specification of data to be monitored and eventually other non-functional requirements such as e.g. data sampling interval. This requirement refers to the need to provide a configuration mechanism that could be eventually extended.	Mandatory	IKL	SYS_MON_5

STO2.1_7	The monitoring service shall provide an interface to allow its configuration through an API.	Mandatory	IKL	SYS_MON_6
STO2.1_8	The monitoring service shall provide an interface to allow its configuration through a configuration file.	Mandatory	IKL	SYS_MON_5
STO2.1_9	The service shall provide a client library to enable internal application data monitoring (i.e. application monitors) over Ethernet. In other words, a library to enable code instrumentation shall be provided. This library should be used by application developers to trace internal application variables. Note: Useful for tracing of internal application data not available in the fieldbus	Mandatory	IKL	SYS_MON_4
STO2.1_10	Data format of the application monitoring traces shall be defined in json format	Mandatory	IKL	SYS_MON_7
STO2.1_11	The monitored data shall be optionally persisted to disk	Mandatory	IKL	SYS_MON_3
STO2.1_12	Logical monitoring shall be performed by elaborating/aggregating physical or application monitoring. To achieve this, specific ad-hoc monitoring services might be necessary	Mandatory	IKL	SYS_MON_5
STO2.1_13	The generic monitoring service might provide logical monitoring through the execution of configurable basic rules (e.g. mean, basic algebraic rules)	Optional	IKL	TBD

STO2.1_14	The generic monitoring service shall have an option of local and cloud-based monitoring.		TUW	TBD
STO2.1_15	The generic monitoring service shall be constructed from two main components: behaviour collector and correctness analyser.	Optional	TUW	SYS_MON_1 SYS_MON_2 SYS_MON_8
STO2.1_16	The generic monitoring service will provide ability to perform synchronous and asynchronous monitoring.	Optional	TUW	SYS_MON_10 STO2.1_12
STO2.1_17	The generic monitoring service will provide a monitor code generation from the equivalent temporal logic formula.	Optional	TUW	TBD
STO2.1_18	The behaviour collector part of the generic monitoring service shall be able to collect system and application specific data.	Optional	TUW	TBD
STO2.1_19	After a release is deployed real platform, the generic monitoring microservice shall start running.	Mandatory	UES	SYS_MON_11
STO2.1_20	After a release is deployed virtual environment, the generic monitoring microservice shall start running.	Mandatory	UES	SYS_MON_11
STO2.1_21	When requested from the continuous validation microservice, the specified test monitorization data shall be sent to it.	Mandatory	UES	SYS_MON_12

NOTE: Some of the requirements above have comments.

** STO2.1_14: It is easier to perform data analysis on runtime if the data is monitored in cloud.*

** STO2.1_15: Separation of these task makes the monitoring service more flexible and adaptable to different platforms.*

** STO2.1_16: Creating a generic service where we could configure monitoring mode.*

** STO2.1_17: The monitoring rules can be defined in temporal logic and verified before deployed in monitoring system.*

2.3.4.2.2 Non-Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO2.2_1	The behaviour collector resource consumption shall be limited	Optional	TUW	TBD

2.3.5 STO 2.2 Continuous Validation Microservice Requirements

The objective of STO2.2 is to provide a continuous validation microservice that will enable to automatically validate new software releases of CPSoS at different test levels as well as in operation. To this end, several test artifacts, including test oracles, test arbiters, test stimuli, etc. will interact among them with the goal of detecting bugs as early as possible. The behaviour of every component in this STO will be specified through a DSL and automatically generated by using different automated techniques (STO 3.1). The continuous validation microservice aim at detecting bugs when these are exhibited for the first time. When this happens, this microservice communicates with the recovery microservice (STO 2.4) for taking the appropriate action.

The execution of test cases is coordinated by a test executor, which decides in each moment which test should be executed when and at which level. Tests will be automatically executed by using Automated verification and validation test benches. This test benches will be used to test that the new release works correctly within a virtual infrastructure. Test cases are executed from the professional TaaS tool (EGM) at different teste levels. (Co)-simulation-based testing as a service will be used, and test cases (including test inputs and test oracles) will be re-used across MiL, SiL and HiL test levels. This will provide an agile verification and validation framework for the automated test execution of test cases for simulation-based testing of CPSoS.

To automate the verification and validation process, test oracles are required. These test oracles are the sources in charge of determining whether a test has passed or failed. Three types of test oracles will be investigated: 1) Specified oracles, which are assertion-based like test oracles; 2) Implicit oracles, which leverage the power of AI algorithms to determine whether the system is behaving correctly or not; and 3), derived oracles, which consider metamorphic relations in order to validate that the system is behaving as expected. These oracles will be executable both at design-time as well as at operation-time, enabling to have an streamlined continuous validation workflow.

2.3.5.1 Definitions, Acronyms and Abbreviations

Test case: behavioural feature or behaviour specifying tests. A test case specifies how a set of test components interact with an SUT to realize a test objective. Test cases are owned by test contexts, and therefore have access to all parts of the test configuration, other global variables (e.g., data pools, etc.) or further behavioural features (e.g., auxiliary methods). A test case always returns a verdict. The verdict may be arbitrated -- calculated by the arbiter, or non-arbitrated (i.e., provided by the test behaviour). In CPSoS, a test case might have no test inputs, especially when these are in operation, where the test input is substituted by the input from users (environment)

Test oracle: It is the source in charge of providing whether a test has passed or failed; this is termed as verdict. The test oracles shall have access to the inputs and outputs of the system.

Verdict: Predefined enumeration specifying the set of possible evaluations of a test case. Five enumeration literals are defined: none, pass, fail, inconclusive, error.

- none: the test case has not been executed yet
- pass: the system under test adheres to the expectations
- inconclusive: The evaluation cannot be evaluated to be pass or fail
- fail: The system under test differs from the expectation.
- error: An error has occurred within the testing environment.

The Verdict type may be extended by the users with more literals.

In the context of CPSoS, besides test verdicts being enumerations, they might also have the option of being numerical, which provides with information of how critical a failure was or how far was the system from failing.

Oracle criticality level: Level of criticality of the functionalities evaluated by a test oracle. In CPSoS, there are functionalities that might be critical, and thus, test oracles might require its observation to be in soft real-time during operation.

Test executor: A software component in charge of executing test cases and evaluating its outcomes. It coordinates the execution of the test cases via the test control and evaluates whether test cases have been successful through the test arbiter.

Test context: A test context acts as a grouping mechanism for a set of test cases. The composite structure of a test context is referred to as test configuration. The classifier behaviour of a test context may be used for test control.

Test configuration: Collection of test component objects and of connections between the test component objects and to the SUT. A test configuration defines both (1) test component objects and connections when a test case is started (the initial test configuration) and (2) the maximal number of test component objects and connections during the test execution.

SUT: It is applied to one or more properties of a classifier to specify that they constitute the system under test. The features and behaviour of the SUT is given entirely by the type of the property to which the stereotype is applied. It is part of a test context. It refers to a system, subsystem, or component that is being tested. An SUT

can consist of several objects. The SUT is stimulated via its public interface operations and signals by the test components. No internals of a SUT are known or accessible during the test case execution, due to its black-box nature.

Test arbiter: it is a predefined interface defining operations used for arbitration of tests. Test cases, test contexts and the runtime system can use realizations of this interface to assign verdicts of tests and to retrieve the current verdict of a test. The purpose of a test arbiter is to determine the final verdict for a test case. This determination is done according to an arbitration strategy, which is provided in the implementation of the arbiter interface. A test case or a test context use the arbiter to evaluate test results and to assign the overall verdict of a test case or test context, respectively. There is a default arbitration algorithm based on functional, conformance testing, which generates Pass, Fail, Inconclusive and Error as verdict, there the precedence of the verdicts is defined as Pass<Inconclusive<Fail<Error

Test component. Test components are part of the test environment and are used to communicate with the system under test (SUT) and other test components. The main function of test components is to drive a test case by stimulating the SUT through its provided interfaces and to evaluate whether the actual responses of the SUT comply with the expected ones.

Preconditions: The initial state of the SUT needed to start the execution of a test case.

Test stimulus: Test data sent to the SUT to control it and to make assessments about the SUT when receiving the SUT reactions to these stimuli.

Test starting criteria: condition under which a test case can start its execution.

Test finishing criteria: condition under which a test case can finish its execution.

Criteria checker: module in charge of evaluating whether the criteria for starting/finishing a test case have been given.

Test as a Service: Service-based approach for remote and automated CPSoS testing. It resolves constraints regarding coordination, costs, and scalability issues of traditional software testing.

2.3.5.2 Continuous Validation Microservice Requirements

2.3.5.2.1 Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO2.2_1	The continuous validation service shall include an automatic test executor for CPSoS: the software component that coordinates the execution of CPSoS tests in MiL, SiL, HiL and Operation. The test Executor shall execute a Validation Plan for CPSoS in all levels (SiL, HiL and operation). The Test Executor shall: 1. Order the deploy and the configuration of the simulation artefacts, monitors, the SUT, the testcases and all the components that are required to execute a validation plan 2. Scheduler: Execute the testcases 3. Test arbiter: Evaluate the results of the testcases	Mandatory	MGEP	SYS_VAL_18
STO2.2_4	The text Executor shall execute the simulators, injectors of inputs to the SUT (in MiL, SiL and HiL) and orchestrate microservices. It shall control the order of testcases and support synchronised testcases	Mandatory	MGEP	SYS_VAL_2
STO2.2_5	The test arbiter shall be configured with the rules to provide an overall test verdict from verdicts of the test cases	Mandatory	MGEP	TBD
STO2.2_6	The continuous validation microservice shall offer an interface to plug-in different physical network connectors (e.g. CAN, Ethernet…) to inject stimulus to the system	Mandatory	MGEP	SYS_VAL_10 SYS_VAL_11 SYS_VAL_4

STO2.2_7	A tool to transform and inject test stimulus and preconditions in MiL/SiL to stakeholder specific field buses (CAN, Ethernet, etc) shall be provided	Mandatory	MGEP	SYS_VAL_1
STO2.2_8	A criteria checker with a common interface valid for all validation levels shall be provided	Mandatory	MGEP	SYS_VAL_12 SYS_VAL_13
STO2.2_9	Different criteria checkers shall be provided 1) time 2) events from the simulations tools 3) status of the CPSoS 4) sequence detected in the CPSoS	Mandatory	MGEP	SYS_VAL_15
STO2.2_10	All the different oracles types shall share a common interface and shall be executable at all validation levels	Mandatory	MGEP	SYS_VAL_13 SYS_VAL_12

Test cases shall be executable using (co)-simulation-based testing as a service.

Test oracles shall provide a verdict in terms of (PASS/FAIL/INCONCLUSIVE/NONE/ERROR).

Test oracles shall optionally provide a quantitative degree of satisfaction for the fulfilment of requirements and/or properties.

ID	Description	Rule	Author	Satisfies Requirement
STO2.2_11	Different types of test oracles shall be provided: 1) time-continuous signals 2) discrete values 3) sequence of discrete events 4) triggering executable files that will allow validating functionalities manually programmed	Mandatory	MGEP	SYS_VAL_14

Test oracles shall have the possibility to validate time-continuous signals

Test oracles shall have the possibility to validate discrete values

Test oracles shall have the possibility to validate a sequence of discrete events

The test oracles shall have the capability of triggering executable files that will allow validating functionalities manually programmed

A level of criticality shall be able to be specified to test oracles

The verdict of test oracles shall be sent to a test arbiter to determine whether a test has passed/failed

ID	Description	Rule	Author	Satisfies Requirement
STO2.2_12	Critical oracles shall be executed in soft real time in HiL and operation	Mandatory	MGEP	SYS_VAL_16

Test logs shall include physical, application data, compliance to triggering conditions and any other information (e.g., verdicts from oracles) relevant for test case analysis

ID	Description	Rule	Author	Satisfies Requirement
STO2.2_18	The validation microservice shall not inject inputs in Operation	Mandatory	MGEP	SYS_VAL_17
STO2.2_13	Test oracles shall guarantee at least a 95% of precision and recall	Mandatory	MGEP	Research

2.3.5.2.2 Non-Functional Requirements

2.3.5.2.2.1 Reliability

ID	Description	Rule	Author	Satisfies Requirement
STO2.2_14	The AI-based test oracles shall include a set of metrics that quantitatively measure different aspects of reliability during the training period of the applied machine learning models and techniques.	Optional	MGEP	TBD
STO2.2_15	The AI-based test oracles shall include a set of metrics that quantitative measure different aspects of reliability after the training period of the applied machine learning models and techniques.	Optional	MGEP	TBD
STO2.2_16	The AI-based test oracles shall perform statistical tests to enable rigours comparisons on the reliability metrics during the training period of the applied machine learning models and techniques.	Optional	MGEP	TBD
STO2.2_17	The AI-based test oracles shall perform statistical tests to enable rigours comparisons on the reliability metrics during after the training period of the applied machine learning models and techniques.	Optional	MGEP	TBD

2.3.6 STO 2.3 Unforeseen Situation Detection Microservice Requirements

The Unforeseen Situation Detection microservice will enable to detect uncertainty situations to achieve the scientific challenges referred as STO 2.3. To this end, we will investigate the use of adaptive techniques (e.g., the L* algorithm and extending it to handle live stream of data from operational CPSoS) and passive learning techniques (e.g., state merging with the AALERGIA algorithm) to incrementally improve the models learned during the design-time based on the monitoring data obtained from STO2.1. Similarity functions will be used to determine whether we have learned an unforeseen situation. If so, we will determine whether the situation is safe or not communicating with the validation microservice described functioning to achieve STO 2.2. If an unforeseen unsafe situation is determined, this microservice will invoke the recovery microservice, described in STO 2.4, to prevent a system from failure.

This scientific challenge involves developing a framework to learn unforeseen situations during the operation of the CPSoS and taking appropriate actions to protect the CPSoS from failure. This task will incrementally build models of the CPSoS based on the monitoring data of the CPSoS and its environment. The existing model built in during design time or existing models from use case providers will be input for this task. We will investigate the use of adaptive techniques (e.g., the L* algorithm) and passive learning techniques (e.g., state merging with the AALERGIA algorithm) to step-by-step improving the models of CPSoS as monitoring data becomes available. To learn unforeseen situations, we will implement a set of heuristics, for example, based on a variety of similarity functions (such as Euclidean, and Levenshtein functions). The key idea is that if we learn a behaviour which is significantly different than the known behaviour (using the similarity functions), it means that an unforeseen situation is observed. In case, an unforeseen situation is not safe, determined, appropriate recovery mechanisms will be invoked to protect the CPSoS from failure. Also, predetermined test oracles will also be continuously checked at the runtime to see if the CPSoS is not close to failure. If the distance to failure is smaller to potentially lead to failure of CPSoS, then appropriate recovery mechanisms will be invoked to protect the CPSoS from failure. The unforeseen learning methods developed in this task will be deployed at the elastic architecture based on embedded microservices, which will allow moving the entity from cloud to the edge and from the edge to a local entity (e.g., by including it as a non- priority task in the target processor) and vice versa.

2.3.6.1 Definitions, Acronyms and Abbreviations

CPSoS: Cyber-Physical Systems of Systems

Runtime: Running in the real product or system that is deployed in a production environment and in normal operation.

Test case: behavioural feature or behaviour specifying tests. A test case specifies how a set of test components interact with an SUT to realize a test objective. Test cases are owned by test contexts, and therefore have access to all parts of the test configuration, other global variables (e.g., data pools, etc.) or further behavioural features (e.g., auxiliary methods). A test case always returns a verdict. The verdict may be arbitrated -- calculated by the arbiter, or non-arbitrated (i.e., provided by the test behaviour).

Test oracle: It is the source in charge of providing whether a test has passed or failed. It has access to the inputs and outputs of the system.

Test context: A test context acts as a grouping mechanism for a set of test cases. The composite structure of a test context is referred to as test configuration. The classifier behaviour of a test context may be used for test control.

SUT: It is applied to one or more properties of a classifier to specify that they constitute the system under test. The features and behaviour of the SUT is given entirely by the type of the property to which the stereotype is applied. It is part of a test context. It refers to a system, subsystem, or component that is being tested. An SUT can consist of several objects. The SUT is stimulated via its public interface operations and signals by the test components. No internals of an SUT are known or accessible during the test case execution, due to its black-box nature.

Test stimulus: Test data sent to the SUT to control it and to make assessments about the SUT when receiving the SUT reactions to these stimuli.

Evaluation criteria: Evaluation criteria are the interventions to be used to assess the development of the microservice, i.e., accuracy, efficiency, and unbiasedness.

Operation uptime: The period microservice was in operation starting from the launch time until the time it stopped.

Digital Twin: Abstraction of CPSoS in operation within a microservice satisfying a specific use case scenario of ADEPTNESS toolchain.

The Image below shows a high-level overview of our operation time uncertainty detection

CPSoS: Cyber-Physical Systems of Systems

Runtime: Running in the real product or system that is deployed in a production environment and in normal operation.

Test case: behavioural feature or behaviour specifying tests. A test case specifies how a set of test components interact with an SUT to realize a test objective. Test cases are owned by test contexts, and therefore have access to all parts of the test configuration, other global variables (e.g., data pools, etc.) or further behavioural features (e.g., auxiliary methods). A test case always returns a verdict. The verdict may be arbitrated -- calculated by the arbiter, or non-arbitrated (i.e., provided by the test behaviour).

Test oracle: It is the source in charge of providing whether a test has passed or failed. It has access to the inputs and outputs of the system.

Test context: A test context acts as a grouping mechanism for a set of test cases. The composite structure of a test context is referred to as test configuration. The classifier behaviour of a test context may be used for test control.

SUT: It is applied to one or more properties of a classifier to specify that they constitute the system under test. The features and behaviour of the SUT is given entirely by the type of the property to which the stereotype is applied. It is part of a test context. It refers to a system, subsystem, or component that is being tested. An SUT can consist of several objects. The SUT is stimulated via its public interface operations and signals by the test components. No internals of an SUT are known or accessible during the test case execution, due to its black-box nature.

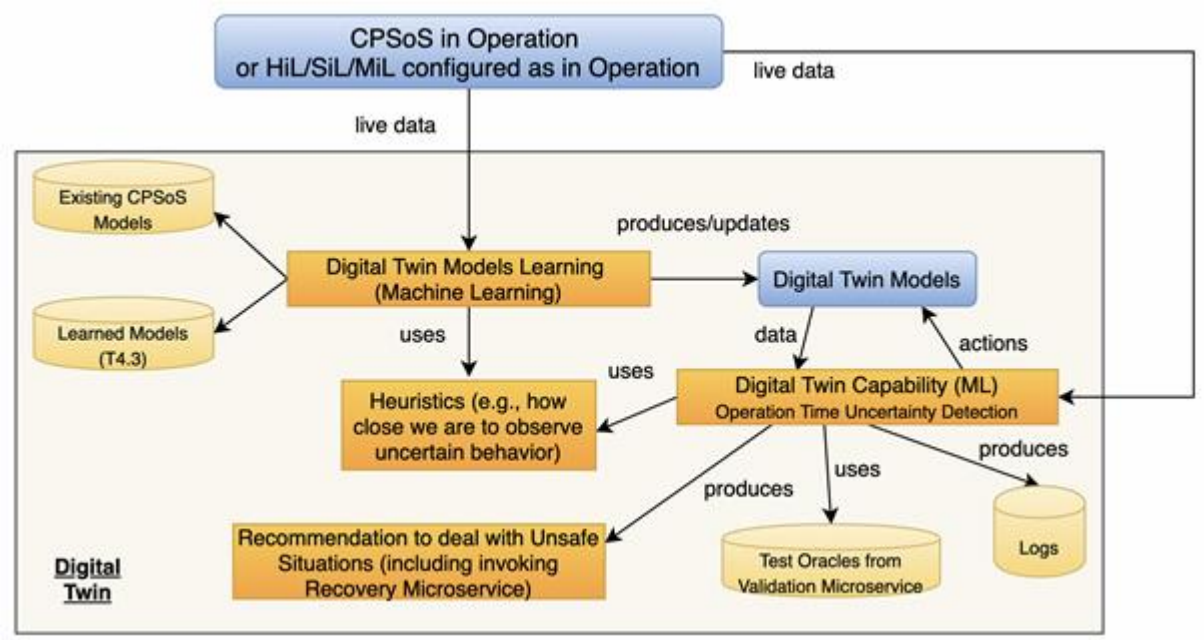
Test stimulus: Test data sent to the SUT to control it and to make assessments about the SUT when receiving the SUT reactions to these stimuli.

Evaluation criteria: Evaluation criteria are the interventions to be used to assess the development of the microservice, i.e., accuracy, efficiency, and unbiasedness.

Operation uptime: The period microservice was in operation starting from the launch time until the time it stopped.

Digital Twin: Abstraction of CPSoS in operation within a microservice satisfying a specific use case scenario of ADEPTNESS toolchain.

The Image below shows a high-level overview of our operation time uncertainty detection



2.3.6.2 Unforeseen Situation Detection Microservice Requirements

2.3.6.2.1 Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO2.3_1	The ADEPTNESS toolchain shall support the uncertainty detection of unforeseen situations within a CPSoS in operation	Mandatory	SRL	SYS_UNC_1
STO2.3_2	The ADEPTNESS toolchain shall support the uncertainty detection of unforeseen situations within MiL configured as CPSoS in operation.	Mandatory	SRL	SYS_UNC_1

STO2.3_3	The ADEPTNESS toolchain shall support the uncertainty detection of unforeseen situations within HiL configured as CPSoS in operation.	Mandatory	SRL	SYS_UNC_1
STO2.3_4	The ADEPTNESS toolchain shall support the uncertainty detection of unforeseen situations within SiL configured as CPSoS in operation.	Mandatory	SRL	SYS_UNC_1
STO2.3_5	The uncertainty detection microservice shall implement heuristics (e.g., based on relevant similarity functions) to determine whether we have learned an unforeseen situation	Mandatory	SRL	SYS_UNC_2
STO2.3_6	The uncertainty detection microservice shall improve the models learned during the design-time	Mandatory	SRL	SYS_UNC_3
STO2.3_7	The uncertainty detection microservice shall use adaptive (/active) and passive techniques (Machine Learning Algorithms) on live monitoring data to learn and determine the uncertain situations.	Mandatory	SRL	SYS_VAL_14
STO2.3_8	The uncertainty detection microservice shall be deployed at the elastic architecture	Mandatory	SRL	Research
STO2.3_9	The ADEPTNESS toolchain shall support the configuration of various scenarios (operation or HiL/SiL/MiL configured as in operation)	Mandatory	SRL	SYS_VAL_12 SYS_VAL_18

STO2.3_10	The ADEPTNESS toolchain shall support integrating various models from various scenarios.	Optional	SRL	SYS_MON_5 SYS_VAL_18
STO2.3_11	The ADEPTNESS toolchain shall support optimal configuration/tuning of employed machine learning algorithms.	Mandatory	SRL	SYS_MON_5
STO2.3_12	The existing model of CPSoS shall act as an input to the uncertainty situation detection microservice to learn and improve the model.	Mandatory	SRL	SYS_VAL_10 SYS_MON_5
STO2.3_13	The uncertainty situation detection microservice shall provide a digital twin of its capabilities in operation (MiL).	Mandatory	SRL	SYS_UNC_5 SYS_UNC_6
STO2.3_14	The uncertainty situation detection microservice shall update a digital twin of its capabilities in operation (MiL) based on the learning model.	Mandatory	SRL	SYS_UNC_5 SYS_UNC_6
STO2.3_15	The uncertainty situation detection microservice shall produce operation logs during each uptime period.	Mandatory	SRL	SYS_UNC_4

2.3.6.2.2 Non-Functional Requirements

2.3.6.2.2.1 Configuration

ID	Description	Rule	Author	Satisfies Requirement
STO2.3_16	The uncertainty detection microservice shall support modular configuration for algorithms and scenarios	Optional	SRL	SYS_MON_5

2.3.6.2.2.2 Performance

ID	Description	Rule	Author	Satisfies Requirement
STO2.3_17	The uncertainty detection microservice shall respond to monitoring, validation, and recovery microservice in x time.	Optional	SRL	SYS_VAL_7 SYS_VAL_8
STO2.3_18	The uncertainty detection microservice shall invoke recovery microservice in x time?	Optional	SRL	SYS_VAL_7 SYS_VAL_8
STO2.3_19	The uncertainty detection microservice shall receive data from monitoring microservice every y unit of time.	Optional	SRL	SYS_VAL_7 SYS_VAL_8
STO2.3_20	The uncertainty detection microservice shall receive test results from validation microservice every y unit of time.	Optional	SRL	SYS_VAL_7 SYS_VAL_8

2.3.6.2.2.3 Reliability

ID	Description	Rule	Author	Satisfies Requirement
STO2.3_21	The uncertainty detection microservice shall guarantee uptime of x%	Optional	SRL	SYS_NF_Rel_1
STO2.3_22	The uncertainty detection microservice shall provide successful demand x% of the time	Optional	SRL	SYS_NF_Rel_1
STO2.3_23	The uncertainty detection microservice shall be able to deal with x% noise in data	Optional	SRL	SYS_NF_Rel_1
STO2.3_24	The uncertainty detection microservice shall define an evaluation criterion, in terms of uncertainty situation detection accuracy to assess the reliability of the microservice	Mandatory	SRL	SYS_NF_Rel_1

This shall involve estimating the accuracy of the applied machine learning algorithms by calculating error, i.e., Mean Square Error (MSE).

ID	Description	Rule	Author	Satisfies Requirement
STO2.3_25	The uncertainty detection microservice shall define an evaluation criterion, in terms of uncertainty situation detection efficiency to assess the reliability of the microservice	Mandatory	SRL	SYS_NF_Rel_1

This shall involve comparing the efficiency of the applied machine learning algorithms by calculating sample variance over the test numbers, i.e., Mean Square Error (MSE).

ID	Description	Rule	Author	Satisfies Requirement
STO2.3_26	The uncertainty detection microservice shall define an evaluation criterion, in terms of uncertainty situation detection unbiasedness, for microservice reliability assessment.	Mandatory	SRL	SYS_NF_Rel_1

This unbiasedness shall refer to the decision verdict of the applied machine learning algorithms.

2.3.7 STO 2.4 Recovery Microservice Requirements

The objective of STO2.4 is to provide a recovery microservice that will enable CPSoS software releases to perform an action when an error or uncertain situation is detected in operation. To this end, results from test oracles and uncertainty detector modules will be used with the goal of starting recovery actions to maintain the system safe as early as possible. The recovery actions will be specified through a specific language that will be defined to this aim and these actions will be have an impact in other microservices: new validation, monitoring and deployment plans could be launched to the respective microservices.

2.3.7.1 Definitions, Acronyms and Abbreviations

Recovery Actions: Recovery actions involve a logical and structural model reconfiguration, to represent newly activated behaviours.

Critical Recovery Functions: Recovery Actions to be launched when failure that could result in loss of life, significant property damage or damage to the environment is detected. Time response is one critical factor to consider when executing this type of functions.

CI: Continuous Integration

Verdict: Predefined enumeration specifying the set of possible evaluations of a test case.

Failing Verdict: The system under test differs from the set of possible evaluations of a test case.

Test oracle: It is the source in charge of providing whether a test has passed or failed; this is termed as verdict. The test oracles shall have access to the inputs and outputs of the system.

Oracle criticality level: Level of criticality of the functionalities evaluated by a test oracle. In CPSoS, there are functionalities that might be critical, and thus, test oracles might require its observation to be in soft real-time during operation.

2.3.7.2 Recovery Microservice Requirements

2.3.7.2.1 Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO2.4_5	The recovery microservice shall allow the specification of recovery actions based on a validation plan either for MiL, HiL or real	Mandatory	MGEP	SYS_REC_1
STO2.4_6	The recovery microservice shall allow the specification of recovery actions based on a monitoring plan (example: start recording data of some days to send to the laboratory to design a new validation plan)	Mandatory	MGEP	SYS_REC_1
STO2.4_7	The recovery microservice shall allow the specification of recovery actions based on new deployment plan (example: deploy a previous stable version)	Mandatory	MGEP	SYS_REC_1
STO2.4_8	The recovery microservice shall allow the specification of recovery actions based on alerts to CI service	Mandatory	MGEP	SYS_REC_1
STO2.4_9	The recovery microservice shall allow the specification of recovery actions based on an adaptation of the system	Optional	MGEP	SYS_REC_1

STO2.4_10	The circumstances that will be handled by the recovery mechanisms shall be failing verdicts from critical and non-critical oracles and uncertainty module	Mandatory	MGEP	SYS_REC_3

2.3.7.2.2 Non-Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO2.4_12	Critical recovery functions shall be executed in Linux and windows platforms	Mandatory	MGEP	SYS_REC_4
STO2.4_13	The recovery microservice shall be non-invasive regarding the actual application	Mandatory	MGEP	SYS_REC_5

2.3.7.2.2.1 Configuration

ID	Description	Rule	Author	Satisfies Requirement
STO2.4_3	The recovery microservice shall be configured with rules that specify the recovery action to be launched	Mandatory	MGEP	SYS_REC_2
STO2.4_4	A language to specify recovering actions shall be provided	Mandatory	MGEP	SYS_REC_2

2.3.7.2.2.2 Performance

ID	Description	Rule	Author	Satisfies Requirement
STO2.4_14	The toolchain shall provide a recovery service in soft real time	Mandatory	MGEP	SYS_REC_6

2.3.8 STO 3.1 DSL For Continuous Validation Requirements

The DSL will enable the specification of CPSoS properties that will allow for the automated generation of verification and validation test artefacts at different levels (MiL, SiL, HiL and Operation). The latter aims to provide a framework to automatically and incrementally build statistical models of the CPSoS by operational data from different sources (e.g., log files, or test logs). This model will later be used to automatically (re)-generate test cases.

2.3.8.1 Definitions, Acronyms and Abbreviations

A validation Plan shall reference a deployment plan with the components that shall be deployed for the execution of the validation plan

The validation plan shall reference a monitoring plan with the configuration of the monitoring microservice

The validation plan shall define a test context

The test context includes the test configuration a definition of the SUT and the test cases

A test context shall include a test configuration to be set before the CPSoS test cases are executed

Test components shall include all the tools, executables and files required to execute a CPSoS test case

A CPSoS test case shall be defined as inputs and initial status of the CPSoS (both optional), criteria to start/finish evaluation and evaluation with respect to the expected results (oracle)

2.3.8.2 DSL for Continuous Validation Requirements

2.3.8.2.1 Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO3.1_1	All the required information for the specification of the validation plan shall be specified through a Domain Specific Language (DSL)	Mandatory	MGEP	SYS_VAL_5
STO3.1_2	The DSL shall enable the specification of CPSoS properties, such as requirements and interfaces, which will allow the automated generation of test cases and test oracles	Mandatory	MGEP	SYS_VAL_9
STO3.1_3	The DSL shall have enough expressiveness to specify all the necessary properties to allow the generation of different artifacts: -Test Executor -Test stimulus -System status -Triggering and finishing criteria -Oracles	Mandatory	MGEP	SYS_VAL_6
STO3.1_4	The toolchain for the generation of test artifacts (e.g., oracles, test cases, pipelines, etc.) shall take information generated from the DSL tool (e.g., XML, JSON) to automatically generate these artifacts	Mandatory	MGEP	SYS_VAL_19

STO3.1_7	The DSL shall support the separation of the input stimuli (that affects the system state) and the test oracle (i.e., that decides if a system requirement is fulfilled or not).	Mandatory	MDH	TBD
STO3.1_8	The DSL shall support the Easy Approach to Requirement Specification (EARS) language	Mandatory	MDH	TBD
STO3.1_9	The DSL shall support embedded system specific properties such as timing and events.	Mandatory	MDH	TBD
STO3.1_10	The DSL shall support each abstract entity to be mapped to the actual implementation by means of analysing the design documentation.	Mandatory	MDH	TBD
STO3.1_11	The toolchain shall be used for writing test cases containing timing information that result in correct verdicts based on temporal specifications	Mandatory	MDH	TBD
STO3.1_12	The toolchain shall be used for interactive writing of requirements using the DSL.	Mandatory	MDH	TBD

NOTE: Some of the requirements above have comments.

** STO3.1_1: SYS_VAL_5*

** STO3.1_2: SYS_VAL_9*

** STO3.1_3: SYS_VAL_6*

** STO3.1_4: SYS_VAL_19*

2.3.8.2.2 Non-Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO3.1_5	The DSL shall have sufficient expressiveness to automatically generate 80% of the test validation artifacts	Mandatory	MGEP	SYS_VAL_19
STO3.1_6	The test validation artifacts generators shall be able to automatically generate 85% of the functionalities specified in the DSL	Mandatory	MGEP	SYS_VAL_19

2.3.9 STO 3.2 Unforeseen Situation Detection At Design Time Requirements

This scientific challenge involves developing a framework to learn unforeseen situations before a new CPS in a CPSoS in deployed. The model developed through research and development of a Domain Specific Language and Models from Operational data for continuous validation of CPSoS will be used with the ultimate goal to support: 1) testing in the presence of learned unforeseen situations, 2) in case, the new CPS version cannot deal with such situations, developing an update of the software and deploying it on the CPS to protect the overall CPSoS from failure in the operation. We will use machine learning techniques to learn unforeseen situations by applying active and passive learning techniques and representing the learned unforeseen situations in a suitable model (e.g., automata).To achieve this scientific objective we will investigate the application of passive learning techniques (e.g., invariant detection techniques) on the available data (e.g., operational data, test logs) to learn behaviours of CPSoS under various environmental conditions and represent the learned model in a suitable representation (e.g., same as design models, statistical models) (STO3.1).

Upon collection of operational data, active learning techniques (e.g., reinforcement learning) shall be used to actively execute CPSoS and its associated simulators/emulators to detect unforeseen situations to achieve STO3.2. We will also use a statistical model to train artificial intelligence algorithms (e.g., recurrent neural networks) to automatically re-generate statistically significant test cases (i.e., test cases that can be given in reality) to satisfy STO3.3 which eventually will trace operational information to lifecycle artefacts for further improvement.

The passive machine learning will apply techniques (e.g., invariant detection) on the available data to learn behaviours of CPSoS. Followed by this, we will devise novel mechanisms to determine whether we have learned new behaviours by comparing the learned behaviours with the known behaviours of the CPSoS from the available design models. The active machine learning techniques will interact with the CPS to improve learned models with passive learning techniques. For active learning, we will investigate the use of reinforcement learning (e.g., with Q learning) to guide the execution of the CPS towards discovering unknown behaviour. The output of the learning process would be models of unforeseen situations. Such models will

then be used for additional test case generation followed by test case execution. In case, a CPS cannot handle these learned situations, a software update will be deployed to handle these situations.

2.3.9.1 Definitions, Acronyms and Abbreviations

Design Time: During phases of development and testing. This would include testing phases in MiL, SiL and HiL.

Active Machine Learning techniques: The learning model is updated based on the detected shift in the operational data stream using detection test and perform action on the CPSoS under consideration (SiL, HiL, HiL).

Passive Machine Learning techniques: The learning system is updated continuously as the CPSoS environment is constantly produces operational data.

Test load: Adding automated load or demand on the uncertainty detection module of the CPSoS to check module elasticity

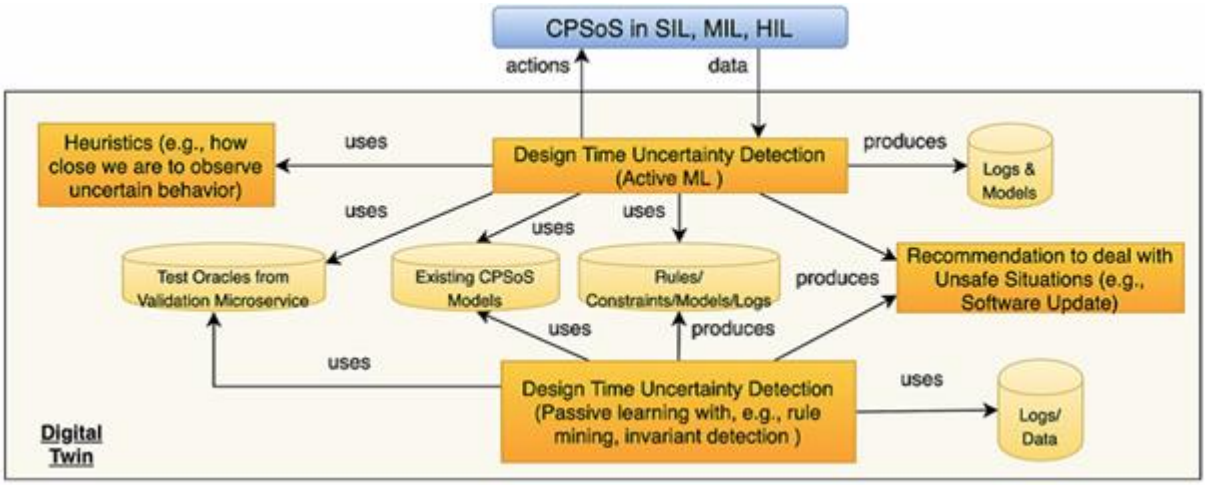
Performance Capabilities: Capabilities of the unforeseen situation detection module with example of intervention as dependence of the module on machine learning algorithms and techniques. The aim of Design Time Uncertainty Detection module shall focus on being modular with application of the detection techniques.

Static design time uncertainty: This involves - design time uncertainty shall perform unforeseen situation detection during phases of development and testing. This includes testing phases in MiL, SiL and HiL.

Dynamic/interactive/runtime uncertainty: This involves - execution time/runtime/interactive/dynamic time unforeseen situation detection technique during execution of the system, independently of the environment that is running on. That is, in simulation or in the target (in HiL or in real).

Reliability metrics: Interventions or metrics to assess the reliability of the uncertainty detection microservice upon deployment on a use case scenario.

The Image below shows a high-level overview of our design time uncertainty detection



2.3.9.2 Unforeseen Situation Detection At Design Time Requirements

2.3.9.2.1 Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO3.2_1	The ADEPTNESS toolchain shall support the design time (static and dynamic/interactive/ runtime) uncertainty detection of unforeseen situations within a CPSoS in MiL.	Mandatory	SRL	SYS_UNC_1
STO3.2_2	The ADEPTNESS toolchain shall support the design time (static and dynamic/interactive/ runtime) uncertainty detection of unforeseen situations within a CPSoS in SiL	Mandatory	SRL	SYS_UNC_1
STO3.2_3	The ADEPTNESS toolchain shall support the design time (static and dynamic/interactive/ runtime) uncertainty detection of unforeseen situations within a CPSoS in HiL.	Mandatory	SRL	SYS_UNC_1
STO3.2_4	The design time uncertainty detection shall reproduce real world scenarios in the virtual infrastructure.	Mandatory	SRL	SYS_UNC_7
STO3.2_5	The design time uncertainty detection shall learn from experience by applying relevant ML (e.g., reinforcement learning) together with the test oracles to determine if situations are unsafe.	Mandatory	SRL	SYS_UNC_3 SYS_UNC_4

STO3.2_6	The design time uncertainty detection will require the definition of novel functions (e.g., rewards functions) to guide the process towards learning unforeseen situations of CPSoS.	Mandatory	SRL	SYS_UNC_3 SYS_UNC_4
STO3.2_7	The design time uncertainty detection shall apply passive machine learning techniques on the available data to learn behaviours of CPSoS.	Mandatory	SRL	SYS_UNC_3
STO3.2_8	The design time uncertainty detection shall provide a methodology to select relevant passive machine learning techniques.	Mandatory	SRL	SYS_UNC_3 SYS_UNC_4
STO3.2_9	The design time uncertainty detection shall apply novel comparison mechanisms to determine whether unknown behaviours are discovered.	Mandatory	SRL	SYS_UNC_3 SYS_UNC_4 SYS_UNC_5
STO3.2_10	The design time uncertainty detection shall apply active machine learning techniques that interact with the CPS to improve learned models with passive learning techniques.	Mandatory	SRL	SYS_UNC_3
STO3.2_11	The design time uncertainty detection shall provide a methodology to select relevant active machine learning techniques.	Mandatory	SRL	SYS_UNC_3

STO3.2_12	The design time uncertainty detection shall provide a methodology for optimized and automated parameter tuning of the machine learning techniques.	Mandatory	SRL	SYS_UNC_6 SYS_UNC_9
STO3.2_13	The design time uncertainty detection shall provide testing methodology to generate test cases from the learned models.	Mandatory	SRL	SYS_VAL_2 SYS_REC_7 SYS_REC_9 SYS_REC_12
STO3.2_14	The output of the design time uncertainty detection learning process shall be the model of unforeseen situations.	Mandatory	SRL	SYS_UNC_4 SYS_VAL_6 SYS_VAL_7 SYS_VAL_8 SYS_VAL_10
STO3.2_15	The uncertainty situation detection at design shall produce module logs and uncertainty models during the active learning period.	Mandatory	SRL	SYS_UNC_5
STO3.2_16	The uncertainty situation detection at design shall produce module logs during the passive learning period according to mining rules	Mandatory	SRL	SYS_UNC_3 SYS_UNC_5
STO3.2_17	The design time uncertainty detection shall be provided with software updates if required and detected by the learning model to manage the generated models of unforeseen situations.	Mandatory	SRL	SYS_UNC_8 SYS_CICD_4 SYS_CICD_5 SYS_CICD_6

2.3.9.2.2 Non-Functional Requirements

2.3.9.2.2.1 Configuration

ID	Description	Rule	Author	Satisfies Requirement
STO3.2_18	The toolchain shall support optimal configuration/tuning of employed machine learning algorithms.	Mandatory	SRL	SYS_UNC_9

2.3.9.2.2.2 Performance

ID	Description	Rule	Author	Satisfies Requirement
STO3.2_19	The design time uncertainty detection shall be configurable to perform under test load at scale.	Optional	SRL	SYS_UNC_6 SYS_UNC_8 SYS_UNC_9
STO3.2_20	The design time uncertainty shall report on performance capabilities in terms of learning speed of the applied learning techniques and models.	Optional	SRL	SYS_UNC_6 SYS_UNC_8 SYS_UNC_9
STO3.2_21	The design time uncertainty shall report on performance capabilities in terms of dependability of the applied learning techniques and models.	Optional	SRL	SYS_UNC_6 SYS_UNC_8 SYS_UNC_9

2.3.9.2.2.3 Reliability

ID	Description	Rule	Author	Satisfies Requirement
STO3.2_22	The design time uncertainty detection shall include a set of metrics that quantitative measure different aspects of reliability during the training period of the applied machine learning models and techniques.	Optional	SRL	SYS_NF_Rel_1
STO3.2_23	The design time uncertainty detection shall include a set of metrics that quantitative measure different aspects of reliability after the training period of the applied machine learning models and techniques.	Optional	SRL	SYS_NF_Rel_1
STO3.2_24	The design time uncertainty shall perform statistical tests to enable rigours comparisons on the reliability metrics during the training period of the applied machine learning models and techniques.	Optional	SRL	SYS_NF_Rel_1
STO3.2_25	The design time uncertainty shall perform statistical tests to enable rigours comparisons on the reliability metrics during after the training period of the applied machine learning models and techniques.	Optional	SRL	SYS_NF_Rel_1

2.3.10 STO 3.3 Automated Test Regeneration Requirements

TBD

2.3.10.1 Definitions, Acronyms and Abbreviations

TBD

2.3.10.2 Automated Test Regeneration Requirements

2.3.10.2.1 Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO3.3_1	The automated test generation microservice shall follow the test plan of testing at MiL/SiL/HiL levels	Mandatory	MDH	SYS_VAL_18
STO3.3_2	The automated test generation microservice shall trigger the execution of test cases at MiL/SiL/HiL levels	Mandatory	MDH	SYS_VAL_18
STO3.3_3	The automated test generation microservice shall decide on the outcome of each test step through specific type of oracle for testing at MiL/SiL/HiL levels	Mandatory	MDH	SYS_VAL_14 SYS_VAL_15
STO3.3_4	The automated test generation microservice shall reuse test cases across simulation levels for testing at MiL/SiL/HiL levels	Mandatory	MDH	SYS_VAL_10
STO3.3_5	The automated test generation microservice shall report on the coverage metrics at different simulation levels of testing at MiL/SiL/HiL levels	Mandatory	MDH	SYS_VAL_18

STO3.3_6	The automated test generation microservice should support the augmentation of already created test cases, manually or automatically generated.	Mandatory	MDH	SYS_VAL_10
STO3.3_7	The automated test generation microservice should provide support for online and offline test selection	Optional	MDH	SYS_VAL_18
STO3.3_9	The automated test generation microservice should provide support to generate a specific test case that follows the exact same execution trace of the original test case while still being different from the original test case.	Mandatory	MDH	SYS_VAL_10
STO3.3_10	The automated test generation microservice should provide support to augment original test cases for achieving 100% test coverage.	Mandatory	MDH	SYS_VAL_10
STO3.3_11	The automated test generation microservice should provide support to reuse the results of the execution of the original test cases on the regenerated and augmented test cases.	Mandatory	MDH	SYS_VAL_10

2.3.11 STO 3.4 Impact of Operation Data On Lifecycle Artifacts Requirements

The objective of STO3.4 is to extracting knowledge by using operational data of the CPSoS. All these situations will help engineers to reduce debugging costs. In addition, based on the OSLC standard, life-cycle management techniques will be traced with operational data extracted from the CPSoS. This will help on visualising all the affected life-cycle artefacts when problems arise in execution, which will also help on reducing re-commissioning costs.

The objective of traceability subsystem is to adapt professional tools for design-operation continuum of CPSoS and integrate them with the objective of supporting the methods investigated in ADEPTNESS project. The main objectives can be summarised as follows:

- To adapt professional tools for integrate CPSoS lifecycle artifacts between the developed prototypical tools.

- To integrate the tools among them to provide the appropriate toolchain to the use case providers.

2.3.11.1 Definitions, Acronyms and Abbreviations

DSL: Domain Specific Language

OSLC: Open Service for Lifecycle Collaboration

TRC: Tracked Resource Set

2.3.11.2 Impact of Operation Data on Lifecycle Artifacts Requirements

2.3.11.2.1 Functional Requirements

ID	Description	Rule	Author	Satisfies Requirement
STO3.4_9	When verdict is published by oracle microservice, traceability microservice shall update the linked development artifacts status with the test validation report data in the developer tool via OSLC.	Mandatory	UES	SYS_TRC_5 SYS_TRC_6 SYS_TRC_7 SYS_VAL_23
STO3.4_10	When verdict is published by oracle microservice, traceability microservice shall update the linked V&V artifacts status with the test validation report data in the validator tool via OSLC.	Mandatory	UES	SYS_TRC_5 SYS_TRC_6 SYS_TRC_7 SYS_VAL_23

STO3.4_46	The test validation report shall contain the following data: - Test Environment Setup - Test Inputs - Test Start Criteria - Test Validation Conditions - Test Finishing Criteria - Test Verdict - Test Execution Operational Data - Linked Development Artifacts - Linked V&V Artifacts	Mandatory	UES	SYS_MON_11 SYS_MON_12 SYS_VAL_20 SYS_VAL_23
STO3.4_47	Traceability microservice shall get test validation report data from related validation, oracle and monitorization microservices.	Mandatory	UES	SYS_TRC_8 SYS_MON_11 SYS_MON_12 SYS_VAL_20
STO3.4_11	DSL shall enable the comprehension of the received OSLC validation report.	Mandatory	UES	SYS_TRC_2
STO3.4_12	DSL shall allow updating the status of development and V&V artifacts with OSLC validation report data.	Mandatory	UES	SYS_TRC_3
STO3.4_13	In case that validation report verdict is not passed, DSL shall allow notifying it in the development tool	Mandatory	UES	SYS_TRC_4

STO3.4_16	In case that validation report verdict is not passed, DSL shall allow notifying it in the validation tool	Mandatory	UES	SYS_TRC_7

2.3.11.2.2 Non-Functional Requirements

2.3.11.2.2.1 General

ID	Description	Rule	Author	Satisfies Requirement
STO3.4_26	Traceability microservice shall allow maintaining traceability of CPSoS artifacts during the development in operation using OSLC standard	Mandatory	UES	SYS_NF_Reg_1
STO3.4_27	Traceability microservice shall allow tracing data from operation to development-time using OSLC standard, in order to identify lifecycle artefacts that are affected by situations occurred at operation-time	Mandatory	UES	SYS_NF_Reg_1
STO3.4_28	Traceability microservice shall allow the integration of the developed design-operation continuum tools in Adeptness toolchain	Mandatory	UES	SYS_NF_4
STO3.4_29	Traceability microservice shall be able to integrate Adeptness workflow to allow design-operation continuum engineering in CPSoS within the developed toolchain	Mandatory	UES	SYS_NF_5
STO3.4_30	Traceability microservice shall be able to integrate Adeptness lifecycle management and development methods to allow for the design-operation continuum engineering of CPSoS	Mandatory	UES	SYS_NF_6

STO3.4_31	Each Adeptness toolchain tool shall define its OSLC domain model	Mandatory	UES	SYS_TRC_5 SYS_TRC_6
STO3.4_32	Each Adeptness toolchain tool shall provide https and SSL support	Mandatory	UES	SYS_TRC_5
STO3.4_33	Adeptness toolchain tools shall establish Consumer/Friend relationship using Oauth to allow servers interaction	Mandatory	UES	SYS_NF_8
STO3.4_34	Each Adeptness toolchain tool shall provide a rootservice document for discovering server discovery capabilities	Mandatory	UES	SYS_NF_7
STO3.4_35	Each Adeptness toolchain tool shall create project area artifact container associations to enable linking between different OSLC resources.	Mandatory	UES	SYS_NF_9
STO3.4_36	Each Adeptness toolchain tool shall define what link types are available based on the chosen artifact container association	Mandatory	UES	SYS_NF_9
STO3.4_37	Each Adeptness toolchain tool shall define a TRS provider	Mandatory	UES	SYS_NF_11
STO3.4_38	Adeptness toolchain tools may customize the preview dialogs	Optional	UES	SYS_NF_10
STO3.4_39	Adeptness toolchain tools may customize the delegated dialogs	Optional	UES	SYS_NF_10

2.3.11.2.2.2 Reliability

ID	Description	Rule	Author	Satisfies Requirement
STO3.4_25	Traceability microservice shall allow the traceability of CPSoS artifacts during the whole lifecycle of the system, including operation and recommissioning using OSLC standard	Mandatory	UES	SYS_NF_Reg_1

3 ADEPTNESS REQUIREMENTS V&V

3.1 Acceptance Tests

1.1.1 BT Acceptance Tests

"Releases of the train control system to train are always preceded by a regression test on the vehicle test level. This regression test is called:

Train control release test

A release of train control usually contains a new version of TCMS. The train control release test is outside the scope of the TCMS project organization.

New versions of TCMS are released internally within Bombardier before they are tested on train. An internal TCMS release is made after a regression test on the system level called:

TCMS release test

The release test specifications contain a selected amount of test cases for safety related functionality and other important functionality. Among the test cases is a train drive cycle (e.g. start of train from standstill with acceleration to a selected speed, coasting and braking to standstill)."

3.1.1.1 BT_SHR_Test_1

Author: BT

Status: Proposed

Validates Requirements:

All SH requirements for BT

Validation Method: TBD

Test Data:

Preconditions	Description	Expected Results
TBD	If the tests do not reach the acceptance criteria the test leader may decide that the test specification and execution should be repeated over and over until the criteria is reached.	TBD

3.1.1.2 BT_SHR_Test_2

Author: BT

Status: Proposed

Validates Requirements:

All SH requirements for BT

Validation Method: TBD

Test Data:

Preconditions	Description	Expected Results
TBD	The Validation Plan will be executed at MiL and SiL with a smoke test	TBD

3.1.1.3 BT_SHR_Test_3

Author: BT

Status: Proposed

Validates Requirements:

All SH requirements for BT

Validation Method: TBD

Test Data:

Preconditions	Description	Expected Results
TBD	The Validation Plan will be executed at HiL with a smoke test	TBD

3.1.1.4 BT_SHR_Test_4

Author: BT

Status: Proposed

Validates Requirements:

All SH requirements for BT

Validation Method: TBD

Test Data:

Preconditions	Description	Expected Results
TBD	TCMS Components: The test specification is up to date and has been approved. For the test specification to be up to date, the component design and the software requirements that are the basis for that design must also be up to date. All test cases have been executed and passed. The test record has been written.	TBD

3.1.1.5 BT_SHR_Test_5

Author: BT

Status: Proposed

Validates Requirements:

All SH requirements for BT

Validation Method: TBD

Test Data:

Preconditions	Description	Expected Results
TBD	If the tests do not reach the acceptance criteria the test leader may decide that the test specification and execution should be repeated over and over until the criteria is reached.	TBD

3.1.1.6 BT_SHR_Test_6

Author: BT

Status: Proposed

Validates Requirements:

All SH requirements for BT

Validation Method: TBD

Test Data:

Preconditions	Description	Expected Results
TBD	Generally, for every testing level the following priority shall be followed in testing (test specification and test execution): Test of safety-related functionality Positive tests – Functionality under normal conditions	TBD

3.1.2 ORO Acceptance Tests

3.1.2.1 ORO_SHR_Test_1

Author: ORO

Status: Proposed

Validates Requirements:

All general reqs

SH_ORO_HIL_VAL_6

SH_ORO_HIL_MON_2

Validation Method: Test

Test Data:

Preconditions	Description	Expected Results
HiL infrastructure ready and connected to Adeptness toolchain	Validation of a new version at MiL, HiL and operation: 1. The developer finishes a new version and tags it 2. The Adeptness toolchain notifies the validator V that the new version is ready 3. The validator V designs and develops the validation plan for the version and registers it in the Adeptness toolchain. Such validation plan is intended for the MiL, HiL and Operation level and includes the deployment and monitoring plans needed to perform the validation and the steps to execute the testcases. More specifically, it follows these steps: 3.1. Define the passenger list and the installations in which the version will be validated 3.2 Define the testcases (start/finish criteria, oracles, inputs: "when an UpPeak is identified, the AWT shall be below 30 sc", "when a call is registered, a lift is assigned to it", etc. 3.3. Define the deployment of the different test artifacts and SUT into the MiL, HiL infrastructure (deployment plan, SH_ORO_GR_4, SH_ORO_GR_7). 4. The developer D implements the new req and checks it into the Adeptness toolchain 5. If the validation plan is ready, the Adeptness toolchain executes it: First it deploys the test artifacts and SUT as defined in 3.3, then injects the test inputs as defined in 3.1, and finally monitors the outputs as defined in 3.2 6. If the validation plan passes, the software version is labelled at HiL level.	1. The analyst can include the new req into Adeptness 2. The validator is notified about the new rew and can define the validation plan, including deployment and monitoring plans that support the validation 3. The developer is notified about the new req, develops the code, and can check it in the Adeptness toolchain. 4. Once the development is completed by the developer, the system automatically executes the validation plan, deploying the test artifacts, injecting the inputs, and monitoring the outputs 5. The test result is PASS and the software version is labelled at HiL level

3.1.2.2 ORO_SHR_Test_2

Author: ORO

Status: Proposed

Validates Requirements: TBD

Validation Method: Test

Test Data:

Preconditions	Description	Expected Results
HiL infrastructure ready and connected to Adeptness toolchain	Registration, development, and validation of a new requirement at HiL level: "support for automatic destination calls injection though the access control system" 1. The analyst introduces a new req in the Adeptness toolchain: the destination call product (EasyFlow) must manage direct calls introduced through the access control system, i.e. the user presents an access card in the ISD and the system injects the destination call associated to that user. 2. The Adeptness toolchain notifies developer D and validator V about the new req 3. The validator V designs and develops the validation plan for the requirement and registers it in the Adeptness toolchain. Such validation plan is intended for the HiL level and includes the deployment and monitoring plans needed to perform the validation. More specifically, it follows these steps: 3.1. Define the passenger list that must be injected into the ethernet network to emulate the card passing. It will contain one passenger going from floor 2 to floor 5 (test input). For this, the validator will use the tool described in SH_ORO_HIL_VAL_6) 3.2. Define the CAN messages that are expected in a predefined time window because of the previous step (monitoring plan). For this, the tool described in SH_ORO_HIL_MON_2 shall be used. These messages include the car allocation and the user notification.	1. The validator is notified about a new version and can define the validation plan, including deployment and monitoring plans that support the validation. The Plan is specified for SIL, MIL and HIL 2. The validator can execute the validation plan at MIL, and at HIL artifacts, injecting the inputs and monitoring the outputs 3. The VP is successful, and the analyst is notified 4. The analyst defines the Deployment Plan

	3.3. Define the deployment of the different test artifacts and SUT into the HiL infrastructure (deployment plan, SH_ORO_GR_4, SH_ORO_GR_7). 3.4. Configure the testExecutor 4. When a validation plan is ready, the Adeptness toolchain executes it: First it deploys the test artifacts and SUT as defined in 3.3, then executes the validation plan at MiL i, if successful, executes the validation plan at HiL. Otherwise, notifies the developer and validator. The result of the validation plan at HiL is notified to the analyst 5.If the validation plan is successful the analyst defines the deployment plan for the new version including the validation artefacts for validation 6. The version is deployed and the continuous validation microservice is executed	

3.2 System Verification Tests

The purpose of this section is to describe how each Adeptness system level requirement is going to be verified.

3.2.1 UES_SYS_Test_1

Author: UES

Status: Proposed

Verifies Requirement:

SYS_MON_11

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Monitorization subsystem is not running	Monitorization subsystem execution 1. Deploy CPSoS release in Virtual Platform 2. Monitorization subsystem start request is send	Monitorization subsystem is running and collecting data, which is stored in MQTT

3.2.2 UES_SYS_Test_2

Author: UES

Status: Proposed

Verifies Requirement:

SYS_MON_11

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Monitorization subsystem is not running	Monitorization subsystem execution 1. Deploy CPSoS release in Real Platform 2. Monitorization subsystem start request is send	Monitorization subsystem is running and collecting data, wich is stored in MQTT

3.2.3 UES_SYS_Test_3

Author: UES

Status: Proposed

Verifies Requirement:

SYS_VAL_20

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Validation subsystem is not running	Validation subsystem execution 1. Deploy CPSoS release in Virtual Platform 2. Validation subsystem start request is send	Validation subsystem is not running

3.2.4 UES_SYS_Test_4

Author: UES

Status: Proposed

Verifies Requirement:

SYS_VAL_20

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Validation subsystem is not running	Validation subsystem execution 1. Deploy CPSoS release in Real Platform 2. Validation subsystem start request is send	Validation subsystem is not running

3.2.5 UES_SYS_Test_5

Author: UES

Status: Proposed

Verifies Requirement:

SYS_MON_12

SYS_VAL_21

SYS_VAL_22

SYS_VAL_23

SYS_TRC_5

SYS_TRC_6

SYS_TRC_7

SYS_TRC_8

SYS_NF_Reg_1

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Oracle, Monitorization and Validation subsystem are running	Different microservices data exchange: 1. Test Executor runs a test 2. Verdict and test execution data is published in MQTT 3. Monitorization subsystem collects operational data while test execution and stores it in MQTT 3. Validation subsystem stores test execution validation data in MQTT 4. Traceability subsystem gets all the test execution related data from MQTT and generates Test Validation Report. 5. Traceability Microservice updates linked lifecycle artifacts in the corresponding lifecycle tool via OSLC 6. Developer and Validator are notified	Validation and Monitorization data are collected and a verdict is obtained. Test Report is sent to linked lifecycle artifacts in corresponding tools and a notification is received by the developer and the validator

3.2.6 MGEP_SYS_Test_6

Author: MGEP

Status: Proposed

Verifies Requirement:

SYS_VAL_*

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Validation artefacts in the repository monitoring and deployment microservices running	Deployment, Monitorization and Validation data exchange: 1- A validation plan is executed 2. The deployment microservice deploys validation artifacts 3. The monitorisation microservice is configured 2- Monitorization subsystem collects test data and stores it in MQTT 3- Validation subsystem gets monitorization data from MQTT	Artefacts are deployed Monitoring provides the expected data Validation microservice provides the expected result

3.2.7 MGEP_SYS_Test_7

Author: MGEP

Status: Proposed

Verifies Requirement:

SYS_REC_*

SYS_VAL_*

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Validation microservice is running	Validation and recovery data exchange 1. A validation microservice provides a failing 2. RM checks the event with the configured rules and selects the output (recovery action to be launched) 3. RM send the message that will activate a new validation plan 4. The continuous validation microservice executes the new validation plan	The new validation plan is executed

3.2.8 UES_SYS_Test_8

Author: UES

Status: Proposed

Verifies Requirement:

SYS_TRC_2

SYS_TRC_3

SYS_TRC_4

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
TBD	[TBD] DSL - OSLC integration	TBD

3.2.9 UES_SYS_Test_9

Author: UES

Status: Proposed

Verifies Requirement:

SYS_NF_4

Verification Method: Not Set

Test Data:

Preconditions	Description	Expected Results
TBD	Traceability between different lifecycle management tools using OSLC standard	TBD

3.2.10 UES_SYS_Test_10

Author: UES

Status: Proposed

Verifies Requirement:

SYS_NF_5

Verification Method: Not Set

Test Data:

Preconditions	Description	Expected Results
TBD	Adeptness workflow integration	TBD

3.2.11 UES_SYS_Test_11

Author: UES

Status: Proposed

Verifies Requirement:

SYS_NF_6

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Lifecycle management and development methods are defined and documented	Adeptness lifecycle management and development methods integration	Lifecycle management and development methods are integrated in Adeptness toolchain

3.2.12 UES_SYS_Test_12

Author: UES

Status: Proposed

Verifies Requirement:

SYS_NF_7

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Tool OSLC services are shutdown	Adeptness toolchain root service document: 1. Start tool OSLC Services 2. Enter corresponding public root services URL	Tool specific root services document is shown in the browser

3.2.13 UES_ SYS_Test_13

Author: UES

Status: Proposed

Verifies Requirement:

SYS_NF_8

SYS_NF_9

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Tool OSLC services are shutdown	Adeptness toolchain authentication: 1. Start linked tool OSLC Services 2. Enter correct credentials to access associated tool project 3. Associated project area artifact containers are shown	Access to granted

3.2.14 UES_SYS_Test_14

Author: UES

Status: Proposed

Verifies Requirement:

SYS_NF_8

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Tool OSLC services are shutdown	Adeptness toolchain authentication: 1. Start linked tool OSLC Services 2. Enter wrong credentials to access associated tool project	Access to is not granted

3.2.15 UES_SYS_Test_15

Author: UES

Status: Proposed

Verifies Requirement:

SYS_NF_10

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Customized Adeptness tool dialogs are implemented	Adeptness toolchain OSLC dialogs customization: TBD in accordance to the design document	Dialogs are customized according to the design document

3.2.16 UES_SYS_Test_16

Author: UES

Status: Proposed

Verifies Requirement:

SYS_NF_11

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Each Adeptness tool has defined a TRS provider	Adeptness toolchain OSLC dialogs customization: TBD in accordance to the TRS provider definition	TBD

3.3 STO Verification Tests

3.3.1 STO 1.1 Microservice Based Architecture and Interfaces Tests

3.3.1.1 IKL_STO_1_1_Test_1

Author: IKL

Status: Proposed

Verifies Requirements:

SYS_CICD_1

SYS_CICD_2

STO1.1_1

STO1.1_2

STO1.1_3

STO1.1_4

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
TBD	1. A new requirement is introduced with the adeptness toolchain by the Manager 2. The developer team generates a new version of artifact S1. 3- The validation team generates a new oracle O1 but no deployment plan. 4- The manager tags the new version of S1 as ready to deploy/validate	Developer team receives an email to begin implementation. Validation team receives an email to begin validation. S1 and O1 binaries are in the repository.

3.3.1.2 IKL_STO_1_1_Test_2

Author: IKL

Status: Proposed

Verifies Requirements:

SYS_DEP_5

SYS_DEP_2

SYS_DEP_3

STO1.1_5

STO1.1_6

STO1.1_39

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
TBD	1. A new requirement is introduced with the adeptness toolchain by the Manager 2. The developer team generates a new version of artifact S1. 3- The validation team generates a new oracle O1 and a new development plan to deploy S1 & O1 in target X 4- The binaries are manually generated in repo. 5.The development mechanism is manually launched. 6.With a tested RESTful client invoke the development API to load the development plan. 7.With a tested RESTful client invoke the development API to start the development plan. 8.With a tested RESTful client invoke the development API to ask for the state of the execution plan.	S1 and O1 are in target X. Status of the development mechanism last execution shall be "Execution Succeeded"

Comments: Integration Deployment mechanism & deploy agent

3.3.1.3 SRL_STO_1_1_Test_3

Author: SRL

Status: Proposed

Verifies Requirements:

STO1.1_26

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
TBD	Test that the uncertainty detection microservice shall receive test data from the validation microservice.	The services are integrated.

Comments

Data transmission happening

3.3.1.4 SRL_STO_1_1_Test_4

Author: SRL

Status: Proposed

Verifies Requirements:

STO1.1_21

STO1.1_25

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
TBD	Test that the microservice can determine an unforeseen situation based on the live data provided by the monitoring services in operation time.	The services are integrated.

Comments

Data transmission happening

3.3.1.5 SRL_STO_1_1_Test_5

Author: SRL

Status: Proposed

Verifies Requirements:

STO1.1_27

STO1.1_29

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
	Test if the uncertainty situation is detected whether to be safe or not based on validation verdicts from validation microservice.	-uncertainty detection service started. -detection heuristics applied

Comments

Successful detection of uncertainty using heuristics.

3.3.1.6 SRL_STO_1_1_Test_6

Author: SRL

Status: Proposed

Verifies Requirements:

STO1.1_27

STO1.1_28

STO1.1_29

STO1.1_32

Verification Method: Demostration

Test Data:

Preconditions	Description	Expected Results
TBD	Configure and invoke recovery microservices to take recovery actions upon detection of failure.	-uncertainty detection service started. - detection heuristics applied - detected uncertain situation

Comments

Success response from recovery microservice as started

3.3.1.7 IKL_STO_1_1_Test_7

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.1_7

STO1.1_8

STO1.1_9

STO1.1_10

STO1.1_11

STO1.1_12

STO1.1_17

STO1.1_18

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Service compiled and ready for launch.	1. Launch each one of the services manually (monitoring/validation/recovery/uncertainty/testgeneration/traceability). 2. With a tested RESTful client ask for the status. 3. With a tested RESTful client start the service. 4. With a tested RESTful client ask for the status. 5. With a tested RESTful client stop the service. 6. With a tested RESTful client ask for the status. 7. Kill the service. 8. With a tested RESTful client ask for the status.	After step 2 HTTP response should be 200/OK and status should be READY. After step 4 HTTP response should be 200/OK and status should be OPERATIONAL. After step 6 HTTP response should be 404/NOT_FOUND.

3.3.1.8 IKL_STO_1_1_Test_8

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.1_14,

STO1.1_15,

STO1.1_16,

STO1.1_17,

STO1.1_20,

STO1.1_23,

STO1.1_24

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Services compiled and ready for launch. Config file for M1 prepared with observations needed by O1.	1.Launch a monitoring service M1 manually. 2.Launch a validation service (oracle O1) manually. 3.With a tested RESTful client start the service M1(it begins to publish logical data according to a config file). 4.With a tested RESTful client start the service O1 5.With a tested MQTT subscriber, subscribe to the topic publish by the O1.	MQTT client receives data from O1 with the expected result.

Comments

No SUT required.

M1 will simulate the expected observations.

Integration Monitoring-Validation

3.3.1.9 IKL_STO_1_1_Test_9

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.1_14

STO1.1_15

STO1.1_17

STO1.1_20

STO1.1_23

STO1.1_24

STO1.1_35

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Services compiled and ready for launch.	1.Launch a monitoring service M1 manually. 2.Launch a validation service (oracle O1) manually. 3.With a tested RESTful client start the service M1. 4. With a tested RESTful client configure M1 as expected by O1 5.With a tested RESTful client start the service O1. 6.With a tested MQTT subscriber, subscribe to the topic publish by the O1.	In the U1 log traces appears data received from M1

Comments

No SUT required.

M1 will simulate the expected observations.

Integration Monitoring-Validation

3.3.1.10 IKL_STO_1_1_Test_10

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.1_14,

STO1.1_15,

STO1.1_16,

STO1.1_17,

STO1.1_21,

STO1.1_25

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Services compiled and ready for launch. Config file for M1 prepared with observations needed by U1.	1.Launch a monitoring service M1 manually. 2.Launch an uncertainty service (U1) manually. 3.With a tested RESTful client start the service M1 (it begins to publish logical data according to a config file). 4.With a tested RESTful client start the service U1 5.Collect log traces from U1In the R1 log traces appear data received from M1	In the U1 log traces appears data received from M1

Comments

No SUT required.

M1 will simulate the expected observations.

Integration Monitoring-Uncertainty

3.3.1.11 IKL_STO_1_1_Test_11

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.1_14

STO1.1_15

STO1.1_16

STO1.1_17

STO1.1_34

STO1.1_38

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Services compiled and ready for launch. Config file for M1 prepared with observations needed by R1.	1.Launch a monitoring service M1 manually. 2.Launch a recovery service (R1) manually. 3.With a tested RESTful client start the service M1 (it begins to publish logical data according to a config file). 4.With a tested RESTful client start the service R1 5.Collect log traces from R1In the R1 log traces appear data received from M1	In the R1 log traces appears data received from M1

Comments

No SUT required.

M1 will simulate the expected observations.

Integration Monitoring-Recovery

3.3.1.12 IKL_STO_1_1_Test_12

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.1_14

STO1.1_15

STO1.1_16

STO1.1_17

STO1.1_20

STO1.1_23

STO1.1_24

STO1.1_30

STO1.1_34

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Services compiled and ready for launch. Config file for M1 prepared with observations needed by O1.	1.Launch a monitoring service M1 manually. 2.Launch a validation service (oracle O1) manually. 3.Launch a recovery service (R1) manually 4.With a tested RESTful client start the service M1 (it begins to publish logical data according to a config file). 5.With a tested RESTful client start the service R1 6.With a tested RESTful client start the service O1 5.Collect log traces from R1	In the R1 log traces appears data received from M1 y O1

Comments

No SUT required.

M1 will simulate the expected observations.

Integration Monitoring-Validation-Recovery

3.3.1.13 IKL_STO_1_1_Test_13

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.1_14

STO1.1_15

STO1.1_16

STO1.1_17

STO1.1_21

STO1.1_25

STO1.1_31

STO1.1_34

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Services compiled and ready for launch. Config file for M1 prepared with observations needed by U1.	1.Launch a monitoring service M1 manually. 2.Launch an uncertainty service (U1) manually. 3.Launch a recovery service (R1) manually 4.With a tested RESTful client start the service M1 (it begins to publish logical data according to a config file). 5.With a tested RESTful client start the service R1 6.With a tested RESTful client start the service U1 5.Collect log traces from R1In the R1 log traces appear data received from M1 y U1In the R1 log traces appear data received from M1 y U1	In the R1 log traces appears data received from M1 y U1

Comments

No SUT required.

M1 will simulate the expected observations.

Integration Monitoring-Uncertainty-Recovery

3.3.2 STO 1.2 Deployment Modelling Tests

3.3.2.1 IKL_STO_1_2_Test_1

Author: IKL

Status: Proposed

Verifies Requirements:

SYS_DEP_1

STO1.2_1

STO1.2_2

STO1.2_3

STO1.2_4

STO1.2_5

STO1.2_6

STO1.2_7

Verification Method: Analysis

Test Data:

Preconditions	Description	Expected Results
Deployment plan schema available	Definition of a deployment plan: The validator, following the DSL schema provided by the Adeptness toolchain, defines the following: - Hardware infrastructure nodes - Deployable elements: at least a SUT and test inputs - Allocation of deployable elements to HW nodes - Triggers of the deployment - Action to perform for each deployable element (deploy, deploy and launch) - Event notifications associated to previous actions.	The validator can define all the required aspects following the schema

3.3.3 STO 1.3 Deployment Orchestration Tests

3.3.3.1 IKL_STO_1_3_Test_1

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.3_19

STO1.3_1

STO1.3_10

STO1.3_4

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Deployment plan already defined: one step with a deploy and launch action of a binary software artifact in a specific target. Software artifact is in repository.	1. Implement an executable software component. 2. Move the executable component S1 to the repository. 3. Define a deployment plan (DSL) to deploy and lunch S1 in a target X. 4. Use a tested asynchronous application (e.g. MQTT client) to publish an event with the deployment plan. 5. Use a tested asynchronous application (e.g. MQTT client) to publish an event to start the deployment. 6. In the target X ask the operating system whether S1 is installed and running.	S1 is deployed and running in the target X.

Comments

Using O.S console commands in target X.

3.3.3.2 IKL_STO_1_3_Test_2

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.3_19

STO1.3_1

STO1.3_10

STO1.3_4

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Deployment plan already defined: one step with a deploy and launch action of a binary software artifact in a specific target. Software artifact is in repository.	1. Implement an executable software component. 2. Move the executable component S1 to the repository. 3. Define a deployment plan (DSL) to deploy and lunch S1 in a target X. 4. Use a tested synchronous application (e.g. Rest client) to send the deployment plan. 5. Use a tested synchronous application (e.g. Rest client) to start the deployment. 6. In the target X ask the operating system whether S1 is installed and running.	S1 is deployed and running in the target X.

Comments

Using O.S console commands in target X.

3.3.3.3 IKL_STO_1_3_Test_3

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.3_8

STO1.3_1

STO1.3_2

STO1.3_10

STO1.3_19

STO1.3_3

STO1.3_18

STO1.3_4

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Deployment plan already defined: one step with a deploy and launch action of a binary software artifact in a specific target. Software artifact is in repository. External client subscribed to notifications	1. Implement an executable software component. 2. Move the executable component S1 to the repository. 3. Define a deployment plan (DSL) to deploy and lunch S1 in a target X. 4. Use a tested synchronous application (e.g. Rest client) to send the deployment plan. 5. Use a tested synchronous application (e.g. Rest client) to start the deployment. 6. Verify that the deployment system has notified the previous steps	The notifications sent by the deployment subsystem are received in an external system by subscription

3.3.3.4 IKL_STO_1_3_Test_4

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.3_9

STO1.3_1

STO1.3_10

STO1.3_19

STO1.3_4

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Deployment plan already defined: one step with a deploy action of a binary software artifact in a specific target. Software artifact is in repository.	1. Implement an executable software component. 2. Move the executable component S1 to the repository. 3. Define a deployment plan (DSL) to deploy S1 in a target X. 4. Use a tested synchronous application (e.g. Rest client) to send the deployment plan. 5. Use a tested synchronous application (e.g. Rest client) to start the deployment. 6. In the target X ask the operating system whether S1 is installed.	Only the deployment is performed. The deployed sw component is deployed but not launched.

3.3.3.5 IKL_STO_1_3_Test_5

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.3_9

STO1.3_1

STO1.3_10

STO1.3_19

STO1.3_4

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Deployment plan already defined: one step with a deploy and launch action of a binary software artifact in a specific target. Software artifact is in repository.	1. Implement an executable software component. 2. Move the executable component S1 to the repository. 3. Define a deployment plan (DSL) to deploy and launch S1 in a target X. 4. Use a tested synchronous application (e.g. Rest client) to send the deployment plan. 5. Use a tested synchronous application (e.g. Rest client) to start the deployment. 6. In the target X ask the operating system whether S1 is installed and running	Both the deployment and launching are performed. The deployed sw component is deployed and launched.

3.3.3.6 IKL_STO_1_3_Test_6

Author: IKL

Status: Proposed

Verifies Requirements:

STO1.3_19

STO1.3_1

STO1.3_10

STO1.3_19

STO1.3_6

STO1.3_7

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Deployment plan already defined: step1- deploy and launch action of M1 artifact in a specific target X step2- deploy and launch action of O1 artifact in a specific target X Software artifacts is in repository.	1. Implement a monitoring software component M1 and an oracle software component O1. 2. Move the executable components M1 & O1 to the repository. 3. Define a deployment plan (DSL) to deploy and lunch M1 & O1 in a target X. 4. Use a tested asynchronous application (e.g. MQTT client) to publish an event with the deployment plan. 5. Use a tested asynchronous application (e.g. MQTT client) to publish an event to start the deployment. 6. In the target X ask the operating system whether M1 & O1 are installed and running.	M1 & O1 are deployed and running in the target X.

Comments

Using O.S console commands in target X.

3.3.3.7 TUW_STO_1_3_Test_7

Author: TUW

Status: Proposed

Verifies Requirements:

STO1.3_20

STO1.3_22

STO1.3_24

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Target platforms A, B are included in the toolchain.	1. Start a project using ADEPTNESS toolchain. 2. Configure deployment subsystem. 3. Implement software artefacts S1, S2 and store to the repository. 4a. Develop a deployment plan P1 for target A based on S1. 4a. Develop a deployment plan P2 for target B based on S2. 5. Execute deployment both plans P1, P2.	Seamless deployment of both platforms.

Comments

Software abstraction layer for deployment or HW extension gateway.

3.3.3.8 TUW_STO_1_3_Test_8

Author: TUW

Status: Proposed

Verifies Requirements:

STO1.3_25

STO1.3_23

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Communication channel from the deployment subsystem to target.	1. Start a project using ADEPTNESS toolchain. 2. Configure deployment subsystem. 3. Create deployment plan P1 for target A. 4. Generate secure access credentials for target A. 5. Establish secure connection between deployment subsystem and target A.	Secure channel for bi-directional communication.

Comments

Secure connection can be established on VPN level or directly from CI/CD system using SSH/SSL ad-hoc connection.

3.3.3.9 TUW_STO_1_3_Test_9

Author: TUW

Status: Proposed

Verifies Requirements:

STO1.3_25

STO1.3_23

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Deployment plan, SW artefact, communication channel	1. Create an update X on software artefact S1. 2. Start deployment of S1.X to target A while target still in operation. 3. Deployment system sends an event to target A to bring the system in safe state. 4. Update is deployed.	Runtime updates to a target.

3.3.3.10 TUW_STO_1_3_Test_10

Author: TUW

Status: Proposed

Verifies Requirements:

STO1.3_27

STO1.3_25

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Security services, communication channels	1. ADEPTNESS toolchain project created. 2. Deployment plan P was implemented. 3. Secure credentials for secure communication and authentication service are generated. 4. Deployment service configures targets. 5. Software artefact A is built. 6. Artefact A is signed using authentication credentials and stored in repository. 7. Artefact A is deployed from repository.	Authentication for SW artefacts.

3.3.4 STO 2.1 monitoring Microservice Tests

3.3.4.1 IKL_STO_2_1_Test_1

Author: IKL

Status: Proposed

Verifies Requirements:

STO2.1_1

STO2.1_10

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Monitoring microservice is running in a hardware connected to a CAN bus	Compliance with the interface defined to plug-in different physical network connectors. Steps to reproduce: 1. Configure the monitoring service to plug in the CAN connector through the predefined interface (SUT) 2. Invoke the interface to configure it to filter a specific CAN frame 3. Start the monitoring service	-The monitoring service can configure the connector through the defined API, and receives the data according to the configuration -The monitoring data follows the json format designed

3.3.4.2 IKL_STO_2_1_Test_2

Author: IKL

Status: Proposed

Verifies Requirements:

STO2.1_1

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Monitoring microservice is running in a hardware connected to Ethernet	Compliance with the interface defined to plug-in different physical network connectors. Steps to reproduce: 1. Configure the monitoring service to plug in the Ethernet connector through the predefined interface (SUT) 2. Invoke the interface to configure it to filter a specific Ethernet frame 3. Start the monitoring service	The monitoring service can configure the connector through the defined API, and receives the data according to the configuration

3.3.4.3 IKL_STO_2_1_Test_3

Author: IKL

Status: Proposed

Verifies Requirements:

STO2.1_2

STO2.1_3

Verification Method: Comparison

Test Data:

Preconditions	Description	Expected Results
Monitoring microservice is running in a hardware connected to a CAN bus	Compliance with the interface defined to plug-in stakeholder specific data parsers for a CAN bus 1. Configure the monitoring service to parse a data protocol of stakeholder A 2. Start the monitoring service 3. Inject a CAN frame (with a trusted client) that follows that protocol (e.g. logical variable X corresponds to bits 7…10 of the CAN frame	The monitoring microservice correctly receives the value of the variable X

3.3.4.4 IKL_STO_2_1_Test_4

Author: IKL

Status: Proposed

Verifies Requirements:

STO2.1_2

STO2.1_3

Verification Method: Comparison

Test Data:

Preconditions	Description	Expected Results
Monitoring microservice is running in a hardware connected to Ethernet	Compliance with the interface defined to plug-in stakeholder specific data parsers for a json structure over Ethernet 1. Configure the monitoring service to parse the data protocol of stakeholder A 2. Start the monitoring service 3. Inject a json frame (with a e.g. mqtt client) that follows that protocol (e.g. logical variable X corresponds to field F of the json	The monitoring microservice correctly receives the value of the variable X

3.3.4.5 IKL_STO_2_1_Test_5

Author: IKL

Status: Proposed

Verifies Requirements:

STO2.1_4

Verification Method: Comparison

Test Data:

Preconditions	Description	Expected Results
Monitoring service started and configured to extract the reference logic data.	Subscription mechanism: A well-known CAN frame with a payload based on a stakeholder specific data protocol is injected (by a trusted tool) in the CAN bus. With a trusted client a subscription to the data provided by the monitoring service is made. The data received in the subscription is evaluated.	data received = reference logic data

3.3.4.6 IKL_STO_2_1_Test_6

Author: IKL

Status: Proposed

Verifies Requirements:

STO2.1_5

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Monitoring service started	The API of the monitoring microservice that provides health status is invoked through a generic client	The returned value is OK

3.3.4.7 IKL_STO_2_1_Test_7

Author: IKL

Status: Proposed

Verifies Requirements:

STO2.1_5

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Monitoring service stopped	The API of the monitoring microservice that provides health status is invoked through a generic client	The returned value is NOK

3.3.4.8 IKL_STO_2_1_Test_8

Author: IKL

Status: Proposed

Verifies Requirements:

STO2.1_6

STO2.1_7

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
-Monitoring service started -Trusted client	Test the configurability of the monitoring service through an API: 1. Configure the monitoring service through the API to monitor variable V on bus B with protocol P, at a specific sampling rate (samples per second) 2. Start the monitoring service 3. Inject an appropriate CAN frame (with a trusted client) that follows the desired protocol	The service gets the data from the bus and notifies its value at the specified rate

3.3.4.9 IKL_STO_2_1_Test_9

Author: IKL

Status: Proposed

Verifies Requirements:

STO2.1_6

STO2.1_8

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
-Monitoring service stopped -Trusted client	Test the configurability of the monitoring service though a configuration file: 1. Write the configuration file to monitor variable V on bus B with protocol P, at a specific sampling rate (samples per second) 2. Launch the monitoring service 3. Start monitoring 4. Inject an appropriate CAN frame (with a trusted client) that follows the desired protocol	- The service gets its configuration from the config file - The service gets data from the bus and notifies its value at the specified rate

3.3.4.10 IKL_STO_2_1_Test_10

Author: IKL

Status: Proposed

Verifies Requirements:

STO2.1_6,
STO2.1_8,
STO2.1_13

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
-Monitoring service stopped -Trusted client	Test the configurability of the monitoring service though a configuration file with rules: 1. Write the configuration file to monitor variable V on bus B with protocol P, at a specific sampling rate (samples per second) and set rule to provide a mean value of a time interval. 2. Launch the monitoring service 3. Start monitoring 4. Inject an appropriate CAN frame (with a trusted client) that follows the desired protocol	- The service gets its configuration from the config file - The service gets data from the bus and notifies the mean value (of the specified time interval) at the specified rate

3.3.4.11 IKL_STO_2_1_Test_11

Author: IKL

Status: Proposed

Verifies Requirements:

STO2.1_9

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
-Source code of the SUT available -Monitoring service started -Trusted client	Test the application monitoring capability of the service: 1. Instrument the source code of the SUT 2. Configure the monitoring service to monitor the variables instrumented in the code 3. Start monitoring 4. Start the SUT	- The service gets data (traces) through Ethernet, coming from the instrumented code, and notifies its value at the specified rate

3.3.4.12 IKL_STO_2_1_Test_12

Author: IKL

Status: Proposed

Verifies Requirements:

STO2.1_11

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
-Monitoring service started -Trusted client	Test the persistence functionality of the monitoring microservice: 1. Configure the monitoring service through the API to monitor variable V on bus B with protocol P, at a specific sampling rate (samples per second) 2. Configure the service to persist the monitoring data 3. Start the monitoring service 4. Inject an appropriate CAN frame (with a trusted client) that follows the desired protocol"	The service gets the data from the bus and stores it

3.3.4.13 TUW_STO_2_1_Test_13

Author: TUW

Status: Proposed

Verifies Requirements:

STO2.1_14

STO2.1_5

STO2.1_15

STO2.1_18

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
- API channel for exporting data	The monitoring data will be exported from a software or hardware artefact and relayed using API, that can be consumed by a monitoring service locally or remotely depending on the use case requirements. This allows monitoring separation in two components behaviour collector that is close to the monitored artefact and behaviour analyser that is observing and aggregating data.	Ability to perform monitoring locally or over the internet.

3.3.4.14 TUW_STO_2_1_Test_14

Author: TUW

Status: Proposed

Verifies Requirements:

STO2.1_14

STO2.1_16

STO2.1_15

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
- Monitoring service configuration	The monitoring service can be configured to collect or collect and analyse. In the collect and analyse mode there will be two possibilities soft real-time analysis and completely asynchronous analysis. Soft real-time analysis will collect, analyse, and store data. While asynchronous analysis will collect, store, and analyse subsequently depending on the type of monitor or oracle.	Different modes of operation depending on the needs of a use case

3.3.4.15 TUW_STO_2_1_Test_15

Author: TUW

Status: Proposed

Verifies Requirements:

STO2.1_17

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
- Monitoring requirements - STL to monitoring configuration generator	1. Define requirements from a runtime monitor (e.g., signal or property) in temporal logic 2. Run monitor configuration generator from STL to get configuration C 3. Configure monitoring service using the configuration C 4. Run monitoring service and verify results.	Automatic translation from temporal logic to monitoring configuration

3.3.4.16 TUW_STO_2_1_Test_16

Author: TUW

Status: Proposed

Verifies Requirements:

STO2.2_1

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
- Monitoring service running - Scenarios for different performance consumption - Performance consumption monitor	1. Start monitoring service on a variety of artefacts 2. Set resource consumption limit per data collector 3. Observe consumption 4. Analyse results	Ability to deploy monitoring service on platforms with limited resources

3.3.5 STO 2.2 Continuous Validation Microservice Tests

3.3.5.1 MGEP_STO_2_2_Test_1

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.2_6

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
UDP frames injector is running connected to the system Monitoring microservice is running in a hardware connected to Ethernet UDP frame is defined	Injection of UDP frames in HiL will be checked. Steps to reproduce: 1. Define a UDP frame with the information of a passenger 2.Monitor the Ethernet bus 2. Run the UDP injector 3. Inject the UDP frame	The injected UDP frame is monitored

3.3.5.2 MGEP_STO_2_2_Test_2

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.2_6

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
CAN frames injector is running Monitoring microservice is running connected to the CAN bus CAN frame is defined	Injection of CAN frames in HiL will be checked. Steps to reproduce: 1. Define a CAN frame with the information of a new landing call 2. Monitor the Bus CAN 3. Run the CAN injector 4. Inject the CAN frame	The injected CAN frame is monitored

3.3.5.3 MGEP_STO_2_2_Test_3

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.2_7

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Tool to transform the passenger to CAN frames is running Monitoring microservice is running connected to the CAN bus Passenger is defined	Injection of passengers in HiL will be checked. Steps to reproduce: 1. Define a passenger 2. Monitor the bus CAN 3. Run the tool to transform the passenger to CAN frames 4. Inject the passenger	The information of the CAN frame is the information of the passenger

3.3.5.4 MGEP_STO_2_2_Test_4

Author: MGEP

Status:

Verifies Requirements:

STO2.2_6

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
The configuration is defined	Configuration of the controllers. Steps to reproduce: 1. Define a configuration for a set of lifts: number of floors, number of lifts 2. Monitor the CAN bus 3. Order the configuration	Controllers are configured according to the defined configuration

3.3.5.5 MGEP_STO_2_2_Test_5

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.2_9

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Monitoring microservice is running testcase is running	Check the triggering of a testcase by identifying a traffic profile 1. Configure a triggering criterion based in traffic profiles 2. Monitor the Up Peak detection 3. Run the testcase 4. Check	When the monitoring service detects the Up Peak, the evaluation of the testcase starts

3.3.5.6 MGEP_STO_2_2_Test_6

Author: MGEP

Status:

Verifies Requirements:

STO2.2_9

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Monitoring microservice is running testcase is running	Check the triggering of a testcase by identifying a CAN frame (a landing call) 1. Configure a triggering criterion based in sequence detection 2. Monitor CAN frames 3. Run the testcase 4. Check	When the monitoring service detects the CAN frame for the landing call and, the evaluation of the testcase starts

3.3.5.7 MGEP_STO_2_2_Test_7

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.2_8

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Interfaces of criteria checkers at SiL, HiL and operation are available	Interfaces of criteria checkers will be reviewed to check platform independence	There is no platform dependence in the interface

3.3.5.8 MGEP_STO_2_2_Test_8

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.2_4

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Elevate is deployed The call injector and analyser at HiL are deployed	Check that the test Executor launches a simulation in SiL and in HiL: 1. Configure the installation in SiL/HiL 2. Execute a test case that injects a list of passengers 3. Check	Simulation in Elevate finishes Simulation in the hybrid environment finishes

3.3.5.9 MGEP_STO_2_2_Test_9

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.2_2

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
The deployment plan for a validation plan is defined	Check that the test Executor orders the deployment of simulation artifacts 1. Define the deployment plan for a validation plan 2. Execute 3. Check that the elements are deployed correctly	Elevate, files are deployed

3.3.5.10 MGEP_STO_2_2_Test_10

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.2_1

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
The monitoring plan is defined Monitors are available and running	Check that the test Executor order the configuration of monitors 1. Define the monitored plan for a validation plan, monitor the AWT 2. Execute 3. Check that the monitors provide the AWT	Monitors provide the expected information

3.3.5.11 MGEP_STO_2_2_Test_11

Author: MGEP

Status:

Verifies Requirements:

STO2.2_5

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Rules are defined	Check that the test arbiter provides the right verdict: 1. Set rules 2. Set test cases verdicts 3. Execute 4. Check that the test arbiter provides the right verdict	The test arbiter provides the right verdict

3.3.5.12 MGEP_STO_2_2_Test_12

Author: MGEP

Status: Proposed

Verifies Requirements:

STO_2.2_11

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Test oracles and their interface are defined. Deployment infrastructure is ready	Check that the different types of test oracles are provided: 1. For each type of test oracle, design a test oracle. 2. Deploy the test oracle in SiL and HiL 3. Execute a test case and check that the test oracle is behaving as expected	Oracles behaving as expected

3.3.5.13 MGEP_STO_2_2_Test_13

Author: MGEP

Status: Proposed

Verifies Requirements:

STO_2.2_12

STO2.2_18

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
HiL infrastructure is ready	Check that critical oracles are executed in soft real-time in HiL and operation: 1. Develop a critical test oracle. 2. Deploy the test oracle in HiL 3. Execute stress test case to the oracle 4. Check that the oracle is executed in soft real-time	Oracles running in soft real-time

3.3.5.14 MGEP_STO_2_2_Test_14

Author: MGEP

Status: Proposed

Verifies Requirements:

STO_2.2_13

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
All the types of test oracles are ready	Propose an empirical validation with mutation testing to check that oracle handle 95% of precision and recall	Test oracles have a precision and recall higher than 95 %

3.3.6 STO 2.3 Unforeseen Situation Detection Microservice Tests

The purpose of this section is to enlist all tests description to enable validation on the scientific requirements of the Unforeseen Situation Detection Microservice should satisfy to support the ADEPTNESS toolchain in determining the uncertainty during the operation of CPSoS. Successful test outcome shall verify the Unforeseen Situation Detection Microservice meets the enlisted functional and non-functional list of requirements. These tests are defined based on the technology used by stakeholders of the two use-cases ADEPTNESS toolchains (e.g., target hardware, type of deployment, communication protocols, security, etc.).

3.3.6.1 Definitions, Acronyms and Abbreviations

Test stimulus: Test data sent to the SUT to control it and to make assessments about the SUT when receiving the SUT reactions to these stimuli.

Evaluation criteria: Evaluation criteria are the interventions to be used to assess the development of the microservice, i.e., accuracy, efficiency, and unbiasedness.

Operation uptime: The period microservice was in operation starting from the launch time until the time it stopped.

Digital Twin: Abstraction of CPSoS in operation within a microservice satisfying a specific use-case scenario of ADEPTNESS toolchain.

3.3.6.2 Tests

3.3.6.2.1 SRL_STO_2_3_Test_1

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_1

STO2.3_9

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
At least one algorithm is implemented and microservices are deployed at operation	Test that uncertainty detection microservice finds uncertain situations within a CPSoS in operation.	The toolchain is functioning with success response message.

Compliance with the interface defined to plug-in different microservices and CPSoS elements. Steps to reproduce:

1. Configure the uncertainty detection service to establish connection with other microservices through the predefined interface (SUT)

2. Invoke the interface to configure within a common request/response-based communication mechanism (e.g., HTTP, RESR) through a common DSL tool (e.g. XML, JSON) used by all the microservices and elements of CPSoS.

3. Start the uncertainty detection service

3.3.6.2.2 SRL_STO_2_3_Test_2

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_2

STO2.3_9

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
At least one algorithm is implemented and microservices are deployed at MiL in operation	Test that uncertainty detection microservice works within MiL configured as CPSoS in operation.	The tool chain is functioning with success response message.

Compliance with the interface defined to plug-in different microservices and CPSoS elements at MiL test level. Steps to reproduce:

1. Configure the uncertainty detection service to establish connection with other microservices through the predefined interface (SUT)

2. Invoke the interface to configure within a common request/response-based communication mechanism (e.g., HTTP, RESR) through a common DSL tool (e.g. XML, JSON) used by all the microservices and elements of CPSoS.

3. Start the uncertainty detection service

3.3.6.2.3 SRL_STO_2_3_Test_3

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_3

STO2.3_9

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
At least one algorithm is implemented and microservices are deployed at SiL in operation	Test that uncertainty detection microservice works within SiL configured as CPSoS in operation.	The tool chain is functioning with success response message.

Compliance with the interface defined to plug-in different microservices and CPSoS elements at SiL test level. Steps to reproduce:

1. Configure the uncertainty detection service to establish connection with other microservices through the predefined interface (SUT)

2. Invoke the interface to configure within a common request/response-based communication mechanism (e.g., HTTP, RESR) through a common DSL tool (e.g. XML, JSON) used by all the microservices and elements of CPSoS.

3. Start the uncertainty detection service

3.3.6.2.4 SRL_STO_2_3_Test_4

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_4

STO2.3_9

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
At least one algorithm is implemented and microservices are deployed	Test that uncertainty detection microservice works within HiL configured as CPSoS in operation.	The tool chain is functioning with success response message.

Compliance with the interface defined to plug-in different microservices, physical connectors and CPSoS elements at HiL test level. Steps to reproduce:

1. Configure the uncertainty detection service to establish connection with other microservices through the predefined interface (SUT)

2. Invoke the interface to configure within a common request/response-based communication mechanism (e.g., HTTP, RESR) through a common DSL tool (e.g. XML, JSON) used by all the microservices and elements of CPSoS.

3. Start the uncertainty detection service

3.3.6.2.5 SRL_STO_2_3_Test_5

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_5

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Heuristic algorithms are implemented and configured within the microservice	Test that the heuristics (e.g., based on relevant similarity functions) to determine whether we have learned an unforeseen situation is generating outputs as configured	Efficient outcome of the algorithms with least percentage of errors, optimal and highest precision.

3.3.6.2.6 SRL_STO_2_3_Test_6

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_6

Verification Method: Comparison

Test Data:

Preconditions	Description	Expected Results
Operation time model available to extract and analysis	Test that uncertainty detection microservice provides improved model outputs learned during the design-time	Comparison reports providing higher precision, lower error in detection, faster triggering and recovery service invoke

3.3.6.2.7 SRL_STO_2_3_Test_7

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_7

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Heuristic algorithms are implemented and configured within the microservice for detecting uncertain situations. - uncertainty detection service started	Test that applied adaptive (/active) and passive techniques (Machine Learning Algorithms) on live monitoring data have learned to determine the uncertain situations.	Successful detection of uncertainty using heuristics.

3.3.6.2.8 SRL_STO_2_3_Test_8

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_8

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
-uncertainty detection service started. -detection heuristics applied -detected uncertain situation -elastic traffic dispatcher	Test elasticity of microservice deployment	Microservice is operational with upscaled and downscaled live monitoring data

Comments

Considering load of test oracles with monitoring data during heavy and less traffic generation use-cases scenarios.

3.3.6.2.9 SRL_STO_2_3_Test_9

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_10

Verification Method: Analysis

Test Data:

Preconditions	Description	Expected Results
Rigorous analysis on applied ML algorithms on test oracles	Test optimal tuning of configuration parameters of employed machine learning algorithms.	Optimality evidence in decreased error rate and increased algorithm efficiency

3.3.6.2.10 SRL_STO_2_3_Test_10

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_11

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
- Test scenario generated. - Algorithms and mechanisms implemented for microservice - Uncertainty detection service started. - Detection heuristics applied - Detected uncertain situation	Modularity test of uncertainty detection microservice if capable to support required configuration for algorithms and scenarios	Successful detection of uncertainty using heuristics for different heuristic algorithms and mechanisms

3.3.6.2.11 SRL_STO_2_3_Test_11

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_12

Verification Method: Analysis

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -detection heuristics applied -detected uncertain situation	Test that the uncertainty detection microservice learns from existing model of CPSoS and improve the model.	Learning rate analysis curve upward

3.3.6.2.12 SRL_STO_2_3_Test_12

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_13

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -detection heuristics applied -detected uncertain situation -updated uncertainty model	Test that the digital twin of the uncertainty situation detection microservice functions with similar capabilities in operation (MiL) as designed.	successful functioning behaviour of uncertainty model

3.3.6.2.13 SRL_STO_2_3_Test_13

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_14

Verification Method: Analysis

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -detection heuristics applied -detected uncertain situation	Test that the digital twin of uncertainty situation detection microservice updates the model from leaning.	Digital twin model updated with changes in learning curve.

3.3.6.2.14 SRL_STO_2_3_Test_14

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_15

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -detection heuristics applied -detected uncertain situation -invoked recovery microservice -uncertainty detection service stopped	Test and inspect the operation log of the uncertainty distribution detection microservice for each uptime period.	Entry for every operation

3.3.6.2.15 SRL_STO_2_3_Test_15

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_16

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
-Test scenarios generated - chosen test for modularity test -algorithms and mechanisms implemented for microservice -uncertainty detection service started. -detection heuristics applied -detected uncertain situation -invoked recovery microservice -uncertainty detection service stopped	Test the modularity of uncertainty microservice configuration in terms of applied algorithms and scenarios	- Successful detection of uncertainty using heuristics for different heuristic algorithms and mechanisms - Fully functional microservice - Insignificant changes in functional behaviour

3.3.6.2.16 _STO_2_3_Test_16

Author:

Status: Proposed

Verifies Requirements:

STO2.3_17

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -detection heuristics applied -detected uncertain situation -invoked recovery microservice	Test response function and capture time the uncertainty detection microservice takes to communicate with monitoring microservice, validation microservice, and recovery microservice.	-Return of response function OK. -Captured response time

3.3.6.2.17 SRL_STO_2_3_Test_17

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_18

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -detection heuristics applied -detected uncertain situation -invoked recovery microservice	Tests invoke function and capture time the uncertainty detection microservice takes to communicate with recovery microservice since the detection of uncertainty situation	-Return of invoke function DONE. -Captured invoke time since invoke function is called until the recovery microservice starts recovery function and sends ACK

Comments

Recovery microservice shall send ACK message to Uncertainty Detection microservice once it launches the recovery function

3.3.6.2.18 SRL_STO_2_3_Test_18

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_19

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -Return of response function OK	Test data retrieve function and capture time the uncertainty detection microservice takes to receive data from monitoring microservice since the execution of response function	-Return of receiving function STARTED. -Captured time between requisition and receival of data

3.3.6.2.19 SRL_STO_2_3_Test_19

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_20

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -detection heuristics applied -invoked validation microservice	Test receiving function and capture time the uncertainty detection microservice takes to receive test results from validation microservice since the execution of invoke function	-Return of response function OK. -Captured time between requisition and receival of validation verdict/test result

3.3.6.2.20 SRL_STO_2_3_Test_20

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_21

STO2.3_22

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -detection heuristics applied -detected uncertain situation -invoked recovery microservice -uncertainty detection service stopped	Test and capture uptime for the uncertainty detection microservice	Captured time between start and stop of microservice

3.3.6.2.21 SRL_STO_2_3_Test_21

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_23

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -Return of response function OK -Received monitoring data -Received validation verdict -Data pre-processing started	Test and calculate noise in the received data the uncertainty detection microservice uses	Noise value in the data

3.3.6.2.22 SRL_STO_2_3_Test_22

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_24

Verification Method: Analysis

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -received monitoring data -received test verdicts -detection heuristics applied -detected uncertain situation -invoked recovery microservice -microservice reliability assessment module called	Test the uncertainty situation detection algorithm accuracy to assess the reliability of the uncertainty detection microservice	return on accucary function

3.3.6.2.23 SRL_STO_2_3_Test_23

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_25

Verification Method: Analysis

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -received monitoring data -received test verdicts -detection heuristics applied -detected uncertain situation -invoked recovery microservice -microservice reliability assessment module called	Test the uncertainty situation detection algorithm efficiency to assess the reliability of the uncertainty detection microservice	return on efficiency function

3.3.6.2.24 SRL_STO_2_3_Test_24

Author: SRL

Status: Proposed

Verifies Requirements:

STO2.3_26

Verification Method: Analysis

Test Data:

Preconditions	Description	Expected Results
-Test scenario generated. -Algorithms and mechanisms implemented for microservice -uncertainty detection service started. -received monitoring data -received test verdicts -detection heuristics applied -detected uncertain situation -invoked recovery microservice -microservice reliability assessment module called	Test the uncertainty situation detection algorithm unbiasedness index to assess the reliability of the uncertainty detection microservice	return on unbiasedness calculation function

3.3.7 STO 2.4 Recovery Microservice Tests

3.3.7.1 MGEP_STO_2_4_Test_1

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.4_5

STO2.4_10

STO2.4_12

STO2.4_13

STO2.4_3

STO2.4_4

STO2.4_14

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
TBD	Test Oracle or/and Uncertainty module fail detection: new validation plan. Steps to reproduce: 1. Configure the system to send a fail/uncertain event/signal that must activate a new validation plan. 2. Send the event to the Recovery Microservice (RM) 3. RM checks the event with the configured rules and selects the output (recovery action to be launched) 4. RM send the message that will activate a new validation plan (to be sent to the Continuous Validation Microservice)	TBD

3.3.7.2 MGEP_STO_2_4_Test_2

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.4_6

STO2.4_10

STO2.4_12

STO2.4_13

STO2.4_3

STO2.4_4

STO2.4_14

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
TBD	TestOracle or/and Uncertainty module fail detection: new monitoring plan. Steps to reproduce: 1. Configure the system to send a fail/uncertain event/signal that must activate a new monitoring plan. 2. Send the event to the Recovery Microservice (RM) 3. RM checks the event with the configured rules and selects the output (recovery action to be launched) 4. RM send the message that will activate a new monitoring plan (to be sent to the Monitoring Microservice)	TBD

3.3.7.3 MGEP_STO_2_4_Test_3

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.4_7

STO2.4_10

STO2.4_12

STO2.4_13

STO2.4_3

STO2.4_4

STO2.4_14

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
TBD	TestOracle or/and Uncertainty module fail detection: new deployment plan. Steps to reproduce: 1. Configure the system to send a fail/uncertain event/signal that must activate a new deployment plan. 2. Send the event to the Recovery Microservice (RM) 3. RM checks the event with the configured rules and selects the output (recovery action to be launched) 4. RM send the message that will activate a new deployment plan (to be sent to the Deployment Microservice)	TBD

3.3.7.4 MGEP_STO_2_4_Test_4

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.4_8

STO2.4_10

STO2.4_12

STO2.4_13

STO2.4_3

STO2.4_4

STO2.4_14

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
TBD	TestOracle or/and Uncertainty module fail detection: send an alert to the CI service. Steps to reproduce: 1. Configure the system to send a fail/uncertain event/signal that must activate an alert to be sent to the CI service. 2. Send the event to the Recovery Microservice (RM) 3. RM checks the event with the configured rules and selects the output (send an alert to CI) 4. RM sends the alert to the CI service	TBD

3.3.7.5 MGEP_STO_2_4_Test_5

Author: MGEP

Status: Proposed

Verifies Requirements:

STO2.4_9

STO2.4_10

STO2.4_12

STO2.4_13

STO2.4_3

STO2.4_4

STO2.4_14

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
TBD	TestOracle or/and Uncertainty module fail detection: go to a stable version. Steps to reproduce: 1. Configure the system to send a fail/uncertain event/signal that must activate the action that starts a previous stable deployment plan. 2. Send the event to the Recovery Microservice (RM) 3. RM checks the event with the configured rules and selects the output (recovery action to be launched) 4. RM send the message that will activate a previous stable deployment plan (to be sent to the Deployment Microservice)	TBD

3.3.8 STO 3.1 DSL for Continuous Validation Tests

3.3.8.1 MGEP_STO_3_1_Test_1

Author: MGEP

Status: Proposed

Verifies Requirements:

STO3.1_1

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Validation plan and all its characteristics are detailed	The DSL syntax for the specification of a validation plan will be checked: 1-A validation plan (or set of validation plans) will be designed and written down in a document, which covers several scenarios 2-The validation plan is translated into the DSL	The DSL can specify the validation plan for MiL, SiL and HiL

3.3.8.2 MGEP_STO_3_1_Test_2

Author: MGEP

Status: Proposed

Verifies Requirements:

STO3.1_2

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Different characteristics of CPSoS properties have been analysed	The DSL Syntax for the specification of CPSoS properties will be checked: 1- The properties of a CPSoS (possible one of the use-cases) will be analysed and written down in a document. 2- The properties are translated into the DSL	The DSL can specify CPSoS properties

3.3.8.3 MGEP_STO_3_1_Test_3

Author: MGEP

Status: Proposed

Verifies Requirements:

STO3.1_3

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Required characteristics of the different artifacts are available	The DSL syntax expressiveness to specify properties of the different artifacts will be checked: 1- The properties that the test executor shall have will be analysed and written down in a document, through specific examples. 2- The properties that the test stimulus shall have will be analysed and written down in a document, through specific examples. 3- The properties different system statuses will be analysed based on the use-cases and written down in a document, through specific examples. 4- The properties that triggering and finishing criteria shall have will be analysed and written down in a document, through specific examples. 5- The properties of all the different oracles will be analysed and written down in a document, through specific examples. 6- The different specific examples specified in points 1 to 5 will be translated to the DSL.	The DSL can specify all the necessary properties to allow the generation of different verification and validation artifacts

3.3.8.4 MGEP_STO_3_1_Test_4

Author: MGEP

Status: Proposed

Verifies Requirements:

STO3.1_4

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
The DSL and the M2T generator are ready	The test artifacts generation interface will be checked: 1-The DSL generates the files to be obtained as inputs the test artifacts. 2- The test artifacts automatically generate the right artifacts by considering the inputs of these files	The test artifacts generators can generate test artifacts through the files generated by the DSL

3.3.8.5 MDH_STO_3_1_Test_5

Author: MDH

Status: Proposed

Verifies Requirements:

STO3.1_5,7,8,9,10

Verification Method: Analysis

Test Data:

Preconditions	Description	Expected Results
Requirements	A representative subset of the requirements is to be expressed in the DSL and automatically generated. The success rate of the translation is noted.	The generated test validation artifacts is checked to be at least 80%

3.3.8.6 MDH_STO_3_1_Test_6

Author: MDH

Status: Proposed

Verifies Requirements:

STO3.1_7

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
Oracles and log files	The independence of the oracles will be checked with respect to i) temporal position in log file, ii) other oracles	Each oracle should be successfully applied on all log files. Evaluation order of the oracles shall not influence the result.

3.3.8.7 MDH_STO_3_1_Test_7

Author: MDH

Status: Proposed

Verifies Requirements:

TBD

Verification Method: Analysis

Test Data:

Preconditions	Description	Expected Results
A representative subset of the requirements translated using the tool	The toolchain will be checked by observing the work of translating requirements	The tool should have given sufficient support for temporal specification using the DSL as well as interactivity.

3.3.9 STO 3.2 Unforeseen Situation Detection at Design Time Tests

The purpose of this section is to refine all the test validation criteria, the Unforeseen Situation Detection should satisfy to support the ADEPTNESS toolchain in determining the uncertainty situations of CPSoS at the design time in an automated fashion. The prime objective of such tests is to check whether the Unforeseen Situation Detection at Design Time meets the enlisted functional and non-functional list of STO requirements by extracting the knowledge from CPSoS operational data for further deployment of CPSoS on real and virtual infrastructure. These requirements are defined based on the technology used by the use case providers (e.g., target hardware, type of deployment, communication protocols, security, etc.).

3.3.9.1 Definitions, Acronyms and Abbreviations

Automated and optimized tests: Perform automation and optimization in the process of test operation to reduce redundancy and increase efficiency.

Parameter Tuning: The technique of adjusting the parameters to perform automated and optimized test which control the behaviour of unforeseen situation detection models.

Reliability metrics: Interventions or metrics to assess the reliability of the uncertainty detection microservice upon deployment on a use case scenario.

3.3.9.2 Tests

3.3.9.2.1 SRL_STO_3_2_Test_1

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_1

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
Models to reproducing the test cases in the different monitoring, validation, and deployment steps	Test that Unforeseen Situation Detection is integrated with toolchain functioning at the design time (static and dynamic/interactive/ runtime) within a CPSoS in MiL.	The toolchain is functional and automatic feedback to real and virtual deployment scenarios are configurable

3.3.9.2.2 SRL_STO_3_2_Test_2

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_2

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
Software in use are configured with monitoring, validation, and deployment requirements	Test that Unforeseen Situation Detection is integrated with toolchain functioning at the design time (static and dynamic/interactive/ runtime) within a CPSoS in SiL	The toolchain is functional and automatic feedback to real and virtual deployment scenarios are configurable

3.3.9.2.3 SRL_STO_3_2_Test_3

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_3

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
Hybrid simulation environment is instrumented with monitoring, validation, and deployment requirements	Test that Unforeseen Situation Detection is integrated with toolchain functioning at the design time (static and dynamic/interactive/ runtime) within a CPSoS in HiL.	The toolchain is functional and automatic feedback to real and virtual deployment scenarios are configurable

3.3.9.2.4 SRL_STO_3_2_Test_4

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_4

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
Virtual deployment environment is configured with monitoring, validation, and deployment requirements to reflect real world scenarios	Test that uncertainty detection is reproducible within real world scenarios in the virtual infrastructure.	The toolchain is functional and automatic feedback to virtual deployment scenarios are configurable

3.3.9.2.5 SRL_STO_3_2_Test_5

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_5

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Virtual deployment environment is configured with monitoring, validation, and deployment requirements to reflect real world scenarios	Test and analyse the learning behaviour of the applied ML (e.g., reinforcement learning) on the test oracles to determine if situations are unsafe.	Test results, detected safe or unsafe

3.3.9.2.6 SRL_STO_3_2_Test_6

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_6

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
Learning algorithms, rewards function and policy model are implemented within the toolchain	Specification verification of reinforcement learning algorithm (e.g., rewards functions) to guide the process towards learning unforeseen situations of CPSoS.	Optimization and verification through validating the algorithm, platform, and data assumptions.

3.3.9.2.7 SRL_STO_3_2_Test_7

Author: SRL

Status:

Verifies Requirements:

STO3.2_7

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Learning algorithms, rewards function and policy model are implemented within the toolchain	Test and verify passive machine learning techniques are learning on the available data to understand the behaviours of CPSoS.	Valid algorithm, platform, and data assumptions.

Comments

Optimization and verification of passive techniques

3.3.9.2.8 SRL_STO_3_2_Test_8

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_9

Verification Method: Comparison

Test Data:

Preconditions	Description	Expected Results
Learning algorithms, rewards function and policy model are implemented within the toolchain	Test comparison mechanisms to determine whether unknown behaviours are discovered through passive learning.	Positive detection on uncertainty situation in operational test oracle

3.3.9.2.9 SRL_STO_3_2_Test_9

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_10

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Learning algorithms, rewards function and policy model are implemented within the toolchain	Test and verify active machine learning techniques are learning on the available data to improve the learning model in understanding the behaviours of CPSoS.	Valid algorithm, platform, and data assumptions.

Comments

Optimization and verification of passive techniques

3.3.9.2.10 SRL_STO_3_2_Test_10

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_11

Verification Method: Comparison

Test Data:

Preconditions	Description	Expected Results
Learning algorithms, rewards function and policy model are implemented within the toolchain	Test comparison mechanisms to determine whether unknown behaviours are discovered through active learning.	Positive detection on uncertainty situation in operational test oracle

3.3.9.2.11 SRL_STO_3_2_Test_11

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_12

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Learning algorithms, rewards function and policy model are implemented within the toolchain	Test the optimization of parameter tuning used in the machine learning techniques.	TBD

Comments

To discuss

3.3.9.2.12 SRL_STO_3_2_Test_12

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_13

Verification Method: Comparison

Test Data:

Preconditions	Description	Expected Results
Learning algorithms, rewards function and policy model are implemented within the toolchain	Execute the methodology to generate test cases from the learned models.	Output data matches the reference data

3.3.9.2.13 SRL_STO_3_2_Test_13

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_14

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Learning algorithms, rewards function and policy model are implemented within the toolchain	Test design time uncertainty detection learning process whether it provides a model of unforeseen situations.	Output data is the parameters of an uncertainty model

3.3.9.2.14 SRL_STO_3_2_Test_14

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_15

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
TBD	Test and inspect the model logs the uncertainty situation detection at design produces during the active learning period.	TBD

3.3.9.2.15 SRL_STO_3_2_Test_15

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_16

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
TBD	Test and inspect the data logs uncertainty situation detection at design produces during the passive learning period according to mining rules	TBD

3.3.9.2.16 SRL_STO_3_2_Test_16

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_17

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Learning algorithms, rewards function and policy model are implemented within the toolchain. Need for software update is triggered to carry out the learning.	Test if software update is permeable as the requirements are met.	Updated software version/ infrastructure status

Comments

When the software update can be triggered? Under what circumstances or situation

3.3.9.2.17 SRL_STO_3_2_Test_17

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_18

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
- Learning algorithms, rewards function and policy model are implemented within the toolchain. - Change in optimal configuration and perform repetitive workflow	Test that uncertainty detection at design time has optimal configuration capacity	Successful functional behaviour of toolchain with least bugs in the workflow - Detects bugs in the flow with changed optimal settings

3.3.9.2.18 SRL_STO_3_2_Test_18

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_19

STO3.2_20

STO3.2_21

Verification Method: Demonstration

Test Data:

Preconditions	Description	Expected Results
Learning algorithms, rewards function and policy model are implemented within the toolchain. Test cases generated in continuous manner with load traffic on the operational data.	Configure test to perform learning under traffic load at scale.	Successful functional behaviour of toolchain

3.3.9.2.19 SRL_STO_3_2_Test_19

Author: SRL

Status: Proposed

Verifies Requirements:

STO3.2_24

Verification Method: Comparison

Test Data:

Preconditions	Description	Expected Results
Learning algorithms, rewards function and policy model are implemented within the toolchain	Test and compare on the reliability metrics during the training period of the applied machine learning models and techniques.	Efficiency, accuracy, and unbiasedness parameters of different algorithms in application

3.3.9.2.20 _STO_3_2_Test_20

Author:

Status: Proposed

Verifies Requirements:

STO3.2_22

STO3.2_23

STO3.2_25

Verification Method: Comparison

Test Data:

Preconditions	Description	Expected Results
Learning algorithms, rewards function and policy model are implemented within the toolchain	Test and compare on the reliability metrics during after the training period of the applied machine learning models and techniques.	Efficiency, accuracy, and unbiasedness parameters of different algorithms in application

3.3.10 STO 3.3 Automated Test Regeneration Tests

3.3.10.1 MDH_STO_3_3_Test_1

Author: MDH

Status: Proposed

Verifies Requirements:

STO3.3_1

STO3.3_2

STO3.3_3

Verification Method: TBD

Test Data:

Preconditions	Description	Expected Results
TBD	A test case is successfully executed in MiL/SiL/HiL through the validation microservice	TBD

3.3.10.2 MDH_STO_3_3_Test_2

Author: MDH

Status: Proposed

Verifies Requirements:

STO3.3_4

STO3.3_6

STO3.3_9

STO3.3_10

STO3.3_11

Verification Method: TBD

Test Data:

Preconditions	Description	Expected Results
TBD	A test case is successfully reused or augmented between MiL/SiL/HiL validation	TBD

3.3.10.3 MDH_STO_3_3_Test_3

Author: MDH

Status: Proposed

Verifies Requirements:

STO3.3_5

Verification Method: TBD

Test Data:

Preconditions	Description	Expected Results
TBD	The coverage metrics calculated at the end of each test execution is correct	TBD

3.3.10.4 MDH_STO_3_3_Test_4

Author: MDH

Status: Proposed

Verifies Requirements:

STO3.3_7

Verification Method: TBD

Test Data:

Preconditions	Description	Expected Results
TBD	The validation microservice allows online and offline selection of tests	TBD

3.3.11 STO 3.4 Impact of Operation Data On Lifecycle Artifacts Tests

3.3.11.1 UES_STO_3_4_Test_1

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_9

STO3.4_13

STO3.4_46

STO3.4_47

STO3.4_25

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Test Verdict and Monitorization Operational Data is provided to MQTT	Development artifacts status update from Test Validation Report data: 1. Verdict is published in MQTT Broker 2. Traceability microservice collects oracle microservice data: Test Environment Setup, Test Inputs, Test Start Criteria, Test Validation Conditions, Test Finishing Criteria and Test Verdict 3. Traceability microservice collects related validation microservices data: Linked Development Artifacts and Linked V&V Artifacts. 4. Traceability microservice collects related operational data from monitorization microservice: Test Execution Operational Data. 5. Traceability microservice generates Test Validation Report. 6. Traceability Microservice updates corresponding development artifact in the developer tool via OSLC	The test validation report data is updated in development tools via OSLC and notification is sent to developer

3.3.11.2 UES_STO_3_4_Test_2

Author: UES

Status:

Verifies Requirements:

STO3.4_10

STO3.4_16

STO3.4_46

STO3.4_47

STO3.4_25

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Test Verdict and Monitorization Operational Data is provided to MQTT	V&V artifacts status update from Test Validation Report data: 1. Verdict is published in MQTT Broker 2. Traceability microservice collects oracle microservice data: Test Environment Setup, Test Inputs, Test Start Criteria, Test Validation Conditions, Test Finishing Criteria and Test Verdict 3. Traceability microservice collects related validation microservices data: Linked Development Artifacts and Linked V&V Artifacts. 4. Traceability microservice collects related operational data from monitorization microservice: Test Execution Operational Data. 5. Traceability microservice generates Test Validation Report. 6. Traceability Microservice updates corresponding V&V artifact in the validation tool via OSLC	The test validation report data is updated in validation tools via OSLC and notification is sent to validator

3.3.11.3 UES_STO_3_4_Test_3

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_11

STO3.4_12

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
DSL OSLC interface is defined	DSL - OSLC integration	DSL OSLC interface document is registered and its fulfilled

3.3.11.4 UES_STO_3_4_Test_4

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_27

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
A non-linked operation-time data related to development-time artifact is identified	Traceability link generation	OSLC link is created between operation-time data and the lifecycle artifact

3.3.11.5 UES_STO_3_4_Test_5

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_26

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Provided MQTT data is modified	Traceability of CPSoS lifecycle artifacts during operation	Traceability microservice updates the data in all linked lifecycle artifacts

3.3.11.6 UES_STO_3_4_Test_6

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_29

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Workflow is defined and documented	Adeptness workflow integration	Workflow is integrated in Adeptness toolchain

3.3.11.7 UES_STO_3_4_Test_7

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_30

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Lifecycle management and development methods are defined and documented	Adeptness lifecycle management and development methods integration	Lifecycle management and development methods are integrated in Adeptness toolchain

3.3.11.8 UES_STO_3_4_Test_8

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_31

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Each Adeptness tool OSLC domain model is defined	Adeptness toolchain OSLC domain model	Each Adeptness tool OSLC domain model document is stored in NextCloud

3.3.11.9 UES_STO_3_4_Test_9

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_32

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Each Adeptness tool has implemented https and SSL support	Adeptness toolchain https and SSL	https connection is stablished with tools

3.3.11.10 UES_STO_3_4_Test_10

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_33

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Consumer/Friend relationship is implemented and configured between Adeptness tools	Adeptness toolchain authentication	OSLC Oauth tests are passed

3.3.11.11 UES_STO_3_4_Test_11

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_34

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Each Adeptness tool has defined a rootservice document for discovery capabilities	Adeptness toolchain rootservice document	Rootservices documents are available in given URLs

3.3.11.12 UES_STO_3_4_Test_12

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_35

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Each Adeptness tool has defined a project area artifact container	Adeptness toolchain project area artifact container	Different tools project areas are associated and links between their artifacts are created

3.3.11.13 UES_STO_3_4_Test_13

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_36

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Each Adeptness tool available link types are defined	Adeptness toolchain link types	Defined links are available between different tools

3.3.11.14 UES_STO_3_4_Test_14

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_37

Verification Method: Test

Test Data:

Preconditions	Description	Expected Results
Each Adeptness tool has defined a TRS provider	Adeptness toolchain TRS	TBD

3.3.11.15 UES_STO_3_4_Test_15

Author: UES

Status: Proposed

Verifies Requirements:

STO3.4_38

STO3.4_39

Verification Method: Inspection

Test Data:

Preconditions	Description	Expected Results
Customized Adeptness tool dialogs are implemented	Adeptness toolchain OSLC dialogs customization	Dialogs are customized according to the design document