Melodic
Big data cloud

Title:

Integration & testing requirements

Abstract:

A careful design of the Melodic integration strategy is very important as the main mission of Melodic is to integrate and adapt underlying frameworks, including PaaSage and CACTOS. This deliverable provides a detailed description of the requirements of this integration. The methodology for collecting integration requirements focuses on identifying the requirements separately for Control & Data Plane and Monitoring Plane. The other key set of requirements is functional testing and UI components testing, which specifies rules and conditions verifying the proper implementation of respective functional features of the Melodic system. The functional testing requirements are provided in the form of testing scenarios, which should be executed for a given Melodic release. These scenarios include initial deployment testing, global reconfiguration & local reconfiguration testing, metric management testing, reasoning related testing, and API testing. D2.1 "System specification" and use-case application descriptions are two main sources of identifying testing requirements. Apart from functional, also non-functional testing requirements are covered to verify the proper implementation of the non-functional features of the Melodic system. The non-functional testing scenarios include fault handling testing, performance testing, security testing, and other non-functional testing.

Multi-cloud Execution-ware for Large-scale Optimised Data-Intensive Computing

H2020-ICT-2016-2017
Leadership in Enabling and Industrial Technologies; Information and Communication Technologies

Grant Agreement No.:
731664

Duration:
1 December 2016
30 November 2019

www.melodic.cloud

Deliverable reference:
D5.04

Date:
1 February 2018

Responsible partner:
7bulls

Editor(s):
Paweł Skrzypek

Author(s)
Antonia Schwichtenberg,
Sébastien Kicin,
Katarzyna Materka,
Somnath Mazumdar,
Jörg Domaschka,
Yiannis Verginadis,
Michał Semczuk,
Paweł Skrzypek,
Sebastian Schork

Approved by:
Ernst Gunnar Gran

ISBN number:
N/A

Document URL:
http://www.melodic.cloud/deliverables/D5.04 Integration & testing requirements.pdf

This project has received funding from
the European Union's Horizon 2020 research
and innovation programme under grant agreement No 731664

Document	
Period Covered	M6-12
Deliverable No.	D5.04
Deliverable Title	Integration and Testing Requirements
Editor(s)	Paweł Skrzypek
Author(s)	Sébastien Kicin, Katarzyna Materka, Antonia Schwichtenberg, Somnath Mazumdar, Jörg Domaschka, Yiannis Verginadis, Michał Semczuk, Paweł Skrzypek, Sebastian Schork
Reviewer(s)	Kyriakos Kritikos, Antonia Schwichtenberg, Feroz Zahid
Work Package No.	5
Work Package Title	Integration and Security
Lead Beneficiary	7bulls
Distribution	PU
Version	1.0
Draft/Final	Final
Total No. of Pages	83

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Index of Figures

Index of Tables

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

1 Introduction

The purpose of this document is to specify functional and non-functional requirements of integration and testing of all technical components of the Melodic middleware platform. This chapter describes methodologies used for collecting and presenting the integration requirements (section 1.1), functional testing requirements (section 1.2) and non-functional testing requirements (section 1.3).
The testing requirements are described in the form of general testing scenarios. This is the typical form of describing high level test sequences, which could be transformed in the next step into more detailed test cases. The testing scenarios are prepared for both functional and non-functional requirements. Based on these scenarios, the detailed test cases will be prepared in JIRA[1] before the acceptance tests of Melodic are carried out.

1.1 Scope of the Document

The deliverable contains integration and testing requirements, together with testing scenarios, for the Melodic project. The deliverable is based on the deliverable D2.1 *"System specification"*, D5.01 *"Integration and adaptation strategy"* and D5.02 *"Updates to OSS frameworks"*. Also for the testing requirements, the guidance of test strategy described in the D5.06 *"Test strategy and environment"* deliverable is followed in the area of the rules of test scenarios and test cases creation.

1.2 Audience of the Document

This deliverable should be read by the following persons:

1. Test team – the detailed test cases should be prepared based on presented test scenarios of functional and non-functional testing, in chapter 3 and chapter 4, respectively.
2. Development teams - to confirm the scope and implementation requirements, especially for the integration layer, described in chapter 3.
3. Use case partners - to verify the scope of testing of the Melodic platform, described in chapters 4 and 5.

The deliverable could also be of interest to readers outside of consortium, who would like to know more about the integration and testing required of the Melodic platform.

[1] https://jira.7bulls.eu/secure/Dashboard.jspa?selectPageId=10103 - a free account needs to be created to get access.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

1.3 Purpose and methodology of collecting the integration requirements

1.3.1 Purpose of the integration requirements

The Melodic project is focused on the integration and adaptation of the underlying PaaSage[2] and CACTOS[3], and the extensions covering support for big data management (data awareness and locality). For that reason, a careful design of the integration strategy and method is very important for the project. Based on integration requirements, a detailed description of the integration strategy and integration method is provided. The proper design and further fulfilment of these requirements will be a key point for development and evaluation of the project.

As described in the deliverable D5.01 *"Integration and adaptation strategy"*, the fundamental purpose of integration in Melodic is to achieve smooth cooperation of the components independent from the basic frameworks. This attitude is very important for this project due to the following reasons:

- Only a subset of the components from the PaaSage project are being reused in Melodic, while a large number of new components and extensions are planned to cover data-aware deployment of cross-cloud applications.
- Though Cloudiator[4] - part of the PaaSage and the CACTOS projects - has a certain component structure, the features are exposed by one unified API, which is different than the integration method used in PaaSage. Therefore there is a need to unify integration methods across the whole Melodic project.

There are two separate layers of the integration:

- Control Plane – integration layer for control flow of the process/actions in the system
- Monitoring Plane – integration layer for gathering, processing and storing all the monitoring events and measurements.

Each plane has different purposes and requirements against the integration. The Control Plane is responsible for controlling actions within the process and should be reliable and transactional. The Monitor Plane is focused on fast delivery of a big amount of monitoring data.

The high-level integration and adaptation requirements for each Plane are listed in the deliverable D5.01 *"Integration and adaptation strategy"*. A more detailed description of these requirements is provided in chapter 2 of this deliverable.

[2] http://www.paasage.eu/

[3] http://www.cactosf-project.eu/

[4] https://Cloudiator.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

1.3.2 Methodology of collecting integration requirements

The assumption for the integration requirement collection methodology is that the requirements are identified separately for the Control Plane and for the Monitoring Plane.

The methodology of collecting and describing integration requirements is as follows:

1. The first step of the methodology is to review carefully the objectives of the Melodic project provided in the project's "Description of Action" (DoA), and to create the initial list of integration requirements.
2. The second step is to align and extend the created list of integration requirements with deliverable D2.1 *"System specification"*. The requirements in D2.1 were based on the generic requirements for cross–cloud data-intensive applications.
3. The third step is to review the use-case application requirements as well as to adjust and refine the list of the integration requirements based on them. The conditions of use case applications are described in the D2.1 deliverable, the DoA and in the JIRA[5] user stories. The missing integration requirements will be identified and added to the list.
4. The final step of the methodology is to apply the cloud computing principles and requirements for modern integration solutions (as presented in [1]) to further improve and finalise the list of integration requirements.

These steps are illustrated by Figure 1.

[5] https://jira.7bulls.eu/projects/MEL/issues/MEL-409?filter=allissues

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

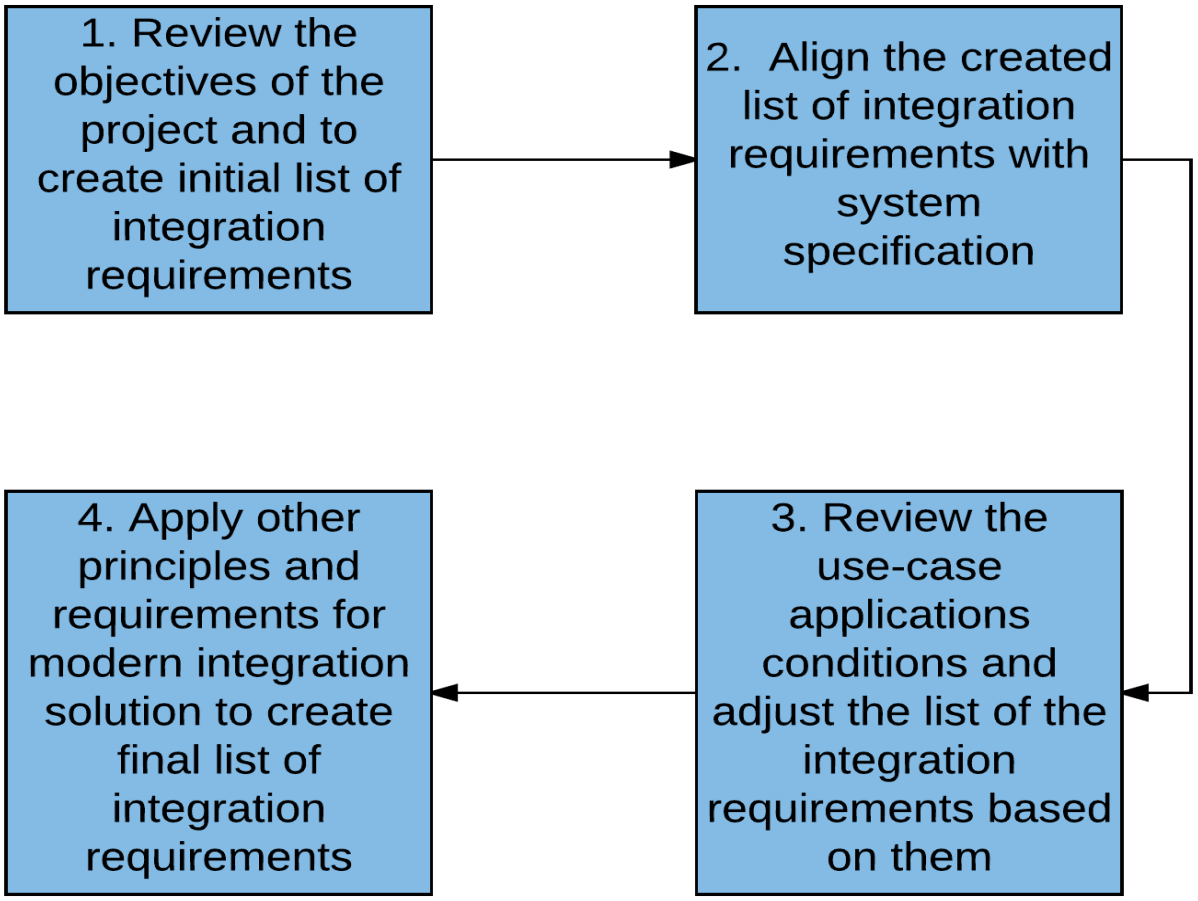

Figure 1 Methodology of collecting integration requirements

1.4 Purpose and methodology of collecting the functional testing requirements

This section contains the purpose, definition and methodology required for collecting the functional testing requirements. References to the sources of the requirements as well as to the quality assurance methodology are also provided.

1.4.1 Purpose of the functional testing requirements

The purpose of functional testing is to perform the respective tests that can lead to an acceptance of a system. A major pre-requisite should be the collection of the functional testing requirements that will guide this functional testing. Once a functional testing requirement is defined, rules and conditions need to be fulfilled to verify if the respective functional feature of the Melodic system is properly implemented. The functional testing requirements are provided in the form of testing

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

scenarios, which should be executed for a given Melodic release. The positive execution of the testing scenarios will constitute a required condition before a Melodic release can be delivered.

The rules and guidance for the testing scenarios creation are described in the deliverable D5.06 *"Test strategy and environment"*.

1.4.2 Methodology of collecting functional testing requirements

The functional testing requirements, in the form of testing scenarios, are collected based on the following methodology, also illustrated by Figure 2:

1. The *"Description of the Action"* is reviewed, and an initial list of high-level functional features described in that document is extracted to be the input for the test scenarios.
2. The deliverable D2.1 *"System specification"* is reviewed and additional testing scenarios are specified for the remaining functional features. For example, based on section 9.4 of D2.1, Architecture Overview, more detailed scenarios for testing the deployment process have been prepared.
3. Also the use-case application descriptions in D2.1 are reviewed and, if needed, missing (functional) testing scenarios are specified. For example, the test scenarios related to the backup and user management have been created.
4. Finally, the produced list of functional test scenarios is reviewed and checked for consistency and completeness.

Test scenarios will be implemented in JIRA as test cases, according to the description provided in D5.10 *"Quality Assurance Guide"*. The test cases name will comprise the scenario number, an indication of whether it is a positive or negative test case and the test case title. The scenario number, the positive/negative indicator and the test case name will be unique for all test cases.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

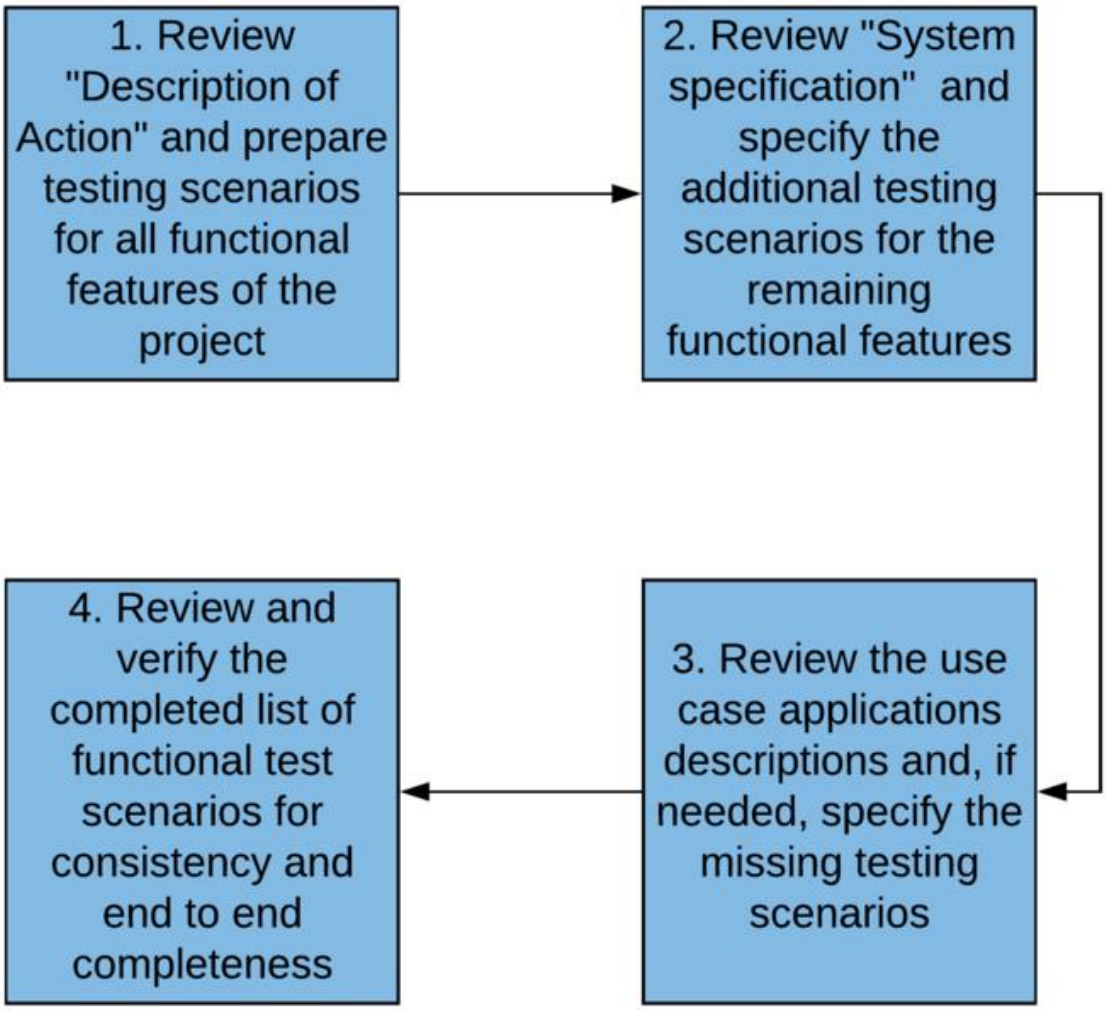

Figure 2 Methodology of collecting functional testing requirements

1.5 Purpose and methodology of collecting non-functional testing requirements

This section contains the purpose, definition and methodology needed for collecting the non-functional testing requirements. References to the sources of requirements as well as to the quality assurance methodology are also provided.

1.5.1 Purpose of non-functional testing requirements

The purpose of the non-functional testing requirements is to verify that the non-functional requirements are fulfilled in order to accept the final release of the system. As the non-functional testing requirements are defined, rules and conditions need to be fulfilled to verify whether the given non-functional feature of the Melodic system is properly implemented. The non-functional

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

testing requirements are provided in form of testing scenarios, which should be executed in the given Melodic release. The positive execution of the testing scenarios will be a required condition for accepting the Melodic release delivered.

The rules and guidance for the testing scenarios creation are described in the D5.06 *"Test strategy and environment"* deliverable.

1.5.2 Methodology of collecting non-functional testing requirements

The methodology for the non-functional testing requirements collection is the following, also shown in Figure 3:

1. The *"Description of Action"* is reviewed and the list of testing scenarios for Melodic non-functional features is prepared. For example, a high-level non-functional feature of Melodic is security in communication between components (see Test Scenario 7.1 in section 5.3 as an example). Based on that high-level feature, testing scenarios will then be prepared.
2. The D2.1 *"System specification"* deliverable is reviewed and additional testing scenarios are specified for the remaining non-functional features. For example, the high-level security requirements for authorization and authentication will be covered in test scenarios. The goal of the example scenario would be to check if the invocation of the method is possible only with valid credentials.
3. Also the use case application descriptions in D2.1 are reviewed and, if needed, the missing testing scenarios are created.
4. Finally, the produced list of non-functional test scenarios is reviewed and checked for consistency and completeness.

During the construction of the test scenarios, the rules described in D5.06 *"Test strategy and environment"* are utilised. Test scenarios will be implemented in JIRA as test cases, according to the description provided in D5.10 "Quality Assurance Guide". The test case's name will contain scenario number, indication if it is a positive or negative test case and the test case title. The scenario number, the positive/negative indicator and the test case name will be unique for all test cases.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

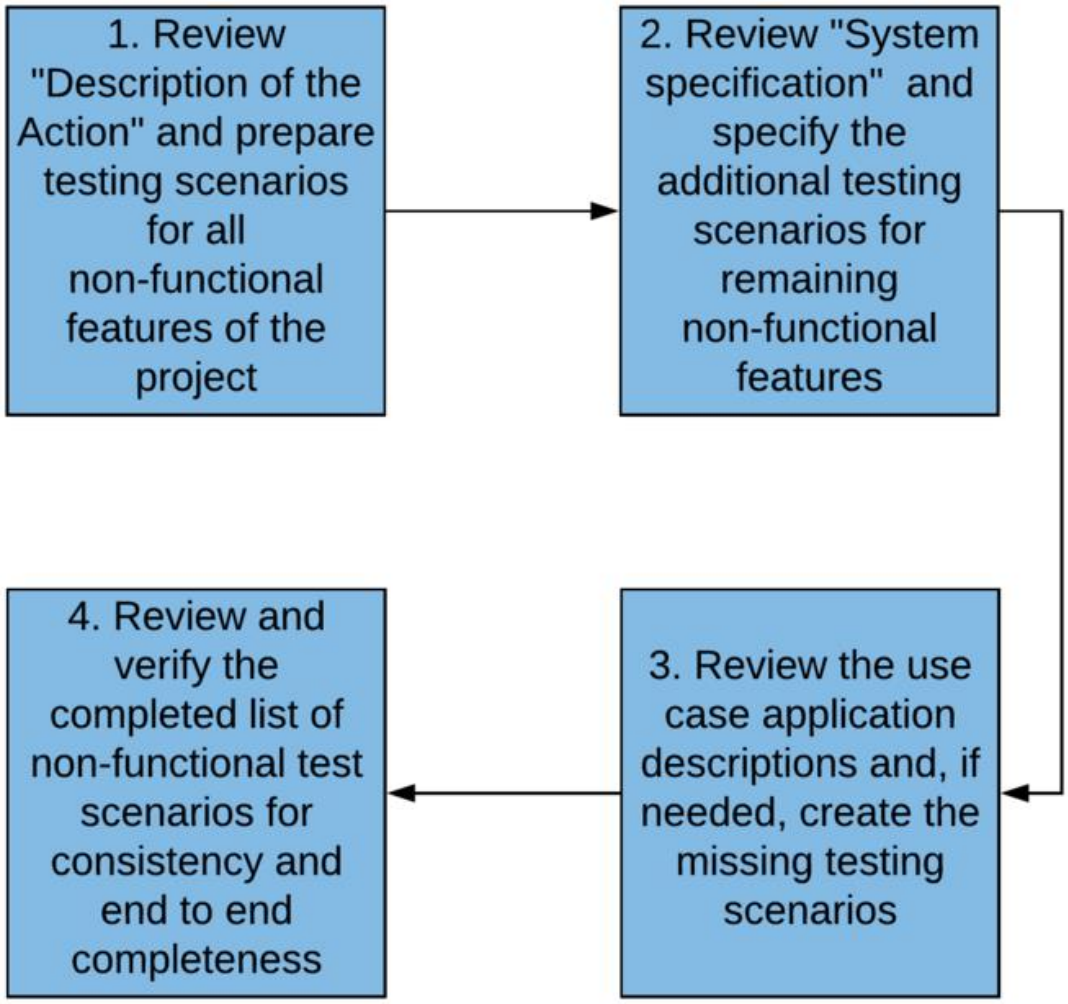

Figure 3 Methodology of collecting non-functional testing requirements

1.6 Structure of the document

This deliverable is divided into two parts. The first part of the document is covered by this chapter. In the second part of the document, covered by chapters 2, 3, 4 and 5, both integration requirements and testing scenarios are analysed in detail. The testing scenarios are divided into functional and non-functional ones; they contain all necessary details to execute the scenarios and evaluate the results. Integration requirements and testing scenarios are concluded in the summary of the deliverable where also some respective conclusions are provided.

The detailed structure of the document is as follows:

- Chapter 2 – Integration requirements: this chapter contains the detailed list of integration requirements with a more detailed description, purpose and usage of each requirement.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

These requirements will be used to produce functional and non-functional testing scenarios.

- Chapter 3 – Functional testing requirements: this chapter analyses functional testing scenarios which are supplied in the form of testing scenarios which should be executed and verified for the Melodic project. The description of each scenario incorporates the scope of the tested functional requirements, the scope of the test data, the list of the steps to execute and the expected result.
- Chapter 4 – Non-functional testing requirements: this chapter provides, in form of testing scenarios which should be executed and verified for the Melodic project, the description of the non-functional testing requirements by describing the list of the non-functional testing scenarios for each type of non-functional requirement. As in the previous chapter, a specific description of each scenario incorporates the scope of the tested functional requirements, the scope of the test data, the list of the steps to execute and the expected result.
- Chapter 5 – Summary: this chapter supplies the summary as well as the main conclusion for this deliverable.

2 Integration requirements

One of the key assumptions for the Melodic project is integration and adaptation of components from fundamental frameworks. This chapter supplies the definition of the integration requirements and a brief description of the integration strategy and method in Melodic, based on the integration strategy chosen in D5.01 *"Integration and adaptation strategy"*. In the following sections, the integration requirements are presented, with their detailed description, purpose and impact on the design and realization of the Melodic system.

2.1 Description of the integration components

The integration and adaptation strategies, along with the description of the method of integration in the Melodic project, are described in the D5.01 *"Integration and adaptation strategy"*. Also, the high-level integration architecture of the Melodic system is presented there.

For the Melodic project, a hybrid solution with two different integration methods has been chosen. Such a hybrid, non-unified solution is more difficult to implement and maintain, but allows for the full utilization of the benefits of the two underlying solutions.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

For the integration of components and their interaction within Melodic (mapping to the Control & Data plane), an Enterprise Service Bus (ESB)[6] and a business process management (BPM)[7] based architecture will be exploited. Using a BPM processes for controlling and orchestrating the behaviour of the system will result in an easier and more efficient implementation of new requirements, leading to a high degree of customization. It also allows for more flexibility as such a behaviour can be flexibly adapted when required.

For the Monitoring Plane, ActiveMQ[8], as a queue-based message broker, has been carefully chosen as it fulfils the requirements of the Melodic project for the Monitoring Plane. The metric collection is not a transactional flow, so the loss or late delivery of one measurement of one metric will not impact the operation of the whole system. This is especially true as the system relies on conditions over metric aggregations in order to adapt/reconfigure a user application. So, the loss of one metric measurement would not burden or destroy the capability of the system to adapt the user application. In case of late delivery of a metric measurement, the system will still be working properly; the only possible disadvantage is that the reconfiguration might be completed later than possible. However, as reconfiguration usually takes a long time to execute, such a delay could be considered as negligible.

2.2 Control Data and Flow Integration Requirements

The section contains integration requirements collected using the methodology described in section 1.3.2 for the Control Data and Flow Plane. The requirements are presented in Table 1 below. The structure of the table, for this section and the following ones, is as follow:

- Requirement's Name – the unique name of the requirement.
- Requirement's Purpose – Purpose of introducing a requirement for the Melodic platform.
- Requirement's description – More detailed description of the requirement.
- Source of requirements – Source (document) which defines the requirement or high-level objectives which provide the need for defining the requirement.

[6] http://www.service-architecture.com/articles/web-services/enterprise_service_bus_esb.html

[7] http://searchcio.techtarget.com/definition/business-process-management

[8] http://zguide.zeromq.org/page:all

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 1 Control Data and Flow Integration Requirements

Req. Id	Requirement's Name	Requirement 's Purpose	Requirement's description	Source of requirements
1	High Reliability	To assure stable functioning of the whole integrated system and reliable communication between components.	• Reliable flow of the invoked operations, with full control over the operations' execution and returned results. • Support for transactions and data integrity, as well as for recoverability after the failure.	D2.1 System specification
2	Performance	To assure that the system is able to fulfil the performance requirements of all use case applications.	• Execute a given number of operations within a certain period of time and with given response time. The system should be able to sustain a certain performance level which can be translated in the requirement to execute • Planned use case applications. Sustaining such performance level also requires the detection and addressing of bottlenecks that affect the normal system operation. The detailed performance measurements will be provided in dedicated test cases.	DoA
3	Scalability	Possibility to improve performance in case of more demanding use case applications through the dynamic reservation of additional resources.	• Ability to improve performance through adding new nodes in the integration layer (scaling horizontally) or by upgrading an existing node of integration layer to a more powerful machine (scaling vertically).	D2.1 System specification

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

4	High availability	Ability to continue the system operation in presence of component failures.	• Support for highly available, multi-node configurations, at least in active-passive configuration (active-active configuration can bring about additional benefits, but is more difficult to implement).	D5.01 Integration and adaptation strategy
5	Flexible orchestration	Possibility to adapt the system flexibly in order to apply changing requirements. Thus, such an adaptation can be more rapidly and efficiently performed.	• Ability to dynamically set up an orchestration for the invocation of the methods of the underlying platform components in a flexible and configurable way. It should be possible to configure such an orchestration without the necessity to code and recompile the whole platform.	D5.01 Integration and adaptation strategy
6	Support for synchronous and asynchronous communication	To be able to support different integration requirements in the most efficient way. Various components and features of the system requires different types of communication to achieve best performance and reliability. The different features of the system and respective connections between components require different types of communication.	• The chosen solution for the integration in this plane should support both synchronous and asynchronous (component) communication with an easy way to change this for a given operation. The change should be transparent for the users. It shall be possible to use both types, according to the given requirements. Not fulfilling these requirements will result in using a not optimal method of communication for some given cases.	D5.01 Integration and adaptation strategy
7	Security	• To achieve secure usage of the system. • To be sure that there will be one central point for	• Support for both authentication (user and method invocation) and authorization (users,	DoA

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

		the authentication and the access control over each component belonging to the system	components and external systems) with the capability to define access rights to invoke given operations, and access to data for the latter.	
8	Monitorability	• Capability to support proper system monitoring and optimization. Also for the troubleshooting as well as the discovery of bugs or misconfigurations. • Monitorability includes logging as an extension of monitoring. It allows to access and review all operations executed by the system, especially all operations finished with errors. Due to that it will be possible to track incorrect system behaviour and fix bugs	• Ability to monitor at configurable levels of detail the operations invoked on the integration layer.	D5.01 Integration and adaptation strategy
10	Support for different integration protocols	Allow efficiently to integrate the underlying frameworks and to integrate different cloud providers with different APIs and protocols through an abstraction layer.	• Ability to support different, most common and widely used integration protocols like REST, SOAP, and JMS.	D2.1 System specification
11	Data model transformation	Allows to have one common data model for the whole system and to assure a unified way of transformation from the common (canonical) model into the models of particular components. The transformation from own to canonical data model guarantees the least possible effort in the entry of new components in the system.	• Ability to support data model transformation in the integration layer. Introduce the canonical model for the system. • Prepare and map data transformation from domain models of particular component to canonical model.	D5.01 Integration and adaptation strategy

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

12	Exception/fault handling.	To guarantee a stable and predictable behaviour of the system which can be quite appealing to its possible exploiters/adopters.	• Unified exception handling and retrying of operations.	D2.1 System specification
13	Support for integration standards and patterns	It allows to avoid vendor lock-in and move the system into a different integration solution, if and when needed. The integration process can also be faster.	• Need to implement in a simple and user-intuitive manner the integration in terms of integrating components and to set up the integration layer. • The most typical integration patterns like splitter and aggregator should be possible to implement.	D2.1 System specification

2.3 Monitor Plane integration requirements

The section contains integration requirements collected using the methodology described in section 1.3.2 for the Monitor Plane. The requirements are presented in Table 2 below, according to the format described in section 2.2.

Table 2 Monitor Plane integration requirements

Req. Id	Requirement's Name	Requirement 's Purpose	Requirement's description	Source of requirements
1	Performance	To assure that the system is able to fulfil the performance requirements of all use case applications. There should not be bottlenecks in the normal system operations. This requirement is particularly important for the Monitor plane due to expected high number of transmitted messages.	• Execute a given number of operations within a certain period of time and within a given response time threshold.	Ability to support real use case applications for commercial and non-commercial organizations
2	Low resource usage	As application monitoring is performed in an intrusive manner in Melodic, there is a need to have low resource usage on monitoring plane without compromising efficient	• Low resource usage by the monitoring plane at both the client and server side.	Optimal deployment based on given utility function (e.g., business constraints)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

		application monitoring as well as the proper execution of the application components (in respective user VMs which translates then to saving enough resources to support the actual computation needs for these components).		
3	Support for integration standards and patterns	Allow fast and efficient integration of the Monitor Plane related components. This requirement ensures that there is no proprietary knowledge required to implement connection to integration layer. It is especially important for the Monitoring Plane, due to the distributed nature of the monitoring capabilities, which are installed on each virtual machine deployed by Melodic.	• Need to implement easily the integration in terms of integrating components and setup of the integration layer. The most typical integration patterns like splitter and aggregator should be possible to implement.	Ability to support real use case applications for commercial and non-commercial organizations

3 Functional testing requirements

In this chapter, testing requirements collected using the methodology described in section 1.4.2 are described in the form of functional testing scenarios. The definition of each scenario incorporates the scope of testing, the requirements for testing and the expected results from the scenario execution.

The functional testing scenarios are divided into *scenario groups*. Each scenario group contains scenarios related to a particular high-level feature or process of the Melodic system.

3.1 Initial deployment testing scenarios

This section presents scenarios related to the Initial Deployment Scenario group. This group contains all scenarios related to the initial deployment of an application in the Melodic platform.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 3 scenarios related to the Initial Deployment Scenario group

Test scenario Id	1.1
Name	Installation and deployment of a simple application on one Cloud Provider - webserver installed on Unix-based OS.
Scenario group	Initial deployment
Components to be tested	• CPGenerator/Rule Processor • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • Cloudiator • ESB • BPM • REST CLIENT • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider has been integrated with the Melodic platform, where the user has provided his/her own credentials for the provider. 3. Meta Solver has been configured in this scenario to use CP solver for that case. 4. Cloudiator properly connected to the given Cloud Provider
Input data	1. Complete CAMEL model of the simple application (which includes the definition of the application components and their installation/lifecycle management scripts). This simple application comprises one component which should be installed as a Unix process (no container), in one virtual machine. For example, installation of the Apache web server using a standard (Unix-specific) installation command. 2. CAMEL model of given Cloud Provider prepared and registered in the Melodic platform with at least one virtual machine offer being provided.
Steps to execute scenario	Using appropriate tool (e.g. possibly ENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload application model in CAMEL 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model in CAMEL

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

	3. Offer filtering and CP model (a kind of deployment optimisation model) generation 4. Deployment optimisation model reasoning 5. Deployment Plan-based application reconfiguration.
Expected results	1. A certain virtual machine on the selected Cloud Provider should be created 2. The sole component (e.g. web server) of the simple application should be installed on that virtual machine 3. The application should be run properly (for example, the root web page of the web server should be displayed properly)

Table 4 Test scenario 1.2

Test scenario Id	1.2
Name	Installation and deployment of a two-component application on one Cloud Provider - application presenting records from database.
Scenario group	Initial deployment
Components to be tested	• CPGenerator/Rule Processor • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • Cloudiator • ESB • BPM • REST CLIENT • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider integrated with the Melodic platform, where the user has provided his/her own credentials for the provider. 3. Meta Solver configured to use CP solver. 4. Cloudiator properly connected to the given Cloud Provider
Input data	1. Complete CAMEL model of the two-component application (which includes the definition of these two components and their installation/lifecycle management scripts). For instance, an example application like WordPress could have one business logic component and another one mapping to the underlying database used, like MySQL, in this case. 2. CAMEL model of given Cloud Provider prepared and registered in the Melodic platform with at least one virtual machine offer provided.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Steps to execute scenario	Using appropriate tool (possibly REST CLIENT[9]), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider model 2. Uploading of the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment plan reasoning 5. Deployment Plan-based application reconfiguration.
Expected results	1. Two virtual machine instances (of the same VM flavour/offering) should be created using the selected Cloud Provider. 2. The application should be installed on those VM instances (one business logic component instance should be installed on the first VM instance and the database component instance should be installed on the second VM instance) 3. The application should run properly. This means that proper communication between the application components should have been established (for example, the WordPress initial page can be loaded and displayed to the user from the underlying database).

Table 5 Test scenario 1.3

Test scenario Id	1.3
Name	Installation and deployment of a two-component application on two different Cloud Providers - application presenting records from database
Scenario group	Initial deployment
Components to be tested	• CP generator/Rule Processor • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • Cloudiator • ESB • BPM • REST CLIENT • CDO Server

[9] Tool to connect to REST API in Melodic platform

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least two cloud providers integrated with Melodic platform, where the user has provided his/her own credentials for both of them. 3. Meta Solver configured to use CP solver for that case. 4. Cloudiator properly connected to given Cloud Providers.
Input data	1. Complete CAMEL model of the two-component application (which includes the definition of these two components and their installation/lifecycle management scripts). One component can map to the main business logic of the application and the other to the underlying database used. An example application could be WordPress which also includes an underlying MySQL database. There should be a requirement in the application CAMEL model to use different Cloud Providers (this could be done in different ways; for example, by placing a location requirement that is then referenced in the virtual machine requirement set). 2. CAMEL models of given Cloud Providers have been prepared and registered in the Melodic platform. Each model should contain at least one virtual machine offering being provided.
Steps to execute scenario	Using appropriate tool (possibly REST CLIENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment plan reasoning 5. Deployment Plan-based application reconfiguration.
Expected results	1. Two virtual machine instances mapping to two different VM flavours of the two given cloud providers should be created. 2. The application should be installed on those VM instances (one business logic component instance should be installed on the first VM instance and the database component instance should be installed on the second VM instance) 3. The application should run properly which actually involves that proper communication between application components should have been established (for example, the WordPress initial page can be loaded and displayed to the user from the underlying database).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 6 Test scenario 1.4

Test scenario Id	1.4
Name	Installation and deployment of a simple application in Docker container on one Cloud Provider - webserver installed on Unix-based OS
Scenario group	Initial deployment
Components to be tested	• CPGenerator/Rule Processor • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • Cloudiator • ESB • BPM • REST CLIENT • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider integrated with Melodic platform, where the user has provided his/her own credentials for the cloud provider. 3. Meta Solver configured to use CP solver for that case. 4. Cloudiator properly connected to given Cloud Providers.
Input data	1. Complete CAMEL model of the two-component application (which includes the definition of these two components and their installation/lifecycle management scripts). For instance, an example application like WordPress could have one business logic component and another one mapping to the underlying database used, like MySQL, in this case. 2. For example, installation of the Jboss Drools Workbench using a standard container installation. 3. CAMEL model of given Cloud Provider prepared with at least one virtual machine offer provided. There should be a proper configuration of the virtual machine both in CAMEL Provider model and on the Cloud Provider side. The configurations should be aligned.
Steps to execute scenario	Using appropriate tool (possibly REST CLIENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment Plan reasoning 5. Deployment Plan-based application reconfiguration.
Expected results	1. One virtual machine instance should be created using the selected Cloud Provider. 2. The application should be installed on the VM instance. 3. The application should run properly.

Table 7 Test scenario 1.5

Test scenario Id	**1.5**
Name	Installation and deployment of a two-component application in Docker containers on one Cloud Provider - application presenting records from database
Scenario group	Initial deployment
Components to be tested	• CP generator/Rule Processor • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • Cloudiator • ESB • BPM • REST CLIENT • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least two cloud providers integrated with the Melodic platform, where the user has provided his/her own credentials for both of them. 3. Meta Solver configured to use CP solver for that case. 4. Cloudiator properly connected to given Cloud Providers.
Input data	1. Complete CAMEL model of a two-component application (which includes the definition of these two components and their installation/maintenance scripts). The two components should be installed within Docker containers on two VM instances. 2. CAMEL model of given Cloud Provider prepared and registered in the Melodic platform with at least one virtual machine offer provided.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Steps to execute scenario	Using appropriate tool (possibly REST CLIENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment Plan reasoning 5. Deployment Plan-based application reconfiguration. 6. Deployment to the selected cloud provider
Expected results	1. Two virtual machine instances (of the same VM flavour/offering) should be created using the selected Cloud Provider. 2. The application should be installed on those VM instances (one business logic component instance should be installed on the first VM instance and the database component instance should be installed on the second VM instance) 3. The application should run properly which actually involves that proper communication between application components has been established.

Table 8 Test scenario 1.6

Test scenario Id	1.6
Name	Installation and deployment of a two-component application in Docker containers on two different Cloud Providers - application presenting records from database
Scenario group	Initial deployment
Components to be tested	• CP generator/Rule Processor • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • Cloudiator • ESB • BPM • REST CLIENT • CDO Server

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least two cloud providers integrated with Melodic platform, where the user has provided his/her own credentials for both of them. 3. Meta Solver configured to use CP Solver for that case. 4. Cloudiator properly connected to given Cloud Providers.
Input data	1. Complete CAMEL model of the two-component application (which includes the definition of these two components and their installation/maintenance scripts). The two components are installed as Docker containers in two VM instances of different VM offerings (each mapping to a different cloud provider). For example, installation of WordPress with DAM application using standard container installation. 2. CAMEL model of given Cloud Provider prepared and registered in the Melodic platform with at least one virtual machine offer provided. 3. There should be a requirement in the application to use different Cloud Providers (for example, a location requirement in the virtual machine requirement set in the user CAMEL model)
Steps to execute scenario	Using appropriate tool (possibly REST CLIENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment Plan reasoning 5. Deployment Plan-based application reconfiguration. 6. Deployment to the selected cloud providers
Expected results	1. Two virtual machine instances should be created using the selected Cloud Providers. 2. The application should be installed on those VM instances 3. The application should run properly which actually involves that proper communication between application components has been established (for example, the WordPress initial page can be loaded and displayed to the user from the underlying database).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 9 Test scenario 1.7

Test scenario Id	1.7
Name	Installation and deployment of a two-component application, where one component is installed in a Docker container and another on a normal VM on two different Cloud Providers – application presenting records from database.
Scenario group	Initial deployment
Components to be tested	• CP generator/Rule Processor • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • Cloudiator • ESB • BPM • REST CLIENT • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least two cloud providers integrated with Melodic platform, where the user has provided his/her own credentials for both of them. 3. Meta Solver configured to use CP solver for that case. 4. Cloudiator properly connected to given Cloud Providers.
Input data	1. Complete CAMEL model of the two-component application (which includes the definition of these two components and their installation/lifecycle management scripts). One application component will be installed as a Docker container, the other as a Unix process. Each component is to be deployed on different cloud providers. 2. Two CAMEL models of two Cloud Providers prepared and registered in the Melodic platform with at least one virtual machine offer provided. 3. There should be requirement in the application to use different Cloud Providers (for example, by specifying a location requirement and referencing it in the virtual machine requirement set)
Steps to execute scenario	Using appropriate tool (possibly REST CLIENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment Plan reasoning 5. Deployment Plan-based application reconfiguration. 6. Deployment to the selected cloud providers
Expected results	1. Two virtual machine instances should be created using the selected Cloud Provider. 2. The application should be installed on those VM instances (one business logic component instance should be installed on the first VM instance and the database component instance should be installed on the second VM instance within a container) 3. The application should run properly which actually involves that proper communication between application components has been established.

Table 10 Test scenario 1.8

Test scenario Id	**1.8**
Name	Testing of deployment requirements enforcement.
Scenario group	Initial deployment
Components to be tested	• CP generator/Rule Processor • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • Cloudiator • ESB • BPM • REST CLIENT • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider integrated with the Melodic platform; the user credentials for this provider should also have been supplied. 3. Meta Solver configured to use CP solver for that case. 4. Cloudiator properly connected to the given Cloud Provider

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Input data	1. Complete CAMEL model of a two-component application (which includes the definition of these two components and their installation/maintenance scripts). One component can map to the main business logic of the application and the other to the underlying database used. An example application could be WordPress which also includes an underlying MySQL database. In the CAMEL model there should be features defined which need to be tested (local/remote communication, specific metrics, different OS images and so on). For each tested feature a separate test case should be created. 2. CAMEL model of given Cloud Provider prepared with at least one virtual machine offer provided.
Steps to execute scenario	Using appropriate tool (possibly REST CLIENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment Plan reasoning 5. Deployment Plan-based application reconfiguration.
Expected results	1. The specific feature/deployment requirement defined in the input CAMEL model is properly applied. 2. One or two virtual machine instances (of the same VM flavour/offering) should be created using one or two selected cloud providers, respectively (depending on the deployment requirements posed). 3. The application should be installed on this VM instance or those VM instances (in case of 2 VM instances one business logic component instance should be installed on the first VM instance and the database component instance should be installed on the second VM instance) 4. The application should run properly which actually involves that proper communication between application components has been established (for example, the WordPress initial page can be loaded and displayed to the user from the underlying database).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 11 Test scenario 1.9

Test scenario Id	1.9
Name	Installation and deployment of a two-component application on two different Cloud Providers with more advanced set of requirements, like non-functional ones – application presenting records from database.
Scenario group	Initial deployment
Components to be tested	• CP Generator • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • ESB • BPM • Cloudiator • REST CLIENT • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least two cloud providers integrated with Melodic platform for which the user has supplied his/her credentials. 3. Meta Solver configured in order to use the CP Solver for given case. 4. Cloudiator properly connected to given Cloud Providers.
Input data	1. Complete CAMEL model of a two-component application (which includes the definition of the two components and their installation/maintenance scripts). The first component represents the core application logic while the second the underlying database. An example application could be WordPress which exploits an underlying MySQL database. 2. CAMEL model of each Cloud Provider is prepared with at least one virtual machine offering included. 3. In the application's CAMEL model there should be a set of requirements described in the table with reference data for this test scenario *(see the row below).*
Steps to execute scenario	Using appropriate tool (possibly REST CLIENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment Plan reasoning 5. Deployment Plan-based application reconfiguration.
Expected results	5. The specific feature/deployment requirement defined in the input CAMEL model is properly applied. 1. Two virtual machine instances (of the same VM flavour/offering) should be created using the selected Cloud Provider(s). 2. The application should be installed on those VM instances (one business logic component instance should be installed on the first VM instance and the database component instance should be installed on the second VM instance as a container) 3. The application should run properly which actually involves that proper communication between application components has been established (for example, the WordPress initial page can be loaded and displayed to the user from the underlying database)

In *Table 12*, the reference data for the Test scenario 1.9 are provided. The scenario 1.9 should be executed with all VM offers, constraints, and utility function combinations presented in the table.

Table 12 Reference data for the Test scenario 1.9

Available VM offers	Constraints	Utility function	Expected solution
1. Cores=2, RAM=8, loc=EU, Cost=100, Provider A 2. Cores=4, RAM=16, loc=US, Cost=200, Provider B 3. Cores=4, RAM=32, loc=EU, Cost=250, Provider A 4. Cores=8, RAM=32, loc=EU, Cost=350, Provider A	1. Mem/Cores>7 2. Loc=eu	Min(Cost)	Chosen VM: 3, 3
1. Cores=2, RAM=8, loc=EU, Cost=100, Provider A 2. Cores=4, RAM=16, loc=US, Cost=200, Provider B 3. Cores=4, RAM=32, loc=EU, Cost=250, Provider A	1. Mem/Cores>3	Min(Cores)	Chosen VM: 1,1

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

4. Cores=8, RAM=32, loc=EU, Cost=350, Provider A			
1. Cores=2, RAM=8, loc=EU, Cost=100, Provider A 2. Cores=4, RAM=16, loc=US, Cost=200, Provider B 3. Cores=4, RAM=32, loc=EU, Cost=250, Provider A 4. Cores=8, RAM=32, loc=EU, Cost=300, Provider A	1. VM.1.Cores*Cost> 2400 2. VM.2.Mem<32 3. VM.2.Mem>8 4. Loc in (EU, US)	Min(Cost)	Chosen VM: 4,2

3.2 Metric management testing scenarios

This section presents scenarios related to the Metric Management scenario group. Metric management means the collection, processing (aggregation), storage and delivery of raw and composite metrics as well as CAMEL events based on these metrics. For this scenario group, the Executionware modules are most tested elements (e.g., Metric Collector), but due to the installation of an application – which has also a definition of corresponding metrics and events – the key modules of the Upperware and Executionware are tested too.

Table 13 Test scenario 2.1

Test scenario Id	2.1
Name	Built-in raw metrics collection
Scenario group	Metric management testing scenarios
Components to be tested	• Meta Solver • Metric Collector • Cloudiator • CDO Server • REST CLIENT
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider is integrated with Melodic platform for which the user has supplied the respective credentials. 3. Cloudiator is properly connected to given Cloud Provider(s).
Input data	1. CAMEL model of a simple application (as described in test scenario 1.1) with definition of raw system/built-in metrics. The raw metrics to test are defined in reference data Table 14 for this test scenario (see below).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Steps to execute scenario	Using appropriate tool (possibly REST CLIENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. Actions executed in scenario 1.1 2. Measurement for the raw metrics of given type (see the table with data for this test scenario) should be generated and collected by Executionware (i.e., stored in the Time Series Database (TSDB) of the respective application nodes). These measurements might also be stored in CDO (model repository), depending on whether they refer to metrics which are mapped to SLOs or other objects defined in CAMEL and respective subscribers might be informed about them.
Expected results	1. Values of the raw built-in metric(s) of the given type(s) should be stored in TSDB(s) situated in those VMs on which the respective component of the application to be measured resides. 2. These measurements/values are possibly stored in the CDO and some subscribers (Meta Solver) might be informed about them.

In the *Table 14*, the reference data for the Test scenario 2.1 are provided. In particular, the scenario 2.1 should be executed with the metrics presented in that table.

Table 14 Reference data for the Test scenario 2.1

Raw metric name	Metric description	Metric unit
RawCPUUsage	Current raw CPU usage.	Real value, percentage of usage.
MemoryUsageSensor	Current raw memory usage.	Real value, MB.
DiskIoReadSensor	Disk IO read transfer per second.	Real value, MB per second.
DiskIoWriteSensor	Disk IO write transfer per second.	Real value, MB per second.
FreeDiskSpaceSensor	Current free disk space.	Real value, MB.
RxBytesSensor	Network read transfer per second.	Real value, MB/sec.
TxBytesSensor	Network write transfer per second.	Real value, MB/sec.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 15 Test scenario 2.2

Test scenario Id	2.2
Name	Custom raw metrics collection
Scenario group	Metric management testing scenarios
Components to be tested	• Meta Solver • Metric Collector • Cloudiator • CDO Server • REST CLIENT
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider is integrated with Melodic platform for which the user has provided his/her credentials 3. Cloudiator is properly connected to given Cloud Providers.
Input data	1. CAMEL model of simple application (described in test scenario 1.1) which includes the definition of the custom raw metrics. The raw metrics to test are defined in reference data Table 16 for this test scenario (see below).
Steps to execute scenario	Using appropriate tool (possibly REST CLIENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. Steps from scenario 1.1 2. Measurements for the raw metrics of given type (see the table with data for this test scenario) should be generated and collected by Executionware (i.e., stored in the TSDB on the nodes where the application component is deployed). These measurements may also be stored in the CDO and respective subscribers might be informed about them. Such raw measurements should be stored in CDO only when they map to SLOs or other relevant objects defined in CAMEL.
Expected results	1. Values of the given type raw metric(s) should be stored in TS database(s) on those VMs/nodes on which the respective component of the application to be measured resides. 2. These measurements/values are possibly stored in the CDO and some subscribers (Meta Solver) might be informed about them.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

In Table 16, the reference data for Test scenario 2.2 are provided. The scenario 2.2 should be executed with metrics presented in that table.

Table 16 Reference data for the Test scenario 2.2

Custom metric name	Metric description	Metric unit
Number of users	Custom metric measuring the current number of users.	Number of (something) unit which will be mapped in the end to a certain dimension. Integer value.
Raw Response time	Custom metric measuring the response time for the last request made	Integer value, millisecond.

Table 17 Test scenario 2.3

Test scenario Id	2.3
Name	Composite metric collection
Scenario group	Metric management testing scenarios
Components to be tested	• Meta Solver • Metric Collector • Cloudiator • CDO Server • REST CLIENT
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider is integrated with Melodic platform for which the user has supplied his/her credentials. 3. Cloudiator is properly connected to given Cloud Providers
Input data	1. CAMEL model of simple application (described in test scenario 1.1) which includes the definition of composite metrics. The composite metrics to test are defined in reference data Table 18 for this test scenario. The needed raw metrics for the scenario (based on which the composite metrics are computed) should also be defined in the CAMEL model.
Steps to execute scenario	Using appropriate tool (possibly REST CLIENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment Plan reasoning 5. Deployment Plan-based application reconfiguration. 6. Deployment to the selected cloud provider 7. Measurements for the composite metrics of given type (see the table with data for this test scenario) should be generated and collected by Executionware (i.e., stored in the TSDB on respective node where the application component is deployed). These measurements may be stored in the CDO and respective subscribers might be informed about them via the Metric Collector. Such measurements should be stored in CDO only when they map to SLOs or other relevant objects defined in CAMEL.
Expected results	1. The measurements of the considered (raw and composite) metrics should be properly and correctly calculated (by, e.g., considering the measurement schedule and window), according to input parameters for the given test case. 2. These measurements should be stored in the TS database(s) on those VMs in which the respective component of the application to be measured resides. 3. Such measurements are possibly stored in the CDO and some subscribers (e.g., Meta Solver) might be informed about them.

In the Table 18, the reference data for the Test scenario 2.3 are provided. The scenario 2.3 should be executed with the metrics presented in that table. Also, more specific use case application metrics could be used, like number of users or response time. Such metrics will participate in the detailed description of the respective test cases which will be prepared in JIRA.

Table 18 Reference data for the Test scenario 2.3

Composite metric name	Metric description	Metric unit
AverageCpuUsage	Average CPU usage in 5 minutes window.	Real value, percent of usage.
AverageMemoryUsage	Average memory usage in 5 minutes window.	Real value, MB.
AverageDiskIoRead	Average Disk IO read transfer per second in 5 minutes window.	Real value, MB per second.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

AverageDiskIoWrite	Average Disk IO write transfer per second in 5 minutes window.	Real value, MB per second.
AverageFreeDiskSpace	Average free disk space in 5 minutes window.	Real value, MB.
AverageRxBytes	Average Network read transfer per second in 5 minutes window.	Real value, MB.
AverageTxBytes	Average Network write transfer per second in 5 minutes window.	Real value, MB/second.

Table 19 Test scenario 2.4

Test scenario Id	2.4
Name	Event generation
Scenario group	Metric management testing scenarios
Components to be tested	• Meta Solver • Metric Collector • Cloudiator • REST CLIENT • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider is integrated with the Melodic platform for which the user providers his/her credentials. 3. Cloudiator properly connected to given Cloud Providers
Input data	1. CAMEL model of simple application (described in test scenario 1.1) which includes the definition of events mapping to conditions on metrics which are also specified in this model. The events to test are defined in data Table 20 for this test scenario (*see below).*
Steps to execute scenario	Using appropriate tool (possibly REST CLIENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start the application deployment For each step, the status of the executed action should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. Upload of the Provider Model into CDO 2. Upload of the Application model

	3. Profiling execution 4. Reasoning execution using CP Solver 5. Adaptation execution 6. Deployment to the selected cloud provider 7. Events of given types (see the table with data for this test scenario) should be generated by Executionware and collected by Metric Collector.
Expected results	1. Events should be generated and provided to Meta Solver. Also, they should be stored in database.

In the Table 20, the reference data for the Test scenario 2.4 are provided. The scenario 2.4 should be executed with the events presented in that table. The detailed parameter of the window size will be provided in each test cases, depends on use case application and testing conditions.

Table 20 Reference data for the Test scenario 2.4

Id.	Event name	Event description
A	AverageCpuUsage above 70%	Average CPU usage in X minutes window above 70%.
B	AverageMemoryUsage above 85%	Average memory usage in X minutes window above 85%.
C	AverageCpuUsage below 50%	Average CPU usage in X minutes window below 50%.
D	AverageMemoryUsage below 65%	Average memory usage in X minutes window below 65%.
E	High usage	Conjunction of events A and B which should cause application scaling out.
F	Low usage	Conjunction of events C and D which should cause application scale-in.

3.3 Local reconfiguration testing scenarios

This section presents scenarios related to the local reconfiguration scenario group. Local reconfiguration means that reconfiguration of the application or its parts occurs in a certain cloud and is based on the scalability rules defined in SRL in the CAMEL model of the application. For this type of scenario, the selected modules of Upperware and Executionware are tested (SRL adapter, Cloudiator). Before the execution of the scenarios listed below, the scenarios described in section 3.2 should be executed with positive outcome.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 21 Test scenario 3.1

Test scenario Id	3.1
Name	Scale out application
Scenario group	Local reconfiguration testing scenarios
Components to be tested	• SRL adapter • Cloudiator
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider is integrated with the Melodic platform for which the user has provided his/her credentials. 3. Cloudiator properly connected to given Cloud Provider(s). 4. Application with SRL rules of scaling out and in (application components) in CAMEL is configured and installed properly via the Melodic platform (using for example Test Scenario 1.1 with proper CAMEL SRL configuration). The scale-out SRL rule has been defined as follows: if average CPU usage on all deployed nodes in a 5-minute windows interval is more than 70%, then one additional node of the sole application component is added to current application configuration.
Input data	1. No input data, see Prerequisites, Point 4.
Steps to execute scenario	1. It is possible to execute this scenario in two ways: either generate artificial CPU usage on given/all nodes via script invocation or stress the (sole) component of the application.
Actions performed by the system	1. Collection of raw measurements over CPU usage. 2. Composite metric with average CPU usage in 5-minute window should be generated by the Executionware based on the raw CPU utilisation measurements collected. 3. The event to scale should be generated by Cloudiator. 4. SRL Adapter should receive event to scale out the application. 5. Executionware API should be invoked by the SRL Adapter to scale the application. 6. One node of application on a separate virtual machine instance should be added.
Expected results	1. One new application component instance on a separate virtual machine (instance) should be periodically added every 5 minutes. 2. Up to 4 application component instances should be added in around 20 minutes.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 22 Test scenario 3.2

Test scenario Id	3.2
Name	Scale in application
Scenario group	Local reconfiguration testing scenarios
Components to be tested	• SRL adapter • Cloudiator
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider integrated with Melodic platform for which the user has supplied his/her credentials. 3. Cloudiator is properly connected to given Cloud Providers.
Input data	1. Application with SRL rules of scaling out and in (application components) in CAMEL is configured and installed properly on Melodic platform (using for example Test Scenario 1.1 with proper CAMEL SRL configuration). A SRL rule in CAMEL is defined as remove 1 instance of sole component of the application, down to 1 instance, if average CPU usage on all deployed nodes in 5 minutes window is less than 50% (this rule should be already defined in deployed CAMEL).
Steps to execute scenario	1. Execute scenario 3.1 to deploy 4 nodes via scaling out rules.
Actions performed by the system	1. Measurements over raw CPU usage should be generated by Executionware and collected by Metric Collector. 2. Measurements over mean CPU usage should be generated by Executionware and collected by Metric Collector. 3. The event for scaling should be generated by Executionware. 4. SRL adapter should receive event to scale in the application. 5. SRL adapter should invoke Executionware API to remove one application component instance. 6. One application component instance on a separate virtual machine (instance) should be removed by the Executionware.
Expected results	1. One previously deployed application component instance should be removed each time (i.e., each time the SRL rule of scale in is triggered). 2. Finally, all application instances except one should be removed.

3.4 Global reconfiguration testing scenarios

This section presents scenarios related to the global reconfiguration scenario group. Global reconfiguration is the reconfiguration of the application at a global scope where a new solution is

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

applied globally for the whole application and not only its specific Clouds (in contrast to local reconfiguration). Such a reconfiguration is applied mainly by the Upperware component group and especially the solvers in the presence of a contextual change (for example, new metric measurements, provider offering modifications, etc.).

Table 23 Test scenario 4.1

Test scenario Id	4.1
Name	Attributes of used VM offerings changed
Scenario group	Global reconfiguration testing scenarios
Components to be tested	• CP Generator • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • Cloudiator • ESB • BPM • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider integrated with Melodic platform for which the respective user credentials have been supplied. 3. Cloudiator properly connected to given Cloud Providers. 4. There exists at least one cloud provider with at least one offering satisfying the requirements posed by the user. 5. CAMEL model of each Cloud Provider prepared and uploaded to the platform with at least two virtual machine offerings provided each with different cost parameter. 6. Complete CAMEL model of a simple application (which includes the definition of one component and its installation/maintenance scripts). The sole application is installed as a Unix process (no container) in one virtual machine. An example application could be the Apache web server deployed using standard installation commands. The CAMEL model should include an optimisation requirement which states that the overall application cost should be minimized. 7. Application with the definition of optimization requirement is installed by executing test scenario 1.1. The virtual machine offering with the smallest cost should be chosen.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Input data	1. The updated CAMEL model of one cloud provider with changed cost parameter per virtual machine offer concerned. The changed VM offering cost should cause the production of a new optimal solution of the application to be validated and enforced by the adapter.
Steps to execute scenario	1. Upload of the updated CAMEL model of cloud provider.
Actions performed by the system	1. The change in CAMEL model requires to start the deployment/Upperware flow from the beginning in order to modify the CP model considered with the exception that we do not deal with a totally new deployment solution, but a reconfiguration of an existing one. 2. Finding of the new solution should be performed. 3. Reconfiguration of the application should be executed in order to be deployed on the newly selected type of virtual machine.
Expected results	1. Application should be reconfigured to use a different type of virtual machine according to newly founded deployment and optimization plan. The previous virtual machine (instance) should be terminated. 2. Application should work properly; this means that its web page should be properly displayed (continuing the previous example with an Apache web server).

Table 24 Test scenario 4.2

Test scenario Id	4.2
Name	Global reconfiguration testing
Scenario group	Global reconfiguration testing scenario
Components to be tested	• CP Generator • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • Cloudiator • ESB • BPM • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider integrated with Melodic platform for which the respective user credentials have been supplied. 3. Cloudiator properly connected to given Cloud Providers. 4. There exists at least one cloud provider with at least two offering satisfying the requirements posed by the user.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

	5. CAMEL model of each Cloud Provider prepared and uploaded to the platform with at least two virtual machine offers provided with different cost parameter.
Input data	1. Complete CAMEL model of a simple application (which includes the definition of one component and its installation/maintenance scripts). The sole application is installed as a Unix process (no container) in one virtual machine. An example application could be the Apache web server deployed using standard installation commands. The CAMEL model should include definition of events and metrics needed to execute the particular test case.
Steps to execute scenario	This is generic reconfiguration scenario. There should be defined detailed test cases in JIRA for different reconfiguration tests, according to the use case applications needs. Steps to execute scenario: 1. Upload updated CAMEL model. 2. Execute actions needed to generate reconfiguration event (for example extra load generated on the deployed VM).
Actions performed by the system	1. Reconfiguration of application should be executed according to defined events and metrics.
Expected results	1. Application should be reconfigured according to defined SLOs. 2. Application should work properly; this means that its web page should be properly displayed (continuing the previous example with an Apache web server).

3.5 Reasoning related testing scenarios

This section presents scenarios related to the reasoning scenario group. Reasoning maps to the capability to find an optimal deployment solution for the application at hand based on the requirements that have been posed for it. The test scenarios are focused on isolated tests of each particular solver. For this scenario type, mostly the Upperware modules are tested (CP generator, Meta Solver, CP Solver, MILP Solver, LA Solver).

Table 25 Test scenario 5.1

Test scenario Id	5.1
Name	Linear constraints and optimization solving – CP Solver
Scenario group	Reasoning

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Components to be tested	CP Solver
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts.
Input data	1. CP model of given optimization problem – as described in Table 28 with reference data *(see below)* for a two-component application. CP model should be uploaded to CDO.
Steps to execute scenario	Using SOAP UI tool, execute the following step: 1. Invoke CP Solver REST API exposed on ESB and pass prepared CDO path to CP model as an input parameter. Method invocation result should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. CP Solver solves the given problem.
Expected results	1. The optimal solution is produced. 2. The CP model is updated in CDO 3. In CP Solver log, the main constraint problem and its optimal solution found should be logged.

Table 26 Test scenario 5.2

Test scenario Id	5.2
Name	Linear constraints and optimization solving – MILP Solver
Scenario group	Reasoning
Components to be tested	MILP Solver
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts.
Input data	1. CP model of given optimization problem – as described in the Table 28 with reference data. Model should be uploaded to CDO.
Steps to execute scenario	Using SOAP UI tool, execute the following step: 1. Invoke MILP Solver REST API exposed on ESB and pass prepared CDO path to CP model as parameters. Method invocation result should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. MILP Solver solves the problem.
Expected results	1. Optimal solution is found 2. CP model is updated in CDO. 3. In MILP Solver log, the constraint problem and its solution derived are logged.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 27 Test scenario 5.3

Test scenario Id	5.3
Name	Linear constraints and optimization solving – LA Solver
Scenario group	Reasoning
Components to be tested	LA Solver (LA Orchestrator)
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts.
Input data	1. CP model of given optimization problem – as described in the Table 28 with reference data.
Steps to execute scenario	Using SOAP UI tool, execute the following step: 1. Invoke LA Orchestrator REST API exposed on ESB and pass prepared CP model as parameters. Method invocation result should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. LA Solver should find the optimal solution for the given problem.
Expected results	1. Optimal solution is found 2. CP model is updated in CDO. 3. In LA Solver log, the main constraint problem and its optimal solution found should be logged.

In the *Table 28*, the reference data for the Test scenarios 5.1, 5.2, and 5.3 are provided. The scenarios 5.1, 5.2 and 5.3 should be executed with the respective VM offers, constraints, and utility function combinations presented in the table.

Table 28 Reference data for the Test scenarios 5.1, 5.2, and 5.3

Available VM offers	Constraints	Utility function	Expected solution
1. Cores=2, RAM=8, loc=EU, Cost=130, Provider A 2. Cores=4, RAM=16, loc=US, Cost=200, Provider B 3. Cores=4, RAM=32, loc=EU, Cost=250, Provider A 4. Cores=8, RAM=32, loc=EU, Cost=350, Provider A	1. Cores>2 2. Loc=eu	Min(Cost)	Chosen VMs: 3, 3

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

1. Cores=2, RAM=8, loc=EU, Cost=100, Provider A 2. Cores=4, RAM=16, loc=US, Cost=200, Provider B 3. Cores=4, RAM=32, loc=EU, Cost=250, Provider A 4. Cores=8, RAM=32, loc=EU, Cost=350, Provider A	1. Mem>4	Min(Cores)	Chosen VMs: 1,1
1. Cores=2, RAM=8, loc=EU, Cost=100, Provider A 2. Cores=4, RAM=16, loc=US, Cost=200, Provider B 3. Cores=4, RAM=32, loc=EU, Cost=250, Provider A 4. Cores=8, RAM=32, loc=EU, Cost=310, Provider A	1. VM.1.Cores>4 2. VM.2.Mem<32 3. VM.2.Mem>8 4. Loc in (EU, US)	Min(Cost)	Chosen VMs: 4,2

Table 29 Test scenario 5.4

Test scenario Id	5.4
Name	Non-linear constraints and optimization solving – CP Solver
Scenario group	Reasoning
Components to be tested	CP Solver
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts.
Input data	1. CP model of given non-linear optimization problem – as described in Table 31 with reference data (see below).
Steps to execute scenario	Using SOAP UI tool, execute the following step: 1. Invoke CP Solver REST API exposed on ESB and pass prepared CP model as parameters. Method invocation result should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. CP Solver should find the optimal solution for the given problem.
Expected results	1. Optimal solution is found 2. CP model is updated in CDO. 3. In CP Solver log, the constraint problem and its derived solution are logged

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 30 Test scenario 5.5

Test scenario Id	5.5
Name	Non-linear constraints and optimization solving – LA Solver
Scenario group	Reasoning
Components to be tested	LA Solver (LA Orchestrator)
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts.
Input data	1. CP model of given non-linear optimization problem – as described in Table 31 with reference data (see below).
Steps to execute scenario	Using SOAP UI tool, execute the following step: 1. Invoke LA Orchestrator REST API exposed on ESB and pass prepared CP model as parameters. Method invocation result should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. LA Solver should find the optimal solution for the given problem.
Expected results	1. Optimal solution is found 2. CP model is updated in CDO. 3. In LA Solver log the solution for the problem.

In Table 31, the reference data for the Test scenarios 5.4 and 5.5 are provided. The scenarios 5.4 and 5.5 should be executed with all VM offers, constraints, and utility function combinations presented in the table.

Table 31 Reference data for the Test scenarios 5.4 and 5.5

Available VM offers	Constraints	Utility function	Expected solution
1. Cores=2, RAM=8, loc=EU, Cost=100, Provider A 2. Cores=4, RAM=16, loc=US, Cost=200, Provider B 3. Cores=4, RAM=32, loc=EU, Cost=250, Provider A 4. Cores=8, RAM=32, loc=EU, Cost=350, Provider A	Mem/Cores>7 Loc=eu	Min(Cost)	Chosen VMs: 3,3

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

1. Cores=2, RAM=8, loc=EU, Cost=100, Provider A 2. Cores=4, RAM=16, loc=US, Cost=200, Provider B 3. Cores=4, RAM=32, loc=EU, Cost=250, Provider A 4. Cores=8, RAM=32, loc=EU, Cost=350, Provider A	Mem/Cores>3	Min(Cores)	Chosen VMs: 1,1
1. Cores=2, RAM=8, loc=EU, Cost=100, Provider A 2. Cores=4, RAM=16, loc=US, Cost=200, Provider B 3. Cores=4, RAM=32, loc=EU, Cost=250, Provider A 4. Cores=8, RAM=32, loc=EU, Cost=300, Provider A	VM.1.Cores*Cost>2400 VM.2.Mem<32 VM.2.Mem>8 Loc in (EU, US)	Min(Cost)	Chosen VMs: 4,2

3.6 API testing scenarios

In this section, we present test scenarios related to the API Testing scenario group. In this group, only the APIs exposed by the Melodic system to the external modules and systems are tested. This implies that there are no tests of internal methods of components, which are not exposed on ESB. Due to the nature of the API tests, the simplified presentation of the test scenarios has been prepared. It allows for a more comprehensive and compact presentation of the test scenarios (as such scenarios usually execute just one step – the call to the respective API method of just one component – thus it is trivial to represent this step in a very detailed manner). Additional methods might be exposed to the API in case the use case demonstrators face this need.

Table 32 Test scenarios related to the API Testing scenario group

API Method	Prerequisites	Input parameters	Expected results
Camel model upload	Melodic platform installed	Valid CAMEL Provider model	CAMEL Provider Model uploaded into CDO
Camel model upload	Melodic platform installed	Non-valid CAMEL Provider model	Error "Invalid model" raised
Camel model upload	Melodic platform installed CAMEL Provider model(s) uploaded	Valid CAMEL Application model	CAMEL Application Model uploaded into CDO

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Camel model upload	Melodic platform installed CAMEL Provider model(s) uploaded	Invalid CAMEL Application model	Error "Invalid model" raised
Initiate deployment process	Melodic platform installed CAMEL Provider model(s) uploaded CAMEL Application model uploaded	Name of existing application (i.e., mapping to the CAMEL Application model uploaded) was supplied.	Application deployed.
Initiate deployment process	Melodic platform installed CAMEL Provider model(s) uploaded CAMEL Application model uploaded	Invalid application name	Error "Non-existing application" raised.
Initiate deployment process	Melodic platform installed CAMEL Provider model(s) uploaded CAMEL Application model uploaded Application deployed.	Valid application name	Error "Application already deployed" raised.
Get application status	Melodic platform installed CAMEL Provider model(s) uploaded CAMEL Application model uploaded	Valid application name	Status "Application model uploaded" received.
Get application status	Melodic platform installed CAMEL Provider model(s) uploaded CAMEL Application model uploaded Reasoning finished Application deployed	Valid application name	Status "Application deployed" received.
Get application status	Melodic platform installed CAMEL Provider model(s) uploaded CAMEL Application model uploaded MELODIC platform's deployment workflow in progress	Valid application name	Status "Application deployment in progress: Reasoning" or "Application deployment in progress: Deploying" received.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Get application status	Melodic platform installed CAMEL Provider model(s) uploaded CAMEL Application model uploaded Application deployed.	Invalid application name	Status "Inexistent application" received.
Get application status	Melodic platform installed CAMEL Provider model(s) uploaded CAMEL Application model uploaded Application deployed.	Invalid application name	Status "Inexistent application" received.

3.7 UI testing scenarios

In this section, test scenarios at the User Interface (UI) level are presented. The following User Interfaces (UIs) for Melodic will be considered at this moment:

- Web based UI dedicated to present a read-only view of the CAMEL model of an application and its deployment status.
- Eclipse[10] based editor and Web based editor of CAMEL, which supports the editing of the CAMEL model.

Current features of the existing UIs are covered by test scenarios. Moreover, the UI related requirements described in D5.02 *"Updates to OSS frameworks"* are also covered. Based on these scenarios, test cases for the UIs should be prepared. Furthermore, it is planned to extend and unify different UI components. To this end, the test cases for this unification can be completed once the respective unified UI component becomes available and its detailed functionality is known.

Due to the nature of the UI tests and the currently not fully specified requirements, a simplified presentation of the UI test scenarios has been prepared, following the same simplified table structure used for the description of the API testing scenarios above. Such a structure allows for a comprehensive and compact presentation of the test scenarios.

[10] https://www.eclipse.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 33 Web based UI for application view:

Operation/view	Prerequisites	Input parameters	Expected results
Application view	Melodic platform installed CAMEL Application model uploaded CAMEL Provider model(s) uploaded Logged user	None	The view of all relevant elements of the application are presented. Application is not deployed to any Cloud Provider.
Deployment view	Melodic platform installed CAMEL Application model uploaded CAMEL Provider model uploaded Application deployed to Cloud Providers Logged user	None	The view of all relevant elements of the application are presented with deployed location. Application is deployed to Cloud Provider.

Table 34 Eclipse based editor of the CAMEL:

Operation/view	Prerequisites	Input parameters	Expected results
CAMEL Model validation	Eclipse editor properly configured and run.	Valid CAMEL Application model with suitable models/elements of the CAMEL language.	There should not be any syntax error highlighted. The XMI file should be properly generated once the user presses Ctrl-S.
CAMEL Model validation	Eclipse editor properly configured and run.	Invalid CAMEL Application model	For each invalid element of CAMEL, the appropriate syntax error with proper message should be presented. The XMI file should not be generated irrespectively of how many times Ctrl-S is pressed.
Syntax completion	Eclipse editor properly configured and run.	Valid CAMEL Application model	For each element, input completion should be presented according to the CAMEL specification.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

3.8 Data Management Testing Scenarios

Big data applications and data management are key elements of the Melodic system. The test scenarios will cover the following topics:

- Big data application deployment optimization
- Big data application deployment execution
- Big data application monitoring and reconfiguration
- Data locality awareness – features related to data locality and data movement.

4 Non-functional testing requirements

This chapter specifies testing requirements collected using the methodology described in section 1.5.2, related to the non-functional requirements of Melodic in form of non-functional testing scenarios. The description of each scenario includes the scope of testing, the requirements for testing and the expected results from the scenario execution.

Similar to the case of the functional testing scenarios, the non-functional testing scenarios are divided into scenario groups. Each scenario group contains scenarios related to a particular type of non-functional tests, for which the described non-functional requirements should be fulfilled.

The description of each scenario is similar to the scenarios related to functional testing requirements and includes the prerequisites, input data, steps to execute each scenario and the expected results.

4.1 Fault handling testing scenarios

This section presents scenarios related to the Fault handling scenario group. Fault handling maps to the reliability of the system and the ability to properly handle technical failures, crashes and external system inaccessibility. The group's test scenarios present the most common situations and focus on the verification of the (application) deployment process, the global and local reconfiguration as well as the deployment reasoning, thus involving and focusing on all components of the Melodic system.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 35 Test scenario 6.1

Test scenario Id	6.1
Name	Temporary unavailability of Melodic platform components
Scenario group	Fault handling
Components to be tested	• CP Generator • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • ESB • BPM • Cloudiator • REST CLIENT • CDO Server
Prerequisites	• Installed and configured Melodic platform, without any application related artefacts. • At least one cloud provider has been integrated with the Melodic platform, and a user has supplied respective credentials for this provider. • Meta Solver configured to use CP Solver for that case. • Cloudiator properly connected to given Cloud Provider
Input data	• Complete CAMEL model of a simple application (which includes the definition of one component and its installation/lifecycle management scripts). The simple, one-component application is installed as a UNIX process (no container) in one VM. An example application would be the Apache web server installed using a standard installation command. • CAMEL model of given Cloud Provider prepared with at least one virtual machine offer included.
Steps to execute scenario	For each execution of the scenario, one of the following components (scenario should be executed once per each stopped component) is stopped manually to simulate component unavailability: • CP Generator • Meta Solver • CP Solver • LA Solver • Solver to deployment • Adapter/Plan Generator • Metric Collector • SRL adapter • Cloudiator After manually stopping the respective component, the following steps can be executed using appropriate tools (for example REST client), where the

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

	placement/order of Step 4 depends on which component in the Melodic platform is affected (e.g., it is the 4th step, if it concerns a component of the Upperware module): 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start manually stopped component 5. Start application deployment For each step, the status of the executed action should be positive.
Actions performed by the system	The following actions should be executed in the system, where the actions are not listed in exact order of occurrence as this also depends on the placement of the affected (i.e., stopped and restarted) component in the Melodic architecture and its actual order of execution in the deployment workflow. In particular, the action to restart a component takes place in between a pair of stated actions and requires the re-execution of the first step in the action pair. For instance, if the Profiler fails, then we detect this when it is called (i.e., at step 3 and before step 4 is executed thus mapping to the action pair mentioned in previous sentence), we restart the component and then we re-execute the profiling execution action (i.e., step 3) before moving to the reasoning execution one. 1. Uploading of the Provider Model 2. Uploading of the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment Plan reasoning 5. Deployment Plan-based application reconfiguration. 6. Deployment to the selected cloud provider Proper retrying handling should be performed, meaning that the operation of invoking stopped component should be retried after the manual start of the component.
Expected results	The following results are to be produced: • Second attempt to invoke stopped components should be successful. • Proper error message with information about stopped component inaccessibility should be logged. • A VM (instance) on the selected Cloud Provider should be created • The simple application should be installed on that VM (instance) • The application should be run properly (for example, the web server's web page should be displayed properly)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 36 Test scenario 6.2

Test scenario Id	6.2
Name	Temporary unavailability of BPM - verifying proper system behaviour after BPM recovery.
Scenario group	Fault handling
Components to be tested	• CP Generator • Meta Solver • CP Solver • LA Solver • Solver to deployment • Adapter/Plan Generator • ESB • BPM • Metric Collector • SRL adapter • Cloudiator • REST CLIENT • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider has been integrated with Melodic platform, while the user has supplied his/her credentials for this provider. 3. Meta Solver configured to use CP solver for that case. 4. Cloudiator properly connected to given Cloud Provider 5. BPM component is stopped
Input data	1. Complete CAMEL model of a simple application (which includes the definition of one component and its installation/maintenance scripts). The simple, one-component application should be installed as a Unix process (no container) in one VM. An example application would be the Apache web server installed using a standard installation command.
Steps to execute scenario	The goal of the scenario is to verify the situation when the BPM component is down and then it is recovered. The system should return to normal behaviour and the process of deployment of application should be resumed. Using appropriate tool (for example REST CLIENT), the following steps are executed: 1. Ensure that the BPM component is down. Upload Cloud Provider definition 2. Upload Application model 3. Start overall application deployment process 4. Start BPM 5. Start reasoning process 6. Start deploying of the one-component application For each step, the status of the executed action should be positive.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Actions performed by the system	The following actions should be executed in the system: 1. Upload of the Provider Model 2. Upload of the CAMEL Application model 3. Proper retrying handling should be done, the process should start after starting the BPM component, all further steps of process should be executed as described below. 4. Offer filtering and deployment optimisation model generation 5. Deployment Plan reasoning 6. Deployment Plan-based application reconfiguration. 7. Deployment to the selected cloud provider
Expected results	1. Proper error message with information about BPM inaccessibility should be logged. Proper message with the availability of BPM should be logged. 2. A VM (instance) on the selected Cloud Provider should be created 3. The simple application should be installed on that VM (instance) 4. The application should be run properly (for example, the web server's web page should be displayed properly)

Table 37 Test scenario 6.3

Test scenario Id	**6.3**
Name	Temporary unavailability of Cloud Provider
Scenario group	Fault handling
Components to be tested	• CP Generator • Meta Solver • CP Solver • LA Solver • Solver to deployment • Adapter/Plan Generator • ESB • BPM • Metric Collector • SRL adapter • Cloudiator • REST CLIENT • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. One cloud provider has been integrated with Melodic platform, while the user has supplied his/her credentials for this provider. 3. Meta Solver configured to use CP solver for that case. 4. Cloudiator properly configured with given Cloud Provider

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

	5. No network connection to the Cloud Provider - the lack of network connection will be simulated by changes of routing table or firewall rules configuration on the machine where Cloudiator is installed.
Input data	1. Complete CAMEL model of a simple application (which includes the definition of one component and its installation/maintenance scripts). The simple, one-component application should be installed as a Unix process (no container) in one VM. An example application could be an Apache web server installed using a standard installation command. 2. CAMEL model of given Cloud Provider has been prepared with at least one virtual machine offering included. 3. There should be a proper configuration of the virtual machine both in Camel Provider model and on the Cloud Provider side
Steps to execute scenario	Using appropriate tool (for example REST CLIENT), execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start deploying of application 5. Check in logs that Adapter starts to invoke Executionware. 6. Resume the network connection to the Cloud Provider after first deployment attempt on the cloud of this provider has failed For each step, the status of the executed action should be positive.
Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment Plan reasoning 5. Deployment Plan-based application reconfiguration. 6. Deployment to the selected cloud provider 7. Proper retrying handling should be done (one or more times), the deployment to the Cloud Provider should be executed after resuming connection with Cloud Provider.
Expected results	1. Proper error message with information about Cloud Provider inaccessibility should be logged. After that proper message about availability of Cloud Provider should be logged. 2. A VM (instance) on the selected Cloud Provider should be created 3. The simple application should be installed on that VM (instance) 4. The application should be run properly (for example, Apache web server's web page should be displayed properly)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 38 Test scenario 6.4

Test scenario Id	6.4
Name	High Availability Component configuration
Scenario group	Fault handling
Components to be tested	• CP Generator • Meta Solver • CP Solver • LA Solver • Solver to deployment • Adapter/Plan Generator • ESB • BPM • Metric Collector • SRL adapter • Cloudiator • REST CLIENT • CDO Server
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. High Availability (HA) configuration of key components of the Melodic platform (see requirement with Id 1, 'High Reliability' in Table 1, section 2.2); each of the following components should be installed with two instances, with HA configuration on ESB: • CP Generator • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator • ESB 3. At least one cloud provider has been integrated with the Melodic platform while the user has supplied his/her credentials for this provider. 4. Meta Solver has been configured to use the CP solver for that case. 5. Cloudiator properly connected to given Cloud Provider
Input data	Complete CAMEL model of a simple application (which includes the definition of one component and its installation/maintenance scripts). The simple, one component application should be installed as a Unix process (no container) in one virtual machine. An example application could be an Apache web server installed using a standard installation command. CAMEL model of given Cloud Provider prepared with at least one virtual machine offering included. There should be a proper configuration of the virtual machine both in the Camel Provider model and on the Cloud Provider side

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Steps to execute scenario	For each execution of the scenario, one of the instances of the listed component should be stopped: • CP Generator • Meta Solver • CP Solver • Solver to deployment • Adapter/Plan Generator After stopping the respective component, the appropriate tools (for example REST CLIENT) should be used to execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model 3. Start reasoning process 4. Start stopped component 5. Start deploying of application For each step, the status of the executed action should be positive.
Actions performed by the system	The following actions should be executed in the system (the actions are not listed in order of occurrence, due to stopping various components it is not possible to present them in order of occurrence): 1. Uploading of the Provider Model 2. Uploading of the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment Plan reasoning 5. Deployment Plan-based application reconfiguration. 6. Deployment to the selected cloud provider 7. Proper retrying handling should be done, the operation of invoking stopped component should be retried after the start of the component. The actions are listed more or less in order of occurrence with the exception that the action to restart a component takes place in between a pair of stated actions and requires the re-execution of the first step in the action pair. For instance, if the Profiler fails, then we detect this when it is called, we restart the component and then we re-execute the profiling execution action before moving to the reasoning execution one.
Expected results	1. Proper error message with information about fall-back due to component inaccessibility should be logged. After that, proper message about availability of the component should be logged. 2. A VM (instance) on the selected Cloud Provider should be created 3. The simple application should be installed on that VM (instance) 4. The application should be run properly (for example the web server's web page should be displayed properly).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

4.2 Performance testing scenarios

To perform the Melodic middleware testing, it is required to understand the whole Melodic eco-system aiming at improving the users' existing Cloud experience. During performance measuring scenarios, the Melodic business objectives related to performance are primarily given priority. In general, performance testing of Cloud-applications is done similarly to web applications. In performance testing of a web application, it is up to the tester to decide the performance related parameters (mainly latency and throughput), based on specific user provided requirements. However, it must be kept in mind that application performance is dependent on user's perception (e.g., for a latency sensitive web-application, a lower response time is desirable). As the types of such web applications are different, the user requirements could also be differentiated. For instance, throughput can be measured based on the application category: e.g., for a certain type of application, throughput could be the number of jobs processed in a unit time, or for a service-based application, throughput could also be the number of user requests processed per unit of time. In general, some common testing parameters are:

- user serving capacity
- number of user jobs processed in a unit time (throughput testing),
- application robustness,
- system availability.

During the middleware performance testing phase, most of the above-mentioned scenarios are valid, and the impact of scalability on performance should be verified. It should be done by executing the same test benchmarks on the system using the given configuration while scaling the resources. The performance tests should cover the complete application deployment process on a general level as well as its related components on the component's level. For Cloud environments, support for dynamic scaling is also an important issue because offering "less elasticity" could pose a direct threat to the user data integrity (data loss or data destroyed could happen). During the workload test, it is always recommended to load the entire system realistically, so that the real-time usage scenario could be reproduced. The whole measurement process is conceived in a way so that the Melodic platform can work efficiently regardless of application types.

Finally, this section presents all scenarios related to the Melodic middleware platform testing. Based on the above analysis, we will focus on:

a. performance of core components while solving complex optimization problems,
b. dynamic scaling of the system,
c. resources required to run a single Melodic instance.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 39 Test scenario 6.5

Test scenario Id	6.5
Name	Response time while solving complex optimization problems
Scenario group	Performance Testing
Components to be tested	• CP-Generator • CDO Server • Meta Solver • CP Solver • Solver to deployment • Adapter
Prerequisites	1. Installed and configured Melodic without any application related artefacts. 2. Meta Solver has been configured to use the CP solver. 3. One public cloud service provider (CSP) has been integrated with the Melodic platform. 4. Cloudiator is also connected to the given CSP
Input data	1. Complete CAMEL model of an application (such application needs a number of components, potentially with individual VM requirements) which has a huge number of VM requirements of different VM types while satisfying various amount of user given constraints to minimize the overall deployment related cost. 2. CAMEL model of given Cloud Service provider prepared with few VMs offering included.
Steps to execute scenario	For each execution of the scenario, one of the instances of the following components should be executed: CP Generator, Meta Solver, CP Solver, Solver to deployment, Adapter. Next, the following steps should be completed without any error: 1. Upload CSP definition and also Application model. 2. Deploy application on selected VMs 3. Proper logging mechanism (such as execution time) for each Melodic component involved in the testing phase.
Actions performed by the system	The following actions should be executed: 1. Uploading of the Provider model and Application model. 2. Offer filtering and CP model generation. 3. Deployment Plan Reasoning.
Expected results	1. It is needed to log the total response time of the Upperware with the information related to the optimised problem instance. The relevant information related to the optimisation problem could for example be size of the problem instance (such as total number of variables, constraints). 2. It is also needed to log the execution time of all core components so that if any bottleneck exists, they could be identified.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 40 Test scenario 6.6

Test scenario Id	6.6
Name	Dynamic scalability within one Cloud - verification of the execution time
Scenario group	Performance Testing
Components to be tested	• CP-Generator, • CDO Server, • Meta Solver, • CP Solver, • Solver to deployment, • Adapter (SRL adapter), • ESB, BPM, • Metric Collector, • Cloudiator, • REST CLIENT.
Prerequisites	1. Installed and configured Melodic without any application related artefacts. 2. Meta Solver has been configured to use the CP solver. 3. One public cloud service provider (CSP) has been integrated with the Melodic platform. 4. Cloudiator is also connected to the given CSP.
Input data	1. Complete CAMEL model of an application that frequently changes its VM requirements. 2. CAMEL model of given Cloud provider prepared with at least one virtual machine offering included. 3. There should be a proper configuration of the virtual machine both in Camel provider model and on the Cloud provider side.
Steps to execute scenario	The goal of the scenario is to verify the execution times of each component while scaling the application within one Cloud Provider. For each execution of the scenario, one of the instances of the following components should be executed: CP Generator, Meta Solver, CP Solver, Solver to deployment, Adapter. Next, the following steps should be completed without any error: 1. Upload CSP definition and Application model. 2. Deploy application on selected VMs 3. Proper logging mechanism (such as execution time) for each Melodic component involved in the testing phase. 4. Log the information related to generated/used VMs (optional).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Actions performed by the system	The following actions should be executed: 1. Upload the Provider Model 2. Upload the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment Plan reasoning 5. Deployment Plan-based application reconfiguration. 6. Deployment to the selected cloud provider
Expected results	1. The application should come to completion successfully with proper logging (including the errors) information. 2. The execution time of each Melodic component involved has been logged. 3. Another file printing the ID-list of all used VMs including their IPs, VM triggering time by the Cloud provider, and VM boot time (time taken by CSP to start a VM). This information can help to compare the VM generation speed by other Cloud service providers.

Table 41 Test scenario 6.7

Test scenario Id	6.7
Name	Dynamic scalability testing for multi-Cloud feature (using two different locations)
Scenario group	Performance Testing
Components to be tested	• CP-Generator • CDO Server • Meta Solver • CP Solver • Solver to deployment • Adapter (SRL adapter) • ESB • BPM • Metric Collector • Cloudiator • REST client.
Prerequisites	1. Installed and configured Melodic without any application related artefacts. 2. Meta Solver has been configured to use the CP solver. 3. One public cloud service provider (CSP) has been integrated with the Melodic platform. 4. Cloudiator is also connected to the given CSP.
Input data	1. Complete CAMEL model of an application (such application might need a number of components with VM requirements each). 2. CAMEL model of given CSP prepared with few VM offerings included. 3. There should be a proper configuration of the virtual machine, both in Camel provider model and on the Cloud provider side.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Steps to execute scenario	For each execution of the scenario, one of the instances of the following components should be executed: CP Generator, Meta Solver, CP Solver, Solver to deployment, Adapter. Next, the following steps should be completed without any error: 1. Upload CSP definition and Application model. 2. Deploy application on selected VMs 3. Proper logging mechanism (such as execution time) for each Melodic component involved in the testing phase. 4. Log the information related to generated/used VMs (optional).
Actions performed by the system	The following actions should be executed: 1. Upload the Provider Model 2. Upload the Application model 3. Offer filtering and deployment optimisation model generation 4. Deployment Plan reasoning 5. Deployment Plan-based application reconfiguration. 6. Deployment to the selected cloud provider
Expected results	1. The application should come to completion successfully with proper error logging information. 2. (Optional) It would also be good to log the execution time of each Melodic component involved. 3. (Optional) Another sample file printing the ID-list of all used VMs including their IPs, VM triggering time by the Cloud provider, and VM boot time (time taken by CSP to start a VM). This information can help to compare the VM generation speed by other Cloud service providers.

Table 42 Test scenario 6.8

Test scenario Id	6.8
Name	Counting Compute Resource Overhead of Melodic introduced over its host machine
Scenario group	Performance Testing
Components to be tested	• CP-Generator • CDO Server • Meta Solver • CP Solver • Solver to deployment • Adapter (SRL adapter) • ESB • BPM • Metric Collector • Cloudiator • REST client

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Prerequisites	1. Installed and configured Melodic without any application related artefacts. 2. Meta Solver has been configured to use the CP solver 3. Both Upperware components and Cloudiator will be tested.
Input data	1. Complete CAMEL model of a simple one component application in one VM. 2. CAMEL model of given CSP prepared with at least one VM offering included. 3. There should be a proper configuration of the VM both in the Camel provider model and on the CSP side
Steps to execute scenario	For each execution of the scenario, one of the instances of the following components should be executed: CP Generator, Meta Solver, CP Solver, Solver to deployment, Adapter. Next, the following steps should be completed without any error: 1. Upload CSP definition and Application model. 2. Deploy application on selected VMs 3. For both scenarios, a proper logging mechanism (such as execution time) for each Melodic component involved in the testing phase.
Actions performed by the system	The following actions should be executed in two steps to quantify the extra resource consumption by running a Melodic instance: Step1: 1. Uploading of the Provider model and Application model. 2. Offer filtering and deployment of optimized model. 3. Performing plan reasoning, plan-based application reconfiguration 4. Deployment to selected cloud providers. Step2: In this scenario, now manually deploy the same user application in another VM (same size, type). Using both the steps, it should be possible to quantify the extra overhead introduced by MELODIC over the second user VM.
Expected results	1. Both the user applications should come to completion successfully with proper error logging information. 2. A log file should print the CPU usage, memory usage, and network/disk I/O usage for comparison purpose. For both cases, a separate log file should be created.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

4.3 Security testing scenarios

This section presents scenarios related to testing security in the Melodic system and in communication with external systems. Security, for the purpose of these tests, means authentication as well as authorization of methods invocation between components, especially methods exposed on ESB, and usage of secure, cryptographically protected communications protocols. In this chapter, only the basic security related test scenarios are described. The authentication could be different for different ways of component method invocation, but authorization should use a unified, common mechanism. The security topic, requirements, design and testing scenarios related to advanced security will be covered in more detail in deliverable D5.03 *"Security requirements & design"*.

This group contains fundamental scenarios related to Security testing on the Melodic platform.

For the first release of Melodic, security is handled by different components (user related by the Cloudiator UI, intercomponent security by the ESB); for the second and third release, a unified way of handling security is planned. It will be reported in deliverable D5.3 *"Security requirements & design"*.

Table 43 Test scenario 7.1

Test scenario Id	7.1
Name	Method invocation by programmatic access – Successful Authentication
Scenario group	Security Testing
Components to be tested	• Security (Authentication) Service[8] • ESB • Upperware, Executionware
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. Among others this involves: a. the provisioning of security related configuration data b. ESB configuration to require authentication (e.g. OAuth2.0) c. ESB connected to a properly configured directory service (e.g. OpenLDAP) 2. At least one cloud provider has been integrated with the Melodic platform; the user credentials for this provider should have also been supplied. 3. Cloudiator properly connected to the relevant Cloud Provider(s) 4. Valid credentials per component
Input data	1. Complete CAMEL model of simple application comprising the definition of one application component and its installation/lifecycle management scripts which should be installed as a Unix process (no container) in one virtual machine.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

	2. CAMEL model of given Cloud Provider prepared and registered in the Melodic platform with at least one virtual machine offer being provided.
Steps to execute scenario	Using appropriate tool to execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model in CAMEL 3. Start reasoning process 4. Start the application deployment For each step, the corresponding component should be successfully authenticated and the status of the executed action should be positive in order to be able to move to the next step.
Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model in CAMEL 3. Reasoning 4. Execute deployment plan 5. Deploy new solution Every action involves first the successful authentication of the respective component.
Expected results	1. Authentication success per component is logged 2. A certain virtual machine on the selected Cloud Provider should be created 3. The sole component (e.g., web server) of the simple application should be installed on that virtual machine 4. The application should be run properly (for example, the root web page of a web server should be displayed properly)

Table 44 Test scenario 7.2

Test scenario Id	7.2
Name	Unsuccessful authentication
Scenario group	Security Testing
Components to be tested	• ESB • Security (Authentication) Service • Upperware
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. Among others this involves: a. the provisioning of credentials per MELODIC component b. provisioned credentials for Adapter are invalid c. ESB configuration to require authentication (e.g. OAuth2.0) d. ESB connected to a properly configured directory service (e.g., OpenLDAP)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

	2. At least one cloud provider has been integrated with the Melodic platform; the user credentials for this provider should have also been supplied. 3. Cloudiator properly connected to the relevant Cloud Provider(s)
Input data	1. Complete CAMEL model of simple application comprising the definition of one application component and its installation/lifecycle management scripts which should be installed as a Unix process (no container) in one virtual machine. 2. CAMEL model of given Cloud Provider prepared and registered in the Melodic platform with at least one virtual machine offer being provided. 3. Scenario should be repeatedly executed with a different component each time having invalid credentials.
Steps to execute scenario	Using appropriate tool to execute the following steps: 1. Upload Cloud Provider definition 2. Upload Application model in CAMEL 3. Start reasoning process 4. Receive authentication failure For each step, the corresponding components should be successfully authenticated (except the one with invalid credentials) and the status of the executed action should be positive in order to move to the next step/component in the flow. For the component with invalid credentials, authentication to Security Service fails and the deployment workflow terminates after having reported the failure.
Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model in CAMEL 3. Deployment workflow: a. Components before component with invalid credentials are authenticated and successfully connected to ESB b. The attempt of the component with invalid credentials to connect to ESB fails 4. Report authentication failure
Expected results	1. Authentication success per component (except the component with invalid credentials) is logged 2. Authentication failure for component with invalid credentials is logged 3. No virtual machine should be created

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 45 Test scenario 7.3

Test scenario Id	7.3
Name	Successful Authorisation Request
Scenario group	Security Testing
Components to be tested	• Authorisation service • Adapter • ESB
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider has been integrated with the Melodic platform; the user credentials for this provider should have also been supplied. 3. An XACML policy template, allowing the application deployment (corresponds to Adapter authorisation) when current time is after a parametric threshold. 4. Cloudiator properly connected to the relevant Cloud Provider(s)
Input data	1. Complete CAMEL model of simple application comprising the definition of one application component and its installation/lifecycle management scripts which should be installed as a Unix process (no container) in one virtual machine. 2. CAMEL model of given Cloud Provider prepared and registered in the Melodic platform with at least one virtual machine offer being provided.
Steps to execute scenario	Using appropriate tool to execute the following steps: 1. Generate a concrete XACML policy using as time threshold the current time minus 20 minutes (since the policy dictates that the request time should be after the time threshold, this step will guarantee that the Adapter will receive an authorisation permit result). 2. Load XACML policy into authorisation service policy repository. 3. Upload Cloud Provider definition 4. Upload Application model in CAMEL 5. Start reasoning process 6. Start the application deployment (since this was automatically authorized, by the respective service, based on the current time that is subsequent to the time threshold of the policy) For each step, the status of the executed action should be positive in order to move to the next step of the process.
Actions performed by the system	The following actions should be executed in the system: 1. Uploading of the Provider Model 2. Uploading of the Application model in CAMEL

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

	3. Loading and activation of XACML policy 4. Reasoning 5. Action graph generation (in Adapter) 6. Policy-based action graph elements' authorisation. a. Each action graph element is checked against authorization policy b. Each one of them is successfully authorised (i.e., has a permit result) 7. Report authorisation permit 8. Trigger Cloudiator execution 9. Deploy new solution
Expected results	1. PERMIT authorisation decision is logged 2. An authorization permit event (produced by Security Service/Authorisation) sent to ESB 3. A certain virtual machine on the selected Cloud Provider should be created 4. The sole component (e.g., web server) of the simple application should be installed on that virtual machine 5. The application should be run properly (for example, the root web page of a web server should be displayed properly)

Table 46 Test scenario 7.4

Test scenario Id	**7.4**
Name	Unsuccessful authorisation request
Scenario group	Security Testing
Components to be tested	• Security Service/Authentication • Adapter • ESB
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. At least one cloud provider has been integrated with the Melodic platform 3. An XACML policy template, allowing the application deployment (corresponds to Adapter authorisation) when current time is after a parametric threshold. 4. Cloudiator properly connected to the relevant Cloud Provider(s)
Input data	1. Complete CAMEL model of simple application comprising the definition of one application component and its installation/lifecycle management scripts which should be installed as a Unix process (no container) in one virtual machine. 2. CAMEL model of given Cloud Provider prepared and registered in the Melodic platform with at least one virtual machine offer being provided.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Steps to execute scenario	Using appropriate tool to execute the following steps: 1. Generate a concrete XACML policy using as time threshold the current time plus 20 minutes ahead in time (since the policy dictates that the request time should be after the time threshold, this step will guarantee that the Adapter will receive an authorisation deny result). 2. Load XACML policy into authorization service policy repository. 3. Upload Cloud Provider definition 4. Upload Application model in CAMEL 5. Start reasoning process For each step, the status of the executed action should be positive. In order to move to the next step of the process. In this testing scenario, the process ends with a successful reasoning that it doesn't result in the initiation of the deployment actions since these failed to be authorised (since the deployment actions were about to be performed before the time threshold set in the policy).
Actions performed by the system	The following actions should be executed in the system: 1. Loading and activation of XACML policy 2. Uploading of the Provider Model 3. Uploading of the Application model in CAMEL 4. Reasoning 5. Action graph generation (in Adapter) 6. Policy-based action graph elements' authorisation. a. Each action graph element is checked against authorisation policy b. At least one such authorisation fails (i.e. deny result) since the deployment action was about to be performed before the time threshold set in the policy. 7. Report authorisation deny
Expected results	1. DENY decision is logged 2. An authorisation deny event is sent to ESB 3. No virtual machine should be created

Table 47 Test scenario 7.5

Test scenario Id	7.5
Name	Unsuccessful user authorisation with administrator privileges
Scenario group	Logging testing scenarios
Components to be tested	• MELODIC platform administration

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts.
Input data	
Steps to execute scenario	1. Log into the MELODIC administration platform as administrator 2. Provide a new user profile with all mandatory information and without administration rights 3. Submit the new user 4. Get temporary password 5. Try to log in administration platform with new user profile
Actions performed by the system	1. Add new user 2. Generate temporary password
Expected results	1. Access denied to the new user due to lack of administration privileges.

Table 48 Test scenario 7.6

Test scenario Id	7.6
Name	Logging within MELODIC platform
Scenario group	Unified administration procedures testing scenarios
Components to be tested	MELODIC platform administration
Prerequisites	1. Installed Melodic platform, with or without any application related artefacts. 2. Cloudiator properly connected to the given Cloud Provider
Input data	1. Component name
Steps to execute scenario	1. Login as administrator of the MELODIC installation 2. Find logs for relevant components in the corresponding area
Actions performed by the system	1. Maintain logs 2. Create viewable or downloadable text
Expected results	1. All relevant logs can be accessed

4.4 Other non-functional testing scenarios

This section presents further testing scenarios for non-functional requirements that are not covered by the testing scenarios described in scenario groups from section 4.1 to 4.3, like backup and recovery, platform user addition/removal, monitoring, logging, administration and maintenance tasks – aspects which are relevant to guarantee the reliability and recoverability of the system as well as the overall administration. The test scenarios from this group are described

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

in tables which have the same format as most of the previously described scenarios (for example scenarios in section 4.1). For the first Melodic release only the Cloudiator UI is available, so the testing of user management in this release is limited to the Cloudiator UI. For the next releases of Melodic, the method of user authentication and authorisation will be implemented (as it will be designed in D5.3 *"Security requirements & design"*). It will replace Cloudiator UI in the context of user management.

Table 49 Test scenario 8.1

Test scenario Id	8.1
Name	Adding user
Scenario group	User management testing scenarios
Components to be tested	• Melodic platform • Security Services (in the 1st release user management will be handled by Cloudiator; for further releases by a dedicated Security Component and ESB)
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts.
Input data	
Steps to execute scenario	1. Log into the Melodic platform as administrator 2. Provide new user information (login, attributes, role) according to MELODIC management rules 3. Submit the new user
Actions performed by the system	1. User addition 2. Temporary password generation
Expected results	1. New user profile should be active and according to role 2. Temporary password generated

Table 50 Test scenario 8.2

Test scenario Id	8.2
Name	Removing user
Scenario group	User management testing scenarios
Components to be tested	• Melodic platform • Security Services (in the 1st release user management will be handled by Cloudiator; for further releases by a dedicated Security Component and ESB)
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Input data	1. One user id or user email
Steps to execute scenario	1. Log in as administrator 2. Select one user to be removed 3. Delete this user
Actions performed by the system	1. User deletion
Expected results	1. User profile should be deleted

Table 51 Test scenario 8.3

Test scenario Id	8.3
Name	Updating user password
Scenario group	User management testing scenarios (in the 1st release user management will be handled by Cloudiator; for further releases by a dedicated Security Component and ESB)
Components to be tested	• Melodic platform • Security Services
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts.
Input data	One user id or user email
Steps to execute scenario	1. Log in as administrator 2. Select one user 3. Generate new temporary password
Actions performed by the system	1. Update user password with new temporary one
Expected results	1. New temporary password

Table 52 Test scenario 8.4

Test scenario Id	8.4
Name	Updating user profile
Scenario group	User management testing scenarios (in the 1st release user management will be handled by Cloudiator; for further releases by a dedicated Security Component and ESB)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Components to be tested	• Melodic platform • Security Services
Prerequisites	Installed and configured Melodic platform, without any application related artefacts.
Input data	1. User profile data
Steps to execute scenario	1. Log in as administrator 2. Select one user 3. Select information to be updated (email, first name, last name, role etc.) 4. Provide updated information 5. Submit information update
Actions performed by the system	1. Update user profile information
Expected results	1. User profile information should be updated

Table 53 Test scenario 8.5

Test scenario Id	8.5
Name	Unified starting, stopping and restarting of MELODIC platform
Scenario group	Unified administration procedures testing scenarios.
Components to be tested	• CP-Generator, • CDO Server, • Meta Solver, • CP Solver, • Solver to deployment, • Adapter (SRL adapter), • ESB, BPM, • Metric Collector, • Cloudiator,
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. Application is configured and installed properly on Melodic platform (using for example Test Scenario 1.1 (Table 3) with proper CAMEL SRL configuration). 3. Meta Solver configured to use CP solver for that case. 4. Cloudiator properly connected to the given Cloud Provider 5. Virtual machine on the selected Cloud Provider should be created 6. The sole component (e.g., web server) of the simple application should be installed on that machine 7. The application should be run properly (for example the root web page of a web server should be displayed properly)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Input data	
Steps to execute scenario	1. Start the Melodic platform. 2. Deploy application 3. Stop the Melodic platform 4. Restart Melodic platform 5. Get the status of deployed application
Actions performed by the system	1. Components of the Melodic platform are restarted.
Expected results	• All Melodic components should always have the same status (STARTED) as a final outcome of the scenario. • Status is STARTED (after starting or restarting platform). The retrieved status of the deployment should indicate that the deployment continues as planned.

Table 54 Test scenario 8.6

Test scenario Id	8.6
Name	Configuring backup
Scenario group	Unified administration procedures testing scenarios
Components to be tested	Melodic platform
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. Application is configured and installed properly on Melodic platform (using for example Test Scenario 1.1 (Table 3) with proper CAMEL SRL configuration).
Input data	
Steps to execute scenario	1. Login as administrator of the MELODIC installation 2. Select suitable drive for backup 3. Select backup components 4. Configure schedule for backup
Actions performed by the system	1. Store drive information 2. Store component selection 3. Store backup schedule
Expected results	Backup configuration is stored

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Table 55 Test scenario 8.7

Test scenario Id	8.7
Name	Executing backup
Scenario group	Unified administration procedures testing scenarios
Components to be tested	Melodic platform
Prerequisites	1. Installed and configured Melodic platform, without any application related artefacts. 2. Application is configured and installed properly on Melodic platform (using for example Test Scenario 1.1 (Table 3) with proper CAMEL SRL configuration). 3. Execution of 8.6 test scenario.
Input data	
Steps to execute scenario	1. Login as administrator of the MELODIC installation 2. Start backup process
Actions performed by the system	1. Execute backup periodically
Expected results	1. Backup is properly performed according to the right schedule. Backup files stored according to the backup configuration

Table 56 Test scenario 8.8

Test scenario Id	8.8
Name	Recover Melodic platform
Scenario group	Unified administration procedures testing scenarios
Components to be tested	Melodic platform
Prerequisites	1. Freshly installed Melodic platform, without any application related artefacts. 2. Meta Solver configured to use CP solver for that case. 3. Cloudiator properly connected to the given Cloud Provider 4. Virtual machine on the selected Cloud Provider should be created 5. The sole component (e.g., web server) of the simple application should be installed on that machine 6. The application should be run properly (for example the root web page of a web server should be displayed properly) 7. Execution of the test scenarios 8.6 and 8.7

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Input data	1. Backup files of first fully configured installation (the result of execution of the 8.7 test scenario).
Steps to execute scenario	1. Stop first Melodic installation 2. Install Melodic platform on a new VM, without any application related artefacts. 3. Start recovery process with previously stored back up files of initial platform configuration/deployment/installation. 4. Start the new Melodic platform. 5. Log in as administrator of the new Melodic platform 6. Monitor deployed application
Actions performed by the system	1. Stop first Melodic installation 2. Install a second Melodic platform on a new VM 3. Execute recovery process with backup files on this second platform 4. Start the second Melodic platform.
Expected results	1. New Melodic platform is equivalent to original one and has control over the already deployed application

Table 57 Test scenario 8.9

Test scenario Id	8.9
Name	Monitor Melodic platform
Scenario group	Unified administration procedures testing scenarios
Components to be tested	Melodic platform
Prerequisites	1. First installed Melodic platform, with or without any application related artefacts. 2. Cloudiator properly connected to the given Cloud Provider
Input data	
Steps to execute scenario	1. Login as administrator of the MELODIC installation 2. Access monitored system parameters
Actions performed by the system	1. Monitor system health 2. Aggregate and deliver monitoring results
Expected results	1. Monitoring results can be accessed

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

5 Summary

Integration is the process of combining different parts of a software system. The term itself is overused and can refer to very different notions, such as:

a. integrating the changes from different developers in the same code base as used for instance in Continuous Integration[11]

b. integrating various executable software artefacts in one larger system as used in Enterprise Integration Patterns [3].

This document presented detailed requirements for integrating the various components of the Melodic platform. It has based its outcome on the system specification supplied in D2.1 *"System specification"*. This has led to the identification of 13 integration requirements for the Data and the Control Flow Plane, and three requirements for the Monitoring Plane which were collected based on a specific methodology. The methodology of collecting integration requirements is described in section 1.3.2.

Testing is the process of verifying that a system is working as intended and expected. Testing usually occurs on several levels of the system, ranging from unit tests to acceptance tests. Based on the requirements collected using the methodologies described in section 1.4.2 and section 1.5.2, this deliverable defines a series of testing requirements for functional and non-functional testing. They are specified in the form of testing scenarios. Such scenarios cover:

- initial application deployment (functional, nine sub-scenarios),
- global reconfiguration (functional, two sub-scenarios),
- local reconfiguration (functional, two sub-scenarios),
- metric management (functional, four sub-scenarios),
- reasoning-related (functional, five sub-scenarios),
- APIs (functional, eleven sub-scenarios),
- user interfaces (functional, seven sub-scenarios),
- fault-handling (non-functional, four sub-scenarios),
- performance (non-functional, eight sub-scenarios),
- security (non-functional, four sub-scenarios),
- other (non-functional, seven sub-scenarios).

Scenarios described in the deliverable could be further extended according to new features which will be introduced to the platform. Correspondingly, new test cases will evolve and expand in conjunction with the evolution of the Melodic platform.

[11] http://searchsoftwarequality.techtarget.com/definition/continuous-integration

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

Based on this deliverable, all Melodic releases will be tested (including the *integration* release (D5.07), the *platform prototype* release (D5.08), and the *final* release (D5.09)). In addition, the integration requirements of this deliverable will influence the Melodic final framework to be described in deliverable D2.3 *"Final framework and external APIs"*.

The test scenarios designated in this deliverable will be updated during the project lifetime and enriched with new test cases such that, in the end, a very robust and complete framework will be delivered.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664

6 References:

[1] Liqiang Chen, "Integrating Cloud Computing Services Using Enterprise Service Bus (ESB)". Business and Mgmt. Res. Vol. 1, No. 1, 2012, http://sciedu.ca/journal/index.php/bmr/article/download/674/387

[2] Mehdi Bahrami and Mukesh Singhal, "The Role of Cloud Computing Architecture in Big Data", Chapter 13 in "Information Granularity, Big Data, and Computational Intelligence", W. Pedrycz and S.- M. Chen (eds.) Vol. 8, Springer, 2015

[3] Gregor Hohpe and Bobby Woolf, "Enterprise Integration Patterns", 11st Edition, ISBN: 978-0321200686, Addison-Wesley. 2004.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731664