Melodic
Big data cloud

Title:

Final framework and external APIs

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 -
31 January 2020

www.melodic.cloud

Deliverable reference:
D2.3

Date:
31 January 2020

Responsible partner:
Simula Research Laboratory

Editor(s):
Kyriakos Kritikos, Feroz Zahid

Author(s):
Feroz Zahid, Yiannis Verginadis, Ioannis Patiniotakis, Daniel Baur, Daniel Seybold, Christos Chalaris, Theodora Mavrodopoulou, Gregoris Mentzas, Geir Horn, Amirhosein Taherkordi, Kyriakos Kritikos, Marta Rozanska, Pawel Skrzypek, Marcin Prusinski

Reviewed By:
Paweł Szkup, Tomasz Przeździęk

Approved by:
Tomasz Przeździęk

ISBN number:
N/A

Document URL:
http://www.melodic.cloud/deliverables/D2.3 Final framework and external APIs.pdf f

Abstract:

This document describes the final Melodic framework and the application programming interfaces (APIs) offered by the framework components. The APIs enables both the internal inter-component integration as well as third-party software integrations. With the help of the external interfaces, third party integrators and developers can write extensions for the Melodic platform to enhance existing component functionality or add new feature components.

We describe APIs offered by various components of the *Upperware* and *Executionware*, the two main modules of the Melodic middleware platform. An overview of the scope of potential third party extensions are also given for selected components. Finally, We also cover *CAMEL 2.0*, which is the updated version of the CAMEL modelling language that is being exploited in the project.

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

Document	
Period Covered	M6-30
Deliverable No.	D2.3
Deliverable Title	Final framework and external APIs
Editor(s)	Kyriakos Kritikos & Feroz Zahid
Author(s)	Feroz Zahid, Yiannis Verginadis, Ioannis Patiniotakis, Daniel Baur, Daniel Seybold, Christos Chalaris, Theodora Mavrodopoulou, Gregoris Mentzas, Geir Horn, Amirhosein Taherkordi, Kyriakos Kritikos, Marta Rozanska, Pawel Skrzypek, Marcin Prusinski
Reviewer(s)	Paweł Szkup, Tomasz Przeździęk
Work Package No.	2
Work Package Title	Architecture and Data Management
Lead Beneficiary	Simula Research Laboratory
Distribution	PU
Version	1.0
Draft/Final	Final
Total No. of Pages	89

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

Table of Contents

List of Figures 4
List of Tables 4
1 Introduction 5
1.1 Scope of the Document 5
1.2 Structure of the Document 6
2 The Melodic Platform 7
2.1 Component Integration 8
2.2 External Interfaces 8
3 Upperware Interfaces 10
3.1 CP Generator 10
3.2 MetaSolver 14
3.3 Optimization Solvers 20
3.4 CP Solver 21
3.5 LA Solver 23
3.6 Utility Generator 24
3.7 Solver to Deployment 26
3.8 DLMS 29
3.9 Adapter 52
3.10 Event Processing Management (EMS) 59
4 Executionware Interfaces 72
5 CAMEL 2.0 72
6 External Interfaces and Extensibility 77
6.1 Overview 77
6.2 Upperware 77
Adding a new optimization solver 77
Implementing a new DLMS algorithm 78
Supporting a new storage technology 79
Utility functions 80
6.3 Executionware 82

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

Supporting a new Cloud provider 82
Adding a new data processing framework 83
6.4 Extending CAMEL 83
7 Conclusions 88
References 89

List of Figures

Figure 1: Overview of the MELODIC platform architecture 7
Figure 2: The DLMS Controller interface 79
Figure 3: CAMEL update process 84

List of Tables

Table 1: The REST API offered by the CP Generator 10
Table 2: The REST APIs consumed by the CP Generator 11
Table 3: The REST APIs offered by the MetaSolver 15
Table 4: The REST APIs consumed by the MetaSolver 19
Table 5: The REST API offered by the CP Solver 21
Table 6: The REST API consumed by the CP Solver 22
Table 7: The REST API offered by Solver to Deployment 27
Table 8: The REST API consumed by Solver to Deployment 28
Table 9: The REST API offered by DLMS 29
Table 10: The REST API offered by the Adapter 53
Table 11: The REST APIs consumed by the Adapter 54
Table 12: The REST APIs offered by EMS 59
Table 13: The REST APIs consumed by EMS 69
Table 14: Overview of the coverage of CAMEL 2.0 73
Table 15: Comparison between the two CAMEL editors 75

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

1 Introduction

MELODIC has produced a multi-cloud platform which is able to deploy and dynamically provision data-intensive applications. This platform has been evolved in the context of this project's lifetime. This included updates to the internal architecture of the platform components and the gradual introduction of new but planned functionalities. Nevertheless, the overall platform architecture for MELODIC has not been modified, highlighting its proper design from the very beginning.

A multi-cloud platform is a software product that constantly changes and might require potential adjustments and enhancements before getting into the market. Fortunately, the MELODIC platform is already in the market, which witnesses its implemented ability to provide a robust and innovative, multi-cloud and data-intensive application management offering. However, this platform could be further improved in order to increase its added-value and further boost its market share. Thus, the goal of this deliverable is to highlight what are the main points of extension to this platform towards enhancing its functionality. Further, it attempts to detail the interfaces exposed by the MELODIC platform components in form of APIs to facilitate: (a) the understanding of how this platform operates at the interface layer; (b) based on this understanding, to enable the proper development or adoption of existing software products or modules and their integration with the MELODIC platform. Finally, CAMEL 2.0 is presented, which is the newest version of CAMEL that has been followed for the development of the latest releases of the MELODIC platform.

This deliverable plays a complementary role with respect to the ones in [1], [2], where the latter are dedicated to explaining the internal architecture of the main platform modules, i.e., the Upperware and Executionware. In particular, its focus is mainly on the interface layer and, thus, does not need to enter details about what is the internal architecture of a certain component and how it has been realised. However, the joining of the knowledge of all these deliverables will be the main knowledge and reference point for the whole platform, raising the level of its understanding as well as clarifying to the ultimate degree how the platform is functioning and how it can be extended.

1.1 Scope of the Document

This document is intended for the developers interested in learning about the interfaces of the Melodic components for extending the Melodic platform and providing third-party software integrations. Parts of the document require basic understanding of how Melodic works, and we refer readers to the Melodic deliverable D2.2 'Architecture and Initial Feature Definitions' [3] for more details about the architecture of the Melodic platform.

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

1.2 Structure of the Document

This deliverable is structured as follows. Chapter 2 provides an overview of the Melodic platform in terms of its high-level architecture, the way its components are integrated and which kinds of APIs it offers to its potential adopters. Chapters 3 and 4 detail the APIs of the two main modules of the Melodic platform, i.e., the Upperware and Executionware, respectively. Chapter 5 is dedicated to analysing the new version of CAMEL 2.0 which is exploited by the MELODIC platform both for application model editing as well as following the models@runtime paradigm. Chapter 6 explicates the main extension points in Upperware, Executionware as well as CAMEL Chapter 7 concludes this deliverable

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

2 The Melodic Platform

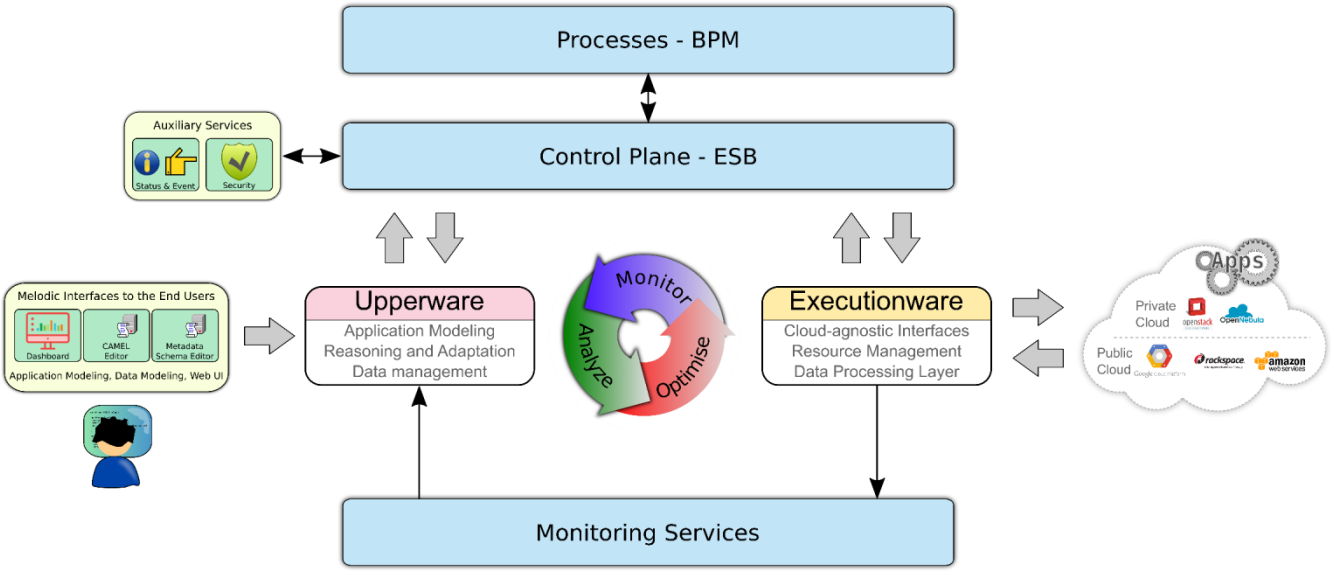

Figure 1: Overview of the MELODIC platform architecture

The Melodic platform is conceptually divided into three main component groups, the Melodic interfaces to the end users, the Upperware, and the Executionware. The Melodic interfaces to the end users include tools and interfaces used by the Melodic users to model their applications and datasets and interact with the Melodic platform. The Melodic modelling interfaces, through the *CAMEL* modelling language [4], provide a rich set of domain-specific languages (DSLs) which cover different modelling aspects, spanning both the design and the runtime of a Cloud application as well as data modelling traits. Applications and data models created through the modelling interfaces, in the form of CAMEL, are given as input to the Melodic Upperware. The job of the Upperware is to calculate the optimal data placements and application deployments on dynamically acquired Cross-Cloud resources in accordance with the specified application and data models in CAMEL by considering the current Cloud performance, workload situation, and costs. The actual Cloud deployments are carried out through the Executionware. The Executionware is capable of managing and orchestrating diverse Cloud resources, while also enables support for cross-cloud monitoring of the deployed applications. Besides the three main component groups, two auxiliary services, for enabling unified and integrated event notifications as well as warranting secure operations with the Melodic platform, respectively, have been also designed. An overview of the Melodic architecture is given in Figure 1.

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

2.1 Component Integration

To realise the Melodic platform, different Melodic components need to interact with each other and exchange information in an efficient and secure manner. Moreover, as the Melodic platform will be developed as an integration of the available open source technologies, while providing the required extensions for efficient cross-cloud data-intensive applications, efficient integration mechanisms are key to successful implementation.

The components in the Melodic platform are integrated through two separate integration layers, the Control Plane and the Monitoring Plane, each bringing its own set of unique requirements. In brief, the Control Plane is responsible for controlling actions within a cloud application management process (e.g., deployment), and, thus, is reliable and transactional. The Monitoring Plane, on the other hand, senses and aggregates a large amount of monitoring data and, thus, requires fast data transfer. Based on a detailed evaluation of the integration and adaptation requirements of each plane, a hybrid solution [3] with two different integration methods has been derived to ensure that the requirements of the two different planes are completely fulfilled.

The Control Plane implementation is based on an Enterprise Service Bus (ESB) architecture [5] with process orchestration achieved through Business Process Management (BPM). The ESB architecture utilises a centralised bus for message propagation between components. Components publish messages to the ESB, which are then forwarded to all subscribing components. BPM orchestration is used to orchestrate invocation of methods from underlying Melodic components in the context of cloud application management processes. To this end, ESB integration with BPM is a flexible integration method allowing both an easy modification to the cloud application management process workflows, as well as the reusability of services exposed by a given component in various processes and features of the system/platform [6]. For the Monitoring Plane, a queue based message broker has been employed ensuring fast message delivery [7].

2.2 External Interfaces

The Melodic platform features external interfaces which enable: (a) component integration in the context of application management processes; (b) third-party software and services to be integrated with it. As such, these interfaces enable the definition, enactment (data-intensive) and overall management of multi-cloud applications. Further, they could give rise to, e.g., building a more complete multi-/cross-cloud application management platform allowing to realise missing platform features as well as enhancing the platform's functionality to, e.g., enable interfacing with additional cloud providers. Due to adopting a service-oriented architecture with an ESB, the platform's external interfaces include not only UIs like those mentioned above (CAMEL editor,

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

metadata schema editor and dashboard) but also REST APIs that encapsulate the functionality of the platforms components.

These APIs are detailed in the next two chapters. The API analysis is facilitated through the use of tables which conform to a particular structure, covering the following information per each API operation:

- Operation's endpoint and HTTP method
- Input and output parameters
- Examples of a request and response that could be issued and given back when calling this API operation

Further, this analysis is conducted per each platform component by unveiling both the API that it exposes as well as the API operations it needs to consume and, thus, interact with (offered by other platform components). Such information is valuable according to the following directions:

- It sets the main integration points in form of API operations between the platform components
- Such integration points also unveil how external software could be integrated with suitable, corresponding platform components by unveiling which API is offered and consumed by a platform component, it enables third party developers or platform extenders to replace this platform component with an enhanced implementation of it in order to increase the added-value of the MELODIC platform.

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

3 Upperware Interfaces

The Upperware module in the Melodic platform allows the reasoning and adaptive provisioning of multi-/cross-Cloud applications. As mentioned in Chapter 2, Upperware introduces the necessary functionality for calculating the optimal data placements and application deployments across Cross-Cloud resources, abiding to requirements and constraints described in a CAMEL model. These core functionalities are offered by the following Melodic software components: i) CP Generator; ii) Metasolver; iii) CP Solver; iv) LA Solver; v) Utility Generator; vi) Solver to Deployment; vii) DLMS; viii) Adapter; and ix) EMS.

In the following sections, we go through each Upperware component and detail the interfaces that each supplies and consumes, respectively, in form of REST APIs. To be noted here that the consumed interfaces are either supplied by other Upperware components, the platform's process management system or the Executionware. A complete analysis of the Upperware modules can be found in the deliverables: i) D3.4 - Workload optimization recommendation and adaptation enactment [1] and ii) D3.5 - MELODIC Upperware [8].

3.1 CP Generator

The CP Generator is the component responsible for constructing the CP (optimisation) model that needs to be solved by the Solvers coordinated by the Meta-Solver so as to produce the optimal deployment plan of the user application. To this end, this component offers one API endpoint with one operation dedicated to the construction of this problem based on the user application's CAMEL model. On the other hand, it consumes two operation APIs: one from the Notification Service to clarify that the CP model construction has been finished and one from the Executionware module to fetch the node candidates matching the requirements of each application component. Table 1 showcases both the offered and consumed API operations of CP Generator.

Table 1: The REST API offered by the CP Generator

Operation Summary	**End Point:**	*/constraintProblem*
	HTTP Method:	POST
Description	This operation can be executed for constructing the CP model for a user application out of the application's CAMEL model and the current offerings from cloud providers.	

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

Parameters	*applicationId:* String [INPUT] – The identifier of the application *notificationURI:* String [INPUT] – The URI of an endpoint to send the result of Constraint Problem creation . *watermark:* Watermark [INPUT] – Watermark of the message. Stores information about origin of the message.

Request Example	<pre>{
	"applicationId": "FCR",
	"notificationURI": "/api/generator/constraintProblemNotification/5dbf4548-f9f1-11e7-a29c-02420a0a0004/ConstraintProblemNotification",
	"watermark": {
	"user": "mprusinski",
	"system": "Process",
	"date": "2018-01-15T12:41:11+0000",
	"uuid": "5dbf4548-f9f1-11e7-a29c-02420a0a0004"
	}
	}</pre>

Response Example	**HTTP status:** 200 no body

In Table 2 we explain the REST APIs consumed by CP Generator.

Table 2: The REST APIs consumed by the CP Generator

Operation Summary	End Point:	*/api/generator/constraintProblemNotification/{processId}/{subjectId}*
	HTTP Method:	POST
Description	This operation is used to communicate the result of the CP model creation (whether successful or not) to the Control Process. The	

	processId is a definition of the currently running process while the subjectId is the id of this request. **Offered by:** *Control Plane*
Parameters	*applicationId*: String [INPUT] – The identifier of the application *cdoResourcePath*: String [INPUT] – The path in the CDO server where CP generator produced models can be found. *result:* Result [INPUT] – The result of creating CP model. *watermark*: Watermark [INPUT] – Watermark of the message. Stores information about origin of the message.

Request Example	`{`
	`  applicationId = "FCR",`
	`  cdoResourcePath = "upperware-models/FCRApp1545987428463",`
	`  result = {`
	`  status = "SUCCESS"`
	`  },`
	`  watermark = {`
	`  user = "cpGenerator",`
	`  system = "cpGenerator",`
	`  date = "2018-12-28T08:57:21+0000",`
	`  uuid = "87e3c8fd-0a7e-11e9-af72-02420a0a0012"`
	`  },`
	`}`

Response Example	**HTTP status:** 200, no body

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

<table>
<tr><td rowspan="2">Operation Summary</td><td>End Point:</td><td>/api/generator/constraintProblemNotification/{processId}/{subjectId}</td></tr>
<tr><td>HTTP Method:</td><td>POST</td></tr>
<tr><td>Description</td><td colspan="2">This method is called per each application component to discover those cloud offerings (termed as node candidates) that suit all its requirements.
Offered by: Executionware</td></tr>
<tr><td>Parameters</td><td colspan="2">As described in Executionware API description
http://cloudiator.org/rest-swagger/#operation/findNodeCandidates</td></tr>
<tr><td>Request Example</td><td colspan="2"><pre>
[{
        "requirementClass": "hardware",
        "requirementAttribute": "ram",
        "requirementOperator": "GEQ",
        "value": "8100",
        "type": "AttributeRequirement"
    }, {
        "requirementClass": "hardware",
        "requirementAttribute": "ram",
        "requirementOperator": "LEQ",
        "value": "10072",
        "type": "AttributeRequirement"
    }, {
        "requirementClass": "hardware",
        "requirementAttribute": "cores",
        "requirementOperator": "GEQ",
        "value": "1",
        "type": "AttributeRequirement"
    }, {
        "requirementClass": "hardware",
        "requirementAttribute": "cores",
        "requirementOperator": "LEQ",
        "value": "1",
        "type": "AttributeRequirement"
    }, {
        "requirementClass": "location",
</pre></td></tr>
</table>

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

```
        "requirementAttribute": "geoLocation.country",
        "requirementOperator": "IN",
        "value": "MD, Europe, RO, Baltic_Region, PT, VA, DK,
SE, Balkan_Peninsula, FR, LI, Scandinavia, BG, LU, RU,
southern_Europe, MC, MK, Benelux, HR, FO, BE, PL,
northern_Europe, AT, SI, IS, BA, HU, LT, RS, UA, CH, CZ, SK,
LV, MT, SM, ME, AL, AD, EE, GR, IT, NO, eastern_Europe,
western_Europe, NL, BY, GB, DE, Iberian_Peninsula, FI, IE,
Nordic_Region, ES",
        "type": "AttributeRequirement"
    }, {
        "requirementClass": "cloud",
        "requirementAttribute": "type",
        "requirementOperator": "EQ",
        "value": "CloudType::PUBLIC",
        "type": "AttributeRequirement"
    }, {
        "constraint": "nodes->forAll(type =
NodeType::IAAS)",
        "type": "OclRequirement"
    }
]
```


3.2 MetaSolver

The Metasolver is the Upperware component used for coordinating and supporting the Constraint Problem (CP) solving process. Specifically, the Metasolver undertakes the task of selecting an appropriate solver for a given CP problem and subsequently verifying that the solution yielded by this solver is significantly better than the currently deployed one. According to that decision, a reconfiguration process, based on the new solution, is realised as a new application deployment topology, across multiple clouds.

The Metasolver offers a certain REST API, as shown in Table 3, which is supplied by one of its core sub-components named as REST Controller. This API is detailed in the following table. In summary, it includes operations for assigning solvers to a CP model, evaluating its solution as well as updating the configuration of event subscriptions and checking the Meta-Solver health status.

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

Table 3: The REST APIs offered by the MetaSolver

Operation Summary	End Point:	*/constraintProblemEnhancement*
	HTTP Method:	POST
Description	This method is called for the assignment of the right solver(s) to a certain CP model which is identified by the path to the (CDO) Models Repository.	
Parameters	*application-id*: String [INPUT/OUTPUT]— The identifier of the application *cdo-models-path*: String [INPUT] — The path in the Models Repository where models required can be found *result:* Notification result [OUTPUT] *status:* "SUCCESS", "ERROR" *errorCode:* Specific error code *errorDescription:* Description of an error *designated-solver [OUTPUT]:* cpsolver,lasolver, milpsolver, nonemilpsolver, none	

Request Example	`{`
	`  applicationId = "FCR",`
	`  cdoResourcePath = "upperware-models/FCRApp1545987428463"`
	`}`	

Response Example	`{`
	`  applicationId = "FCR",`
	`  result = {`
	`  status = "SUCCESS"`
	`  }`
	`  designated-solver = "cpsolver"`
	`}`	

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

Operation Summary	**End Point:**	*/solutionEvaluation*
	HTTP Method:	POST
Description	This method is called for evaluating the suitability of a CP model solution. If the evaluation result is affirmative, then a (re-)configuration of the user application will be launched.	
Parameters	*application-id*: String [INPUT/OUTPUT]— The identifier of the application *cdo-models-path*: String [INPUT] — The path in the Models Repository where models required can be found *evaluation-result* [OUTPUT]: positive, negative, error	

Request Example	`{`
	`applicationId = "FCR",`
	`cdoResourcePath = "upperware-models/FCRApp1545987428463"`
	`}`	

Response Example	`{`
	`applicationId = "FCR",`
	`evaluation-result = "positive"`
	`}`	

Operation Summary	**End Point:**	*/updateSolution*
	HTTP Method:	POST
Description	This method is called for updating the solution to a CP model which could lead to an application's reconfiguration.	
Parameters	*application-id*: String [INPUT/OUTPUT]— The identifier of the application *cdo-models-path*: String [INPUT] — The path in the Models Repository where models required can be found *evaluation-result* [OUTPUT]: positive, negative, error	

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

<table>
<tr><td>Request Example</td><td colspan="2">
<pre>
{
    applicationId = "FCR",
    cdoResourcePath = "upperware-models/FCRApp1545987428463"
   deployment-result = {
     status = "SUCCESS"
  }
}
</pre>
</td></tr>
<tr><td>Response Example</td><td colspan="2">
<pre>
{
    applicationId = "FCR",
    update-result = {
     status = "SUCCESS"
   }
}
</pre>
</td></tr>
<tr><td rowspan="2">Operation Summary</td><td>End Point:</td><td>/updateConfiguration</td></tr>
<tr><td>HTTP Method:</td><td>POST</td></tr>
<tr><td>Description</td><td colspan="2">This method is called for evaluating the suitability of a CP model solution. If the evaluation result is affirmative, then a (re-)configuration of the user application will be launched.</td></tr>
<tr><td>Parameters</td><td colspan="2">mvv: map [INPUT] – an associative array (i.e. map) where a current-config variable (contained in the deployed solution) is mapped to a CP metric in CP model. This mapping is used for copying the (actual) values of current-config. variables onto the corresponding CP metrics of CP model, in order to make them available to solvers for the next solving iteration.

subscriptions: array [INPUT] – an array of subscription objects. Each subscription object is used to instruct Metasolver, to connect to a specific event broker and subscribe to a certain event topic. Each subscription object contains the following information:

• topic: String [INPUT] – The name of event topic where Metasolver will subscribe
• url: String [INPUT] – the connection string of the event broker (including protocol, address, port and other connection parameters)</td></tr>
</table>

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

<table>
<tr><td></td><td colspan="2">

- *client-id:* String [INPUT] – An optional client name used to identify the Metasolver connection when listing the active connections of an event broker
- *type:* String [INPUT] – The type of events sent in this topic. Two values are expected:
  - 'MVV': meaning that values of the received events must be extracted and used to update CP model
  - 'SCALE': meaning that when an event is received, Metasolver must request a new reconfiguration iteration to start.

</td></tr>
<tr><td>Request Example</td><td colspan="2">

```
{
  'mvv': [
    <sol-var> : <cp-metric>,
    …
  ],
  'subscriptions':
    [ {
        'topic': <string>,
        'url': <string>,
        'client-id': <string>,
        'type': <string>
      },
      …
    ]
}
```

</td></tr>
<tr><td>Response Example</td><td colspan="2">Text/Plain string "OK"</td></tr>
<tr><td rowspan="2">Operation Summary</td><td>End Point:</td><td>/health</td></tr>
<tr><td>HTTP Method:</td><td>GET</td></tr>
<tr><td>Description</td><td colspan="2">Via this operation, the health status of the Meta-Solver can be obtained.</td></tr>
<tr><td>Parameters</td><td colspan="2">N/A</td></tr>
<tr><td>Request Example</td><td colspan="2">N/A</td></tr>
<tr><td>Response Example</td><td colspan="2">N/A</td></tr>
</table>

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

In Table 4, we explain the REST APIs consumed by Metasolver. In summary, this table summarizes operations dedicated to: (a) informing the control process that a new reconfiguration iteration must be started; (b) passing to EMS the application's CP model id.

Table 4: The REST APIs consumed by the MetaSolver

Operation Summary	End Point:	*/api/Metasolver/deploymentProcess*
	HTTP Method:	POST
Description	This operation is invoked in order to request starting a new reconfiguration iteration. **Offered By:** *Control Plane*	
Parameters	*application-id*: String [INPUT/OUTPUT]— The identifier of the *application-id:* String [INPUT] — The identifier of the application *use-existing-cp:* Boolean [INPUT] — Boolean flag indicating whether the current CP model must be reused *cdo-resource-path*: String — The path in the Models Repository where required CP model resides. Obligatory when useExistingCP=true *result:* [OUTPUT] *status:* "SUCCESS", "ERROR" *errorCode:* Specific error code *errorDescription:* Description of an error *process-id:* String [OUTPUT] — The process id in the Control Process	

Request Example	{
	applicationId = "FCR",
	use-existing-cp=true,
	cdoResourcePath = "upperware-models/FCRApp1545987428463",
	username = "…",
	password= "…"
	}	

Response Example	{ process-id = "FCR_1",	

	<pre>result = {
	status = "SUCCESS"
	}
	}</pre>	

Operation Summary	End Point:	*/cpModelJson*
	HTTP Method:	POST
Description	This operation is invoked in order to inform EMS about the application's CP model path Offered By: *EMS*	
Parameters	*cp-model-id:* String [INPUT]	

Request Example	<pre>{
	'cp-model-id', <string>
	}</pre>	

Response Example	Plain text value "OK"	

3.3 Optimization Solvers

The role of Melodic is to optimize the Cloud resources used by the managed application to maximize the *utility* of the application with respect to its owner. The utility can be multi-dimensional; for instance, the user may simultaneously want to minimize the cost of the Cloud usage and maximise the performance of the application. Furthermore, the utility can be context dependent, e.g., if it is required for the application to satisfy a given deadline, then application performance may be play a more important role in the application utility than cost the closer one gets to the deadline. Finally, there are constraints that must be respected for the optimal solution. For instance, there can be a limit on the cost budget for the application deployment.

The context dependency of the solution implies that the optimization problem must be re-solved whenever the operational context of the application changes. One context parameter can, for instance, be the number of simultaneous users of the application. This is a random number, and alternatively to solving the stochastic combinatorial optimization problem every time a user joins

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

or leaves the deployed application, one may use a stateful solver trying to optimize for the resources needed in order to serve the *average* or *expected* number of users.

The aim of any relevant Solver in deployment reasoning is to find the application *configuration* that maximises its utility. The benefit is that the solvers will assign the best possible resources needed by the application components, and the optimal number of instances needed for each component. The Melodic platform may accommodate any number of Solvers. The Solvers may use different methods and algorithms for solving the placement optimisation problem, such as constraint and linear programming or reinforcement learning.

Below we describe two solvers used in the Melodic framework, namely CP Solver and LA Solver, and present the APIs provided by each. It must be noted that to their assistance, the *Utility Generator* component, described in section 3.6, is responsible for calculating the *utility value* used as the optimization criterion for them.

3.4 CP Solver

The CP Solver is a constraint programming (CP) solver which is able to solve CP models that can contain also non-linear constraints as well as functions. To this end, this component has a simplified interface which is called each time a new CP model needs to be solved. Similarly, it also consumes just one API operation, dedicated to informing the control process about the result of solving the CP model assigned. The two API operations offered and consumed by this component are detailed in the following two tables, Table 5 and Table 6, respectively.

Table 5: The REST API offered by the CP Solver

Operation Summary	**End Point:**	*/constraintProblemSolution*
	HTTP Method:	POST
Description	This operation is called for producing a solution to the CP model via the CP Solver.	
Parameters	*applicationId*: String [INPUT] – The identifier of the application *cdoModelsPath*: String [INPUT] – The path in the CDO server where CP model can be found. *notificationURI*: String [INPUT] – The URI of an endpoint to send the result of Constraint Problem Solution operation.	

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

	watermark: Watermark [INPUT] — Watermark of the message. Stores information about origin of the message.

Request Example	{
	"applicationId": "FCR",
	"cdoModelsPath": "upperware-models/FCRApp1545987428463",
	"notificationURI": "/api/cpSolver/solutionNotification/87e3c8fd-0a7e-11e9-af72-02420a0a0012/ConstraintProblemSolutionNotification",
	"watermark": {
	"user": "Camunda",
	"system": "Camunda",
	"date": "2018-12-28T08:57:24+0000",
	"uuid": "87e3c8fd-0a7e-11e9-af72-02420a0a0012"
	}
	}

Response Example	**HTTP status:** 200 no body

Table 6: The REST API consumed by the CP Solver

Operation Summary	End Point:	*/api/cpSolver/ConstraintProblemSolutionNotification/{processId}/{subjectId}*
	HTTP Method:	POST
Description	This method is called to highlight the finishing of the solving of a CP model by the CP Solver to the control process. ***Offered By***: *Control Plane*	
Parameters	*applicationId*: String [INPUT] — The identifier of the application *cdoResourcePath*: String [INPUT] — The path in the CDO server where CP Solver updated models can be found. *result:* Result [INPUT] — The result of creating CP model. *watermark*: Watermark [INPUT] — Watermark of the message. Stores information about origin of the message.	

Request Example	<pre>{
	applicationId = "FCR",
	cdoResourcePath = "upperware-models/FCRApp1545987428463",
	result = {
	status = "SUCCESS"
	},
	watermark = {
	user = "CPSolver",
	system = "CPSolver",
	date = "2018-12-28T08:57:21+0000",
	uuid = "87e3c8fd-0a7e-11e9-af72-02420a0a0012"
	},
	}</pre>

Response Example	**HTTP status:** 200 no body

3.5 LA Solver

The Learning Automata (LA) solver is a *stateful* solver that uses reinforcement learning to learn the average best configuration for the constraint stochastic combinatorial problem at hand. It works by searching for better alternatives to the configuration already deployed. This is accomplished through a sequence of interactions with the Utility Generator where the utility of a proposed configuration will vary depending on the application's execution context. For instance, if the solver proposes twice the same configuration, it may receive different utilities because the number of application users have changed between the two utility evaluations. The goal is to pick the configuration producing the best *average* utility value. Once the LA solver can conclude that a configuration different from the currently deployed configuration yields a better average utility, this alternative configuration is proposed for deployment to the Metasolver, and the LA Solver re-starts the search from this.

This has two implications:

1. The solver needs to run continuously to ensure that it will, over time, observe all different execution contexts of the application;

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

2. The solution found by the LA solver must be deployed to ensure that the solver's view of the world when searching for better solutions corresponds with the real measurements of context metrics reported by the running application.

The LA Solver is written in C++ for performance reasons, and because it is based on a modular LA framework[1] in C++ developed and maintained by UiO. To facilitate its integration with Melodic, it is encapsulated by a Java component called LA Orchestrator that defines the variables, metrics, and constraints of the constraint model to be solved based on the CP-model derived from the CAMEL model of the application to deploy. The metric values are initiated with the average historical metric values if the application has already been deployed. Then, the model is compiled and linked with the LA Solver core and started as a stand-alone background application on the Melodic server.

The LA Solver subscribes to the metric values itself to ensure that the constraints always reflect the current context, and a configuration candidate to be evaluated will be sent back to the LA Orchestrator using inter process communication (IPC) based on ZeroMQ[2]. The LA Orchestrator component will then evaluate the proposed configuration using the Utility Generator and return the utility value to the LA Solver via IPC. This means that the communication and interfaces to the Metasolver and the Utility Generator are managed by the LA Orchestrator, and this component complies with the same interface as that of the CP-Solver presented in Table 5 and Table 6.

3.6 Utility Generator

The purpose of the Utility Generator is to evaluate each configuration, or solution candidate, examined by a solver. This is fundamentally done in two steps: The first step identifies the set of *Node Candidates* suitable for deploying the components of the given configuration. Remember that the configuration contains resource requirements of the application components, and the node candidates are generic virtual machine types the application owner has the accounts and rights to use for the deployment. The Node Candidate abstraction is important for Cloud providers that may offer a plethora of variants for the same virtual machine, in particular for academic and community Clouds where users may upload their own specialisations of standard image types. Seen from the application component to be deployed, all these specialisations may provide identical resources to the component, and they are therefore examples of the same Node

[1] https://bitbucket.org/GeirHo/la-framework/src/default/
[2] https://zeromq.org/

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

Candidate. The Node Candidate abstraction will therefore reduce the search space and speed up the selection.

The second step of the Utility Generator will calculate the utility *value* given the set of selected Node Candidates and other parameters in the configuration proposed by the solver, like the number of instances of a given component. This step may also include a reconfiguration penalty calculated by the Data Management Lifecycle System (DLMS) for the reconfiguration of the data placement implied by moving from the currently running configuration to the proposed, new configuration. Similarly, the Adapter will also calculate a reconfiguration penalty based on the complexity of the reconfiguration actions and possible application downtime induced by adapting the current deployment to the new configuration by stopping and starting virtual machines and other services.

Owing to the very high number of possible solutions candidates, i.e. new configurations, to be evaluated, the communication between the Solver and Utility Generator needs to be as fast as possible. For that reason, the Utility Generator does not expose any REST interface but is invoked directly as a java library instead.

The simple usage of the Utility Generator consists of two operations. In the following tables, we explain the interface of the Utility Generator library:

Calling the Utility Generator constructor (once):

`public UtilityGeneratorApplication(String camelModelFilePath, String cpModelFilePath, boolean readFromFile, NodeCandidates nodeCandidates, UtilityGeneratorProperties properties);`	
String camelModelFilePath,	the identifier of the application
String cpModelFilePath	the path to the CP Model
boolean readFromFile,	the information if the CAMEL and CP Model should be read from a file or from the CDO Models Repository
NodeCandidates nodeCandidates	the cache with Node Candidates
UtilityGeneratorProperties properties	the Utility Generator Properties

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

Calling evaluate method for each of produced Solutions:

`public double evaluate(Collection<VariableValueDTO> solution);`	
Collection<VariableValueDTO> solution	the solution of the Constraint Problem as a Collection of VariableValueDTO.

where VariableValueDTO is:

```
public class VariableValueDTO<T extends Number> {
    private String name;
    private T value;
}
```

The example of the Collection<VariableValueDTO>:

```
[cardinality_Component_App = 3 , provider_Component_App = 1,
cardinality_Component_DB = 1, provider_Component_DB = 0]
```

3.7 Solver to Deployment

This component has the main duty to receive the solution to a CP model and to produce a concrete application deployment (instance) model out of it in CAMEL. This model will then be transformed into a detailed deployment plan and be executed by the Adapter. To this end, due to the simplified functionality of this component, only one API operation is offered. Similarly, only one API operation is consumed so as to communicate the result of the CP-model-to-deployment-model transformation to the control process. These two API operations that are offered and consumed by this component are detailed in the following two tables, Table 7 and Table 8, respectively.

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

Table 7: The REST API offered by Solver to Deployment

Operation Summary	End Point:	*/applySolution*
	HTTP Method:	POST
Description	This operation is called to initiate the transformation of a CP model solution to a CAMEL deployment instance model.	
Parameters	*applicationId*: String [INPUT] – The identifier of the application *cdoModelsPath*: String [INPUT] – The path in the CDO server where CP model can be found. *notificationURI*: String [INPUT] – The URI of an endpoint to send the result of Apply Solution operation. *watermark*: Watermark [INPUT] – Watermark of the message. Stores information about origin of the message.	

Request Example	<pre>{
	"applicationId": "FCR",
	"cdoModelsPath": "upperware-models/FCRApp1545987428463",
	"notificationURI": "/api/solverToDeployment/applySolutionNotification/87e3c8fd-0a7e-11e9-af72-02420a0a0012/ApplySolutionNotification",
	"watermark": {
	"user": "Camunda",
	"system": "Camunda",
	"date": "2018-12-28T08:57:28+0000",
	"uuid": "87e3c8fd-0a7e-11e9-af72-02420a0a0012"
	}
	}</pre>	

Response Example	**HTTP status:** 200 no body	

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

Table 8: The REST API consumed by Solver to Deployment

Operation Summary	End Point:	*/api/solverToDeployment/applySolutionNotification/{processId}/{subjectId}*
	HTTP Method:	POST
Description	This operation enables to inform the control process about the transformation result from the CP model solution to the application's deployment instance model in CAMEL. **Offered By:** *Control Plane*	
Parameters	*applicationId*: String [INPUT] – The identifier of the application *result:* Result [INPUT] – The result of creating the CP model. *watermark*: Watermark [INPUT] – Watermark of the message. Stores information about origin of the message.	

Request Example	<pre>{
	applicationId = "FCR",
	result = {
	status = "SUCCESS"
	},
	watermark = {
	user = "S2D",
	system = "S2D",
	date = "2018-12-28T08:57:21+0000",
	uuid = "87e3c8fd-0a7e-11e9-af72-02420a0a0012"
	},
	}</pre>	

Response Example	**HTTP status:** 200 no body	

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

3.8 DLMS

DLMS is a very critical component in the MELODIC platform as it manages the lifecycle of big data. This management is currently confined to the bookkeeping of metadata about all the big data manipulated by user applications, the migration of data as well as evaluating deployment solutions proposed by Solvers in terms of their data management cost. As such, the respective REST API offered by this platform component includes operations that maps to the realisation of the aforementioned three main functionalities. Table 9 details this REST API in terms of the operations that it features.

Table 9: The REST API offered by DLMS

Operation Summary	**End Point:**	*/ds*
	HTTP Method:	GET
Description	This operation enables to receive all the metadata about all the data sets of a user.	
Parameters	*id*: String [OUTPUT] – the ID of the DS *name*: String [OUTPUT] – the name of the DS *dataSourceType*: String [OUTPUT] – the type of the data source *ufsURI*: String [OUTPUT] – the URI of the under file system (UFS) *mountPoint*: String [OUTPUT] – the mount point for the data set	
Request Example	/ds	

Response Example	`[ {`
	`id: 1:`
	`name: "DS1",`
	`dataSourceType: "HDFS",`
	`ufsURI: "http://master:9000/",`
	`mountPoint: "/melodic/ds1"`
	`},`
	`…]`	

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

Operation Summary	End Point:	*/ds/id/{id}*
	HTTP Method:	GET, DELETE, PUT
Description	Depending on the HTTP method involved, this operation can enable to retrieve, delete or update the metadata about a particular data set based on its ID.	
Parameters	*id*: Long [INPUT/OUTPUT] – the ID of the data set to be obtained, deleted and stored. Output only for GET. *name*: String [INPUT/OUTPUT] – the name of the DS *ufsURI*: String [INPUT/OUTPUT] – the URI of the under file system (UFS) *mountPoint*: String [INPUT/OUTPUT] – the mount point for the data set Last 3 parameters are input for PUT and output for GET.	

Request Example	`/ds/id/2 → for GET/DELETE`
	
	`{`
	`"id": 2,`
	`"name": "S3BucketDatas",`
	`"ufsURI": "s3a://datasourcetest2/bat",`
	`"mountPoint": "/melodic/S3BucketDatas"`
	`} → for PUT`	

Response Example	`{`
	`id: 2:`
	`name: "S3BucketData",`
	`dataSourceType: "S3",`
	`ufsURI: "s3a://datasourcetest2/bat",`
	`mountPoint: "/melodic/S3BucketData"`
	`} → for GET`
	
	`Status: 200 OK → for DELETE`
	
	`Status: 204 No Content → for PUT`	

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

Operation Summary	End Point:	*/ds/name/{name}*
	HTTP Method:	GET, DELETE, PUT
Description	Depending on the HTTP method involved, this operation can enable to retrieve, delete or update the metadata about a particular data set based on its name.	
Parameters	*name*: String [INPUT/OUTPUT] – the name of the DS *id*: Long [INPUT/OUTPUT] – the ID of the data set to be obtained, deleted and stored. *ufsURI*: String [INPUT/OUTPUT] – the URI of the under file system (UFS) *mountPoint*: String [INPUT/OUTPUT] – the mount point for the data set Last 3 parameters are input for PUT and output for GET.	

Request Example	`/ds/name/S3BucketData → for GET/DELETE`
	
	`{`
	`"id": 2,`
	`"name": "S3BucketData2",`
	`"ufsURI": "s3a://datasourcetest2/bat",`
	`"mountPoint": "/melodic/S3BucketData2"`
	`} → for PUT`	

Response Example	`{`
	`id: 2:`
	`name: "S3BucketData",`
	`dataSourceType: "S3",`
	`ufsURI: "s3a://datasourcetest2/bat",`
	`mountPoint: "/melodic/S3BucketData"`
	`} → for GET`
	
	`Status: 200 OK → for DELETE`
	
	`Status: 204 No Content → for PUT`	

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

Operation Summary	End Point:	*/dataModel*
	HTTP Method:	POST
Description	This operation enables to submit a whole CAMEL data model to the DLMS. In result, the respective metadata (as encapsulated by this model) will be generated and stored in DLMS.	
Parameters	*applicationId*: String [INPUT] – the ID of the user application *notificationURI*: String [INPUT] – The URI of an endpoint to send the result of this operation *watermark*: Watermark (optional) [INPUT] – Watermark of the message. Stores information about origin of the message	

Request Example	<pre>{
	"applicationId": "PeopleFlow2",
	"notificationURI": "/notification/msg"
	}</pre>	

Response Example	<pre>Status: 201 Created</pre>	
Operation Summary	End Point:	*/migrate/file*
	HTTP Method:	POST
Description	This operation enables to migrate a file from an origin to a destination path.	
Parameters	*pathFrom*: String [INPUT] – the origin path of the file to migrate *pathTo*: String [INPUT] – the destination path for the file migration	

Request Example	<pre>{
	"pathFrom": "/melodic/S3BucketDatas/test.txt",
	"pathTo": "/melodic/S3Bucket2/"
	}</pre>	

Response Example	<pre>Status: 200 OK</pre>	

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

Operation Summary	End Point:	*/migrate/dir*
	HTTP Method:	POST
Description	This operation enables to migrate a directory from an origin to a destination path.	
Parameters	*pathFrom*: String [INPUT] – the origin path of the directory to migrate *pathTo*: String [INPUT] – the destination path of the directory to migrate	

Request Example	<pre>{
	"pathFrom": /melodic/S3BucketDatas/",
	"pathTo": "/melodic/S3Bucket2/"
	}</pre>	

Response Example	<pre>Status: 200 OK</pre>	
Operation Summary	End Point:	*/migrate/ds*
	HTTP Method:	POST
Description	This operation enables to migrate a data set to a destination path.	
Parameters	*id*: Long [INPUT] – the ID of the DS to migrate *pathTo*: String [INPUT] – the destination path for the DS to be migrated	

Request Example	<pre>{
	"id": 3,
	"pathTo": "/melodic2"
	}</pre>	

Response Example	<pre>Status: 200 OK</pre>	

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

Operation Summary	End Point:	*/cloudprovider*
	HTTP Method:	GET, POST
Description	Depending on the HTTP method involved, this operation can enable to retrieve metadata about all cloud providers currently handled as well as to store metadata about a new one.	
Parameters	*id*: Long (optional) [INPUT/OUTPUT] – the ID of the cloud provider *name*: String [INPUT/OUTPUT] – the name of the cloud provider *notes*: String (optional) [INPUT/OUTPUT] – some details about this cloud provider *public*: Boolean (optional) [INPUT/OUTPUT] – indication of whether this provider supplies a public cloud All parameters are given as input only for POST	

Request Example	`{`
	`"name": Google,`
	`"public": "True"`
	`} → for POST`	

Response Example	`[{`
	`id: 1,`
	`name: "AWS",`
	`notes: "",`
	`public: true`
	`},`
	`…] → for GET`
	
	`Status: 200 OK → for POST`	

Operation Summary	End Point:	*/cloudprovider/search?id={id}*
	HTTP Method:	GET
Description	This operation enables to obtain the metadata about a certain cloud provider based on its ID.	

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

Parameters	*id*: Long [INPUT/OUTPUT] – the ID of the cloud provider being searched *name*: String [OUTPUT] – the name of the cloud provider *notes*: String (optional) [OUTPUT] – some details about this cloud provider *public*: Boolean (optional) [OUTPUT] – indication of whether this provider supplies a public cloud	
Request Example	`/cloudprovider/search/?id=1`	
Response Example	`{ id: 1, name: "AWS", notes: "", public: true }`	
Operation Summary	End Point:	*/cloudprovider/search?name={name}*
	HTTP Method:	GET
Description	This operation enables to obtain the metadata about a certain cloud provider based on its name.	
Parameters	*name*: String [INPUT/OUTPUT] – the name of the cloud provider being searched *id*: Long [OUTPUT] – the provider's ID *notes*: String (optional) [OUTPUT] – some details about this cloud provider *public*: Boolean (optional) [OUTPUT] – indication of whether this provider supplies a public cloud	
Request Example	`/cloudprovider/search/?name=aws`	
Response Example	`{ id: 1, name: "AWS", notes: "",`	

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

	<pre>public: true
	}</pre>	

Operation Summary	End Point:	*/cloudprovider/update?id={id}*
	HTTP Method:	PUT
Description	This operation enables to update the metadata about a certain cloud provider based on its ID.	
Parameters	*id*: Long [INPUT] – the ID of the cloud provider being updated *name*: String [INPUT] – the (possibly new) name of the cloud provider *notes*: String (optional) [INPUT] – updated notes for the cloud provider *public*: Boolean (optional) [INPUT] – potential change or determination of whether this provider supplies a public cloud	

Request Example	<pre>{
	"name":"Google",
	"public": "True"
	}</pre>	

Response Example	<pre>Status: 204 No Content</pre>	
Operation Summary	End Point:	*/cloudprovider/update?name={name}*
	HTTP Method:	PUT
Description	This operation enables to update the metadata about a certain cloud provider based on its name.	
Parameters	*name*: String [INPUT] – the name of the cloud provider being updated *id*: Long [INPUT] – the (potentially modified) ID of the cloud provider *notes*: String (optional) [INPUT] – updated notes for the cloud provider *public*: Boolean (optional) [INPUT] – potential change or determination of whether this provider supplies a public cloud	

Request Example	<pre>{
	"name":"Google",
	"public": "True"</pre>	

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

	}	
Response Example	Status: 204 No Content	
Operation Summary	End Point:	*/datacenter*
	HTTP Method:	GET, POST
Description	Depending on the HTTP method involved, this operation enables to obtain the metadata about all data centres or to store the metadata about a certain data centre.	
Parameters	*id*: Long (optional) [INPUT/OUTPUT] – the ID of the datacenter (DC) *name*: String [INPUT/OUTPUT] – the DC name *regionId*: Long (optional) [INPUT/OUTPUT] – the ID of the region in which the DC is situated *cloudProviderId*: Long (optional) [INPUT/OUTPUT] – the ID of the DC's provider All parameters are input for POST and output for GET	
Request Example	{ "name": "UK-West", "regionId": 3, "cloudProviderId": 1, "public": "True" } → for POST	
Response Example	[{ id: 1, name: "ca-central-1", regionId: 1, cloudProviderId: 1 }, …] → for GET Status: 200 OK → for POST	

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

<table>
<tr><td rowspan="2">Operation Summary</td><td>End Point:</td><td>/datacenter /search?id={id}</td></tr>
<tr><td>HTTP Method:</td><td>GET</td></tr>
<tr><td>Description</td><td colspan="2">This operation enables to obtain the metadata about a certain data centre based on its ID.</td></tr>
<tr><td>Parameters</td><td colspan="2">id: Long (optional) [INPUT/OUTPUT] – the ID of the datacenter (DC) being searched
name: String [OUTPUT] – the DC name
regionId: Long (optional) [OUTPUT] – the ID of the region in which the DC is situated
cloudProviderId: Long (optional) [OUTPUT] – the ID of the DC's provider</td></tr>
<tr><td>Request Example</td><td colspan="2"><code>/datacenter/search/?id=1</code></td></tr>
<tr><td>Response Example</td><td colspan="2"><pre>[{
  id: 1,
  name: "ca-central-1",
  regionId: 1,
  cloudProviderId: 1
 },
…] → for GET

{
  id: 1,
  name: "ca-central-1",
  regionId: 1,
  cloudProviderId: 1
}</pre></td></tr>
</table>

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

<table>
<tr><td rowspan="2">Operation Summary</td><td>End Point:</td><td>/datacenter/search?name={name}</td></tr>
<tr><td>HTTP Method:</td><td>GET</td></tr>
<tr><td>Description</td><td colspan="2">This operation enables to obtain the metadata about a certain data centre based on its name.</td></tr>
<tr><td>Parameters</td><td colspan="2">id: Long (optional) [OUTPUT] – the ID of the datacenter (DC)
name: String [INPUT/OUTPUT] – the name of DC being searched
regionId: Long (optional) [OUTPUT] – the ID of the region in which the DC is situated
cloudProviderId: Long (optional) [OUTPUT] – the ID of the DC's provider</td></tr>
<tr><td>Request Example</td><td colspan="2"><code>/datacenter/search/?name=ca-central-1</code></td></tr>
<tr><td>Response Example</td><td colspan="2"><pre>[{
  id: 1,
  name: "ca-central-1",
  regionId: 1,
  cloudProviderId: 1
 },
…] → for GET

{
  id: 1,
  name: "ca-central-1",
  regionId: 1,
  cloudProviderId: 1
}</pre></td></tr>
</table>

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

Operation Summary	End Point:	*/datacenter/update?id={id}*
	HTTP Method:	PUT
Description	This operation enables to update the metadata about a certain data centre based on its ID.	
Parameters	*id*: Long (optional) [INPUT] – the ID of the DC being updated *name*: String [INPUT] – the (potentially updated) DC name *regionId*: Long (optional) [INPUT] – the (potentially updated) ID of the region in which the DC is situated *cloudProviderId*: Long (optional) [INPUT] – the (potentially updated) ID of the DC's provider	
Request Example	`{ "name":" UK-West", "regionId": "5" }`	
Response Example	`Status: 204 No Content`	
Operation Summary	End Point:	*/datacenter/update?name={name}*
	HTTP Method:	PUT
Description	This operation enables to obtain the metadata about a certain data centre based on its name.	
Parameters	*id*: Long (optional) [INPUT] – the (potentially updated) DC ID *name*: String [INPUT] – the name of the DC being updated *regionId*: Long (optional) [INPUT] – the (potentially updated) ID of the region in which the DC is situated *cloudProviderId*: Long (optional) [INPUT] – the (potentially updated) ID of the DC's provider	
Request Example	`{ "name":" UK-West", "regionId": "5" }`	

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

Response Example	`Status: 204 No Content`	
Operation Summary	End Point:	*/region*
	HTTP Method:	GET, POST
Description	Depending on the HTTP method involved, this operation enables to either obtain the metadata about all regions handled or to update the metadata about a specific region.	
Parameters	*id*: Long (optional) [INPUT/OUTPUT] – the ID of the region *name*: String [INPUT/OUTPUT] – the name of the region *cloudProviderId*: Long (optional) [INPUT/OUTPUT] – the id of the provider of this cloud-specific region All parameters are input for POST and output for GET	

Request Example	`{`
	`"name":" Washington"`
	`} → for POST`	

Response Example	`[{`
	`id: 1,`
	`name: "Central",`
	`cloudProviderId: 1`
	`},`
	`…] → for GET`
	
	`Status: 200 OK → for POST`	

Operation Summary	End Point:	*/region /search?id={id}*
	HTTP Method:	GET
Description	This operation enables to obtain the metadata about a certain region based on its ID.	

Parameters	*id*: Long [INPUT/OUTPUT] – the ID of the region being searched *name*: String [OUTPUT] – the name of the region *cloudProviderId*: Long (optional) [OUTPUT] – the id of the provider of this cloud-specific region	
Request Example	`/region/search/?id=1`	

Response Example	`{`
	`id: 1,`
	`name: "Central",`
	`cloudProviderId: 1`
	`}`	

Operation Summary	**End Point:**	*/datacenter/search?name={name}*
	HTTP Method:	GET
Description	This operation enables to update the metadata about a certain region based on its ID.	
Parameters	*id*: Long [OUTPUT] – the ID of the region *name*: String [INPUT/OUTPUT] – the name of the region being searched *cloudProviderId*: Long (optional) [OUTPUT] – the id of the provider of this cloud-specific region	
Request Example	`/region/search/?name=Central`	

Response Example	`{`
	`id: 1,`
	`name: "Central",`
	`cloudProviderId: 1`
	`}`	

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

Operation Summary	End Point:	*/region /update?id={id}*
	HTTP Method:	PUT
Description	This operation enables to obtain the metadata about a certain region based on its name.	
Parameters	*id*: Long (optional) [INPUT] – the ID of the region being updated *name*: String [INPUT] – the (potentially updated) region name *cloudProviderId*: Long (optional) [INPUT] – the (potentially updated) id of the provider of this cloud-specific region	

Request Example	`{`
	`"name":"Central",`
	`"cloudProviderId":"5"`
	`}`	

Response Example	`Status: 204 No Content`	
Operation Summary	End Point:	*/region /update?name={name}*
	HTTP Method:	PUT
Description	This operation enables to update the metadata about a certain region based on its name.	
Parameters	*id*: Long (optional) [INPUT] – the (potentially updated) region ID *name*: String [INPUT] – the name of the region being updated *cloudProviderId*: Long (optional) [INPUT] – the (potentially updated) id of the provider of this cloud-specific region	

Request Example	`{`
	`"name":"Central",`
	`"cloudProviderId":"5"`
	`}`	

Response Example	`Status: 204 No Content`	

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

Operation Summary	End Point:	*/applicationcomponent*
	HTTP Method:	GET, POST
Description	Depending on the HTTP method involved, this operation enables to either obtain the metadata about all application components or to store the metadata about a certain application component.	
Parameters	*id*: Long (optional) [INPUT/OUTPUT] – the ID of the application component *name*: String [INPUT/OUTPUT] – the application component's name All parameters are input for POST and output for GET	

Request Example	`{`
	`"name":"ac3"`
	`} → for POST`	

Response Example	`[`
	`{`
	`id: 1,`
	`name: "ac1"`
	`},`
	`…] → for GET`
	
	`Status: 200 OK → for POST`	

Operation Summary	End Point:	/applicationcomponent/search?id={id}
	HTTP Method:	GET
Description	This operation enables to obtain the metadata about a certain application component based on its ID.	
Parameters	*id*: Long (optional) [INPUT/OUTPUT] – the ID of the application component being searched *name*: String [OUTPUT] – the application component's name	
Request Example	`/applicationcomponent/search/?id=1`	

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

Response Example	<pre>{
	id: 1,
	name: "ac1",
	}</pre>	

Operation Summary	End Point:	*/applicationcomponent/search?name={name}*
	HTTP Method:	GET
Description	This operation enables to obtain the metadata about a certain application component based on its name.	
Parameters	*id: Long (optional) [OUTPUT] – the ID of the application component* *name: String [INPUT/OUTPUT] – the name of the application component being searched*	
Request Example	`/applicationcomponent/search/?name=ac1`	

Response Example	<pre>{
	id: 1,
	name: "ac1",
	}</pre>	

Operation Summary	End Point:	*/applicationcomponent/update?id={id}*
	HTTP Method:	PUT
Description	This operation enables to update the metadata about a certain application component based on its ID.	
Parameters	*id*: Long (optional) [INPUT] – the ID of the application component being updated *name*: String [INPUT] – the updated name of the application component	

Request Example	<pre>{
	"name":"ac4"
	}</pre>	

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

Response Example	`Status: 204 No Content`	
Operation Summary	End Point:	*/applicationcomponent/update?name={name}*
	HTTP Method:	PUT
Description	This operation enables to update the metadata about a certain application component based on its ID.	
Parameters	*id*: Long (optional) [INPUT] – the (potentially updated) ID of the application component *name*: String [INPUT] – the name of the application component being updated	

Request Example	`{`
	`"name":"ac4"`
	`}`	

Response Example	`Status: 204 No Content`	
Operation Summary	End Point:	*/datasource*
	HTTP Method:	GET, PUT
Description	Depending on the HTTP method involved, this operation can enable to either obtain the metadata about all data sources or to store the metadata about a certain data source.	
Parameters	*id*: Long (optional) [INPUT/OUTPUT] – the ID of the data source *name*: String [INPUT/OUTPUT] – the name of the data source All parameters are input for POST and output for GET	

Request Example	`{`
	`"name":"ds3"`
	`} → for POST`	

Response Example	`[{`
	`id: 1,`
	`name: "ds1"`
	`},`	

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

	…] → for GET Status: 200 OK → for POST	
Operation Summary	End Point:	*/datasource /search?id={id}*
	HTTP Method:	GET
Description	This operation enables to obtain the metadata of a certain resource based on its ID.	
Parameters	*id*: Long (optional) [INPUT/OUTPUT] – the ID of the data source being searched *name*: String [OUTPUT] – the name of the data source	
Request Example	`/datasource/search/?id=1`	

Response Example	{
	id: 1,
	name: "ds1"
	}	

Operation Summary	End Point:	*/datasource /search?name={name}*
	HTTP Method:	GET
Description	This operation enables to obtain the metadata of a certain resource based on its ID.	
Parameters	*id*: Long (optional) [OUTPUT] – the ID of the data source *name*: String [INPUT/OUTPUT] – the name of the data source being searched	
Request Example	*/datasource/search/?name=Central*	

Response Example	{
	id: 1,
	name: "ds1"
	}	

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

Operation Summary	End Point:	*/datasource /update?id={id}*
	HTTP Method:	PUT
Description	This operation enables to update the metadata of a certain resource based on its ID.	
Parameters	*id*: Long (optional) [INPUT] – the ID of the data source being updated *name*: String [INPUT] – the updated name of the data source	

Request Example	`{`
	`"id":"3"`
	`"name":"ds3"`
	`}`	

Response Example	`Status: 204 No Content`	
Operation Summary	End Point:	*/datasource /update?name={name}*
	HTTP Method:	PUT
Description	This operation enables to update the metadata of a certain resource based on its name.	
Parameters	*id*: Long (optional) [INPUT] – the (potentially updated) ID of the data source being updated *name*: String [INPUT] – the name of the data source being updated	

Request Example	`{`
	`"id":"3"`
	`"name":"ds3"`
	`}`	

Response Example	`Status: 204 No Content`	

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

Operation Summary	End Point:	*/applicationcomponent_datasource_affinity*
	HTTP Method:	GET
Description	This operation enables to obtain the affinity between an application component and a data set.	
Parameters	*appCompId*: Long [OUTPUT] – the application component's ID *dsID*: Long [OUTPUT] – the data set's ID *affinity*: Double [OUTPUT] – the affinity between the application component and data set	
Request Example	N/A	

Response Example	<pre>[
	{
	acDsKey: {
	appCompId: 1,
	dsID: 1
	},
	affinity: 0.076
	},
	…]</pre>	

Operation Summary	End Point:	*/datacenterzone*
	HTTP Method:	GET
Description	This operation enables to obtain the metadata for all data centre zones.	
Parameters	*dataCenterId*: Long [OUTPUT] – the ID of the data centre *zone*: Long [OUTPUT] – the ID of the DC's zone	
Request Example	N/A	

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

Response Example	<pre>[
	{
	dataCenterId: 1,
	zone: 16
	},
	…]</pre>	

Operation Summary	**End Point:**	*/datacenterzone/search?datacenter_id={id}*
	HTTP Method:	GET
Description	This operation enables to obtain the metadata for the zone involved in the data centre whose ID is given as input.	
Parameters	*id*: Long [INPUT/OUTPUT] – the ID of the data centre *zone*: Long [OUTPUT] – the ID of the DC's zone	
Request Example	`/datacenterzone/search?datacenter_id=1`	

Response Example	<pre>{
	dataCenterId: 1,
	zone: 16
	}</pre>	

Operation Summary	**End Point:**	*/datacenterzone/search?zone_id=={zone}*
	HTTP Method:	GET
Description	This operation enables to obtain the metadata for a specific zone based on its ID.	
Parameters	*id*: Long [INPUT/OUTPUT] – the ID of the data centre *zone*: Long [OUTPUT] – the ID of the DC's zone	

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

Request Example	`/datacenterzone/search?zone_id=16`	
Response Example	`{ dataCenterId: 1, zone: 16 }`	
Operation Summary	End Point:	*/datacenterzone/update?datacenter_id={id}*
	HTTP Method:	PUT
Description	This operation enables to update the metadata for a specific zone based on the ID of its mapped data centre.	
Parameters	*id*: Long [INPUT/OUTPUT] – the ID of the data centre being updated *zone*: Long [OUTPUT] – the updated ID of the DC's zone	
Request Example	`/datacenterzone/update?datacenter_id=1` `{ "dataCenterId":"1" "zone":"15" }`	
Response Example	`Status: 204 No Content`	
Operation Summary	End Point:	*/ dlmsController/utilityValue*
	HTTP Method:	PUT
Description	*DlmsDiffBundle*: List<DlmsConfigurationDiff> [INPUT] – this is a list of configuration differences between the current and proposed solution. It includes the following internal information per item: *id*: Long [INPUT] – unique identifier for the node candidate *NodeCandidate*: String [INPUT] – the name of the node candidate	

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

	cardinality: Integer [INPUT] – the difference in the cardinality of the identified component with respect to the current and proposed solution *results*: Double [OUTPUT] – the actual utility of the deployment solution examined from the point of view of the DLMS
Parameters	*id*: Long [INPUT/OUTPUT] – the ID of the data centre being updated *zone*: Long [OUTPUT] – the updated ID of the DC's zone
Request Example	`[{ "id":"1", "NodeCandidate":"cand1", "cardinality":"1" }, { "id":"2", "NodeCandidate":"cand2", "cardinality":"2" }]`
Response Example	`{ "results": "0.5" }`

3.9 Adapter

The Adapter is the component responsible for orchestrating the deployment of a multi-cloud data-intensive application. Initially, it takes as input a deployment instance model in CAMEL and transforms it into a detailed deployment plan. After being verified, this plan is then orchestrated by executing respective deployment commands via using the REST API of the Executionware. Further, the Adapter is responsible for communicating with the EMS in order to construct and update the application's low-level monitoring infrastructure in terms of particular sensors.

Based on the above analysis, from an external point of view, the Adapter offers a single API operation, dedicated to initiating the application deployment. In addition, it consumes three main interfaces: (a) one from the Notification Service to inform the control process about the outcome

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

of the application deployment attempt; (b) one from the Executionware for issuing the deployment commands; (c) another from the EMS to retrieve the information of the sensors to deploy in the application component nodes so as to establish the application's low-level monitoring infrastructure. As such, the following two tables, Table 10 and Table 11 detail the REST APIs offered and consumed by this MELODIC platform component.

Table 10: The REST API offered by the Adapter

Operation Summary	End Point:	*/applicationDeployment*
	HTTP Method:	POST
Description	This operation is invoked in order to initiate the deployment of an application based on its deployment model instance in CAMEL.	
Parameters	*applicationId*: String [INPUT] – The identifier of the application *cdoModelsPath*: String [INPUT] – The path in the CDO server where application model in CAMEL can be found *notificationURI*: String [INPUT] – The URI of an endpoint to send the result of Constraint Problem creation . *watermark*: Watermark [INPUT] – Watermark of the message. Stores information about origin of the message.	

Request Example	<pre>{
	"applicationId": "FCR",
	"cdoModelsPath": "upperware-models/FCRApp1545987428463",
	"notificationURI": "/api/adapter/deploymentNotification/87e3c8fd-0a7e-11e9-af72-02420a0a0012/ApplicationDeploymentNotification",
	"watermark": {
	"user": "Camunda",
	"system": "Camunda",
	"date": "2018-12-28T08:57:41+0000",
	"uuid": "87e3c8fd-0a7e-11e9-af72-02420a0a0012"
	}
	}</pre>	

Response Example	**HTTP status:** 200 no body	

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

Table 11: The REST APIs consumed by the Adapter

Operation Summary	End Point:	*/api/adapter/deploymentNotification/{processId}/{subjectId}*
	HTTP Method:	POST
Description	This operation enables to inform the control process about the status of the application deployment (i.e., if it was successful or not). **Offered By:** *Control Plane*	
Parameters	*applicationId*: String [INPUT] – The identifier of the application *deployedSolutionId*: String [INPUT] – The identifier of the solution from the CP model that was successfully deployed *result:* Result [INPUT] – The result of deploying the application *watermark*: Watermark [INPUT] – Watermark of the message. Stores information about origin of the message.	

Request Example	<pre>{
	"result" = {
	"status" = "ERROR",
	"errorDescription" = "Built plan was rejected by Plan Validator"
	},
	"watermark" = {
	"user" = "adapter",
	"system" = "adapter",
	"date" = "2018-12-28T09:01:51+0000",
	"uuid" = "87e3c8fd-0a7e-11e9-af72-02420a0a0012"
	},
	"applicationId" = "FCR"
	}</pre>	

Response Example	**HTTP status:** 200 no body	

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

Operation Summary	**End Point:**	*/findJobs*
	HTTP Method:	GET
Description	This operation enables to obtain all the deployment jobs for a particular user **Offered By:** *Executionware*	
Parameters	*As described in Executionware API description* http://cloudiator.org/rest-swagger/#operation/findJobs	
Request Example	N/A	
Response Example	*As described in Executionware API description*	
Operation Summary	**End Point:**	*/nodeGroup*
	HTTP Method:	GET
Description	This operation returns all groups of nodes. **Offered By:** *Executionware*	
Parameters	*As described in Executionware API description*	
Request Example	N/A	
Response Example	*As described in Executionware API description*	
Operation Summary	**End Point:**	*/nodeGroup/{node-group-id}*
	HTTP Method:	GET
Description	This operation enables to obtain metadata about a certain node group. **Offered By:** *Executionware*	
Parameters	*As described in Executionware API description*	
Request Example	N/A	
Response Example	*As described in Executionware API description*	

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

Operation Summary	**End Point:**	*/getProcesses*
	HTTP Method:	GET
Description	This operation enables to retrieve all (deployment) processes for a particular user. **Offered By:** *Executionware*	
Parameters	*As described in Executionware API description* http://cloudiator.org/rest-swagger/#operation/getProcesses	
Request Example	N/A	
Response Example	*As described in Executionware API description*	
Operation Summary	**End Point:**	*/processGroup*
	HTTP Method:	GET
Description	This operation enables to retrieve all groups of (deployment) processes for a particular user. **Offered By:** *Executionware*	
Parameters	*As described in Executionware API description*	
Request Example	N/A	
Response Example	*As described in Executionware API description*	
Operation Summary	**End Point:**	*/getQueuedTasks*
	HTTP Method:	GET
Description	This operation enables to retrieve all queued (deployment) tasks for a particular user. **Offered By:** *Executionware*	
Parameters	*As described in Executionware API description* http://cloudiator.org/rest-swagger/#operation/getQueuedTasks	

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

Request Example	N/A	
Response Example	*As described in Executionware API description*	
Operation Summary	**End Point:**	*/getSchedules*
	HTTP Method:	GET
Description	This operation enables to retrieve all schedules for a certain user. **Offered By:** *Executionware*	
Parameters	*As described in Executionware API description* http://cloudiator.org/rest-swagger/#operation/getSchedules	
Request Example	N/A	
Response Example	*As described in Executionware API description*	
Operation Summary	**End Point:**	*/addJob*
	HTTP Method:	POST
Description	This operation enables to post a new deployment job. **Offered By:** *Executionware*	
Parameters	*As described in Executionware API description* http://cloudiator.org/rest-swagger/#operation/addJob	
Request Example	*As described in Executionware API description*	
Response Example	*As described in Executionware API description*	
Operation Summary	**End Point:**	*/addNode*
	HTTP Method:	POST
Description	This operation enables to post the generation of a new node. **Offered By:** *Executionware*	

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

Parameters	*As described in Executionware API description* http://cloudiator.org/rest-swagger/#operation/addNode	
Request Example	*As described in Executionware API description*	
Response Example	*As described in Executionware API description*	
Operation Summary	End Point:	*/addSchedule*
	HTTP Method:	POST
Description	This operation enables to post the generation of a new schedule. **Offered By:** *Executionware*	
Parameters	*As described in Executionware API description* http://cloudiator.org/rest-swagger/#operation/addSchedule	
Request Example	*As described in Executionware API description*	
Response Example	*As described in Executionware API description*	
Operation Summary	End Point:	*/createProcess*
	HTTP Method:	POST
Description	This operation enables to post the generation of a new (deployment) process. **Offered By:** *Executionware*	
Parameters	*As described in Executionware API description* http://cloudiator.org/rest-swagger/#operation/createProcess	
Request Example	*As described in Executionware API description*	
Response Example	*As described in Executionware API description*	

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

Operation Summary	End Point:	*/monitors*
	HTTP Method:	POST
Description	*As described in EMS API description (See Section 3.10)* Offered By: *EMS*	
Parameters	*As described in EMS API description*	
Request Example	*As described in EMS API description*	
Response Example	*As described in EMS API description*	

3.10 Event Processing Management (EMS)

The EMS component of the MELODIC platform is responsible for establishing and maintaining a layered monitoring architecture for the user application managed by this platform. To this end, it includes API operations which are responsible for: constructing, updating and retrieving the status of this monitoring architecture. Concerning its interfacing requirements, it needs to communicate with the control process in order to inform it about the success in the deployment/updating of the monitoring infrastructure as well as with the Metasolver in order to indicate the event subscriptions that the latter component needs to listen to. Both the offered and consumed REST APIs by this component are detailed in the next two tables, Table 12 and Table 13, respectively.

Table 12: The REST APIs offered by EMS

Operation Summary	End Point:	*/camelModel*
	HTTP Method:	POST
Description	This operation is invoked by the Upperware control process to notify EMS Server that a new application CAMEL model is available.	
Parameters	*application-id:* String [INPUT] – The identifier of the application (also denoting the related CAMEL model)	

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

<table>
<tr><td></td><td colspan="2">notification-uri: String [INPUT] – The URL to call in order to notify ESB process of the invocation outcome. This invocation will occur asynchronously

watermark: WaterMark [INPUT] – Watermark of the message. Stores information about origin of the message</td></tr>
<tr><td>Request Example</td><td colspan="2"><pre>{
    "applicationId": "FCR",
    "notificationURI":
"/api/ems/camelModelNotification/87e3c8fd-0a7e-11e9-af72-02420a0a0012/CamelModelNotification",
    "watermark": {
        "user": "Camunda",
        "system": "Camunda",
        "date": "2018-12-28T08:57:41+0000",
        "uuid": "87e3c8fd-0a7e-11e9-af72-02420a0a0012"
    }
}</pre></td></tr>
<tr><td>Response Example</td><td colspan="2">Text/Plain string “OK”</td></tr>
<tr><td rowspan="2">Operation Summary</td><td>End Point:</td><td>/cpModelJson</td></tr>
<tr><td>HTTP Method:</td><td>POST</td></tr>
<tr><td>Description</td><td colspan="2">This operation is invoked by Metasolver in order to notify EMS Server about a CP model update. The input JSON message is used to extract the path to the application’s updated CP model.</td></tr>
<tr><td>Parameters</td><td colspan="2">cp-model-id: String [INPUT] – The path to application’s CP model</td></tr>
<tr><td>Request Example</td><td colspan="2"><pre>{
  "cp-model-id" : <string>
}</pre></td></tr>
<tr><td>Response Example</td><td colspan="2">Text/Plain string “OK”</td></tr>
</table>

Operation Summary	**End Point:**	*/monitors*
	HTTP Method:	POST
Description	This operation is invoked by the Upperware control process in order to retrieve the Monitors used for deploying sensors in application nodes. Monitors are an abstraction of sensors deployed in application nodes (VM, containers etc) and include information like metrics measured, components, sensors used, sinks and tags.	
Parameters	*application-id:* String [INPUT] – The identifier of the application (also denoting the related CAMEL model) *watermark:* WaterMark [INPUT] – Watermark of the message. Stores information about origin of the message *monitors*: array [OUTPUT] – A list of the Monitors extracted from CAMEL model. Each Monitor item contains the following information: *metric:* String [OUTPUT] – the measured attribute • *component:* String [OUTPUT] – the component the measurement refers to • *sensor:* map [OUTPUT] – map containing information about the sensor that takes the measurements. There are two sensor types; Pull Sensors and Push Sensors. Pull Sensor information: *className*: String [OUTPUT] – the class name of the sensor *interval*: map [OUTPUT] – the interval between two successive measurement reads. This value has the form: { "period": <integer>, "unit": <string> } *configuration*: array [OUTPUT] – array of sensor-specific configuration name-value pairs. Each pair has the form: { "key": <string>, "value": <string> } Push Sensor information: *port*: Integer [OUTPUT] – the port where push sensor sends its measurements	

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

	sinks: array [OUTPUT] – an array with the measurement sinks (used by ExecutionWare to configure sensors). Each sink item contains: • *type*: String [OUTPUT] – the sink type • *configuration*: array [OUTPUT] – an array of key-value pairs with the sink configuration. Each pair has the form: { "key": <string>, "value": <string> } *tags:* array [OUTPUT] – an optional array of monitor-specific configuration name-value pairs. Each pair has the form: { "key": <string>, "value": <string> }

Request Example	{
	"applicationId": "FCR",
	"watermark": {
	"user": "Camunda",
	"system": "Camunda",
	"date": "2018-12-28T08:57:41+0000",
	"uuid": "87e3c8fd-0a7e-11e9-af72-02420a0a0012"
	}
	}

Response Example	{
	"monitors": [{
	"metric": "ResponseTime_Sensor",
	"component": "Component_App",
	"sensor": {
	"className": "",
	"configuration": [],
	"interval": {
	"unit": "SECONDS",
	"period": 5
	}
	},
	"sinks": [{
	"type": "JMS",
	"configuration": [{
	"key": "jms.broker",
	"value":
	"failover:(tcp://localhost:61616)?initialReconnectDelay=1000
	&warnAfterReconnectAttempts=10"

<table>
<tr><td></td><td colspan="2"><pre>
      }, {
        "key": "jms.message.format",
        "value":
"de.uniulm.omi.cloudiator.visor.reporting.jms.MelodicJsonEnc
oding"
      }, {
        "key": "jms.topic.selector",
        "value":
"de.uniulm.omi.cloudiator.visor.reporting.jms.MetricNameTopi
cSelector"
      }]
    }]
  }],
  "watermark": {
    "user": "EMS",
    "system": "EMS",
    "date": "2019-09-11T12:19:51+0000",
    "uuid": "9d65b0b6-40a8-11e7-a919-92ebcb67fe33"
  }
}
</pre></td></tr>
<tr><td rowspan="2">Operation Summary</td><td>End Point:</td><td>/baguette/registerNode</td></tr>
<tr><td>HTTP Method:</td><td>POST</td></tr>
<tr><td>Description</td><td colspan="2">This operation is called to inform EMS Server that a new application node has been deployed. In result, the EMS Server needs to return installation instructions for the EMS client to be installed in the same node.</td></tr>
<tr><td>Parameters</td><td colspan="2">id: String [INPUT] – The new node id as provided by Executionware
name: String [INPUT] – The new node id as provided by Executionware
type: String [INPUT] – The node type as provided by Executionware
providerId: String [INPUT] – The node cloud service provider id as provided by Executionware
operatingSystem: String [INPUT] – The node operating system family as provided by Executionware</td></tr>
</table>

	ip: String [INPUT] – The node public IP address as provided by Executionware
	
	os: String [OUTPUT] – the OS for which the following instructions refer to
	
	instructions: array [OUTPUT] – array of installation instructions. Each instruction object contains the following information:
	
	• *taskType*: String [OUTPUT] – can be:
	  ○ LOG: prints a message in the log,
	  ○ CMD: a shell command to execute. If command exit code is not zero (0) then an error occurs
	  ○ CHECK: like CMD but acceptable exit code can be configured.
	  ○ FILE: create a new file, write its contents and set executable flag
	• *command*: String [OUTPUT] – if taskType=LOG, it is the message to print in logs. If taskType=CMD, it is the command to execute in shell
	• *fileName*: String [OUTPUT] – Applicable only when taskType is FILE. The full path and name of a new file that will be written
	• *contents*: String [OUTPUT] – When taskType is FILE, it provides the contents of the new file. When taskType is CHECK, it provides the error message when command's exit code is not the right one (see next).
	• *executable*: boolean [OUTPUT] – Applicable only when taskType is FILE. Flag indicating if the new file must be executable or not
	• *exitCode*: integer [OUTPUT] – Applicable only when taskType is CHECK. Used in conjunction with 'match'. If match is:
	  ○ *true*: it gives the expected exit code of the command (i.e. success)
	  ○ *false*: it gives a forbidden exit code (i.e. failure if command returns this exit code)
	
	match: boolean [OUTPUT] – Applicable only when taskType is CHECK. Used in conjunction with 'exitCode'. See above

Request Example	{
	"id": "DbComp#1",
	"name": "DbComp1_1",
	"type": "VM",
	"providerId": "1",
	"operatingSystem": "ubuntu",
	"ip": "147.102.1.1",
	}

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

<table>
<tr><td>Response Example</td><td colspan="2"><pre>
{
  "os": "LINUX",
  "instructions": [{
    "taskType": "LOG",
    "command": "Create Baguette Client installation directories",
    "executable": false,
    "exitCode": 0,
    "match": false
  }, {
    "taskType": "CMD",
    "command": "sudo mkdir -p /opt/baguette-client/bin",
    "executable": false,
    "exitCode": 0,
    "match": false
  }, {
    "taskType": "CMD",
    "command": "sudo mkdir -p /opt/baguette-client/conf",
    "executable": false,
    "exitCode": 0,
    "match": false
  }]
}
</pre></td></tr>
<tr><td rowspan="2">Operation Summary</td><td>End Point:</td><td>/baguette/stopServer</td></tr>
<tr><td>HTTP Method:</td><td>GET, POST</td></tr>
<tr><td>Description</td><td colspan="2">This operation shuts down the Baguette Server, closing all connections to EPAs / EMS clients.</td></tr>
<tr><td>Parameters</td><td colspan="2">N/A</td></tr>
</table>

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

Request Example	N/A	
Response Example	Text/Plain string "OK"	
Operation Summary	End Point:	*/ems/shutdown[/{exitApp}]*
	HTTP Method:	GET
Description	This operation allows to shut down the EPM. However, it must be enabled first in the EMS control service configuration.	
Parameters	*exitApp:* boolean [INPUT] – It is an optional URL argument. Legal values are 'true' or 'false'. True forces JVM to also exit (Docker container will terminate too).	
Request Example	/ems/shutdown	
Response Example	Text/Plain string "OK"	
Operation Summary	End Point:	*/ems/topology*
	HTTP Method:	GET, POST
Description	This operation returns information about the currently deployed event processing network (i.e., the connected EPAs and their roles)	
Parameters	N/A	
Request Example	N/A	
Response Example		
Operation Summary	End Point:	*/event/send/{clientId}/{topicName}/ {value}*
	HTTP Method:	GET, POST
Description	This operation commands the specified client(s) to create and publish an event at the specified topic containing the supplied value. This endpoint is used for event debugging purposes.	

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

Parameters	*clientId:* String [INPUT] – the client id (as given by Baguette Server) or '*' denoting all connected EPA *topicName:* String [INPUT] – the topic name in event broker *value:* Double [INPUT] – a double precision number representing a measurement	
Request Example	`/event/send/*/AvgRamMetricContext/45`	
Response Example	`Text/Plain string:` *OK* *ERROR* *EVENT DEBUGGING IS DISABLED* *BAGUETTE SERVER IS DISABLED* *BAGUETTE SERVER IS NOT RUNNING*	
Operation Summary	End Point:	*/event/generate-start/{clientId}/{topicName}/{interval}/{lowerValue}-{upperValue}*
	HTTP Method:	GET
Description	This operation commands the specified client(s) to start generating and publishing events at the specified interval, for the identified topic, containing a random value between *lower value* and *upper value* range. This endpoint is used for event debugging purposes.	
Parameters	*clientId:* String [INPUT] – the client id (as given by Baguette Server) or '*' denoting all connected EPA *topicName:* String [INPUT] – the topic name in event broker *interval:* Long [INPUT] – the interval between event generations in milliseconds *lowerValue & upperValue :* Double [INPUT] – two double precision numbers representing the value range for the topic	
Request Example	`/event/generate-start/*/AvgRamMetricContext/5000/0-150`	

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

Response Example	`Text/Plain string:` *`OK`* *`ERROR`* *`EVENT DEBUGGING IS DISABLED`* *`BAGUETTE SERVER IS DISABLED`* *`BAGUETTE SERVER IS NOT RUNNING`*	
Operation Summary	**End Point:**	*/event/generate-stop/{clientId}/{topicName}*
	HTTP Method:	GET
Description	This operation commands the specified client(s) to stop generating and publishing events. This command cancels out a previous *generate-start* command (or is ignored if there is no such previous command). This endpoint is used for event debugging purposes.	
Parameters	*clientId:* String [INPUT] – the client id (as given by Baguette Server) or '*' denoting all connected EPA *topicName:* String [INPUT] – the topic name in event broker	
Request Example	`/event/generate-stop/*/AvgRamMetricContext`	
Response Example	`Text/Plain string:` *`OK`* *`ERROR`* *`EVENT DEBUGGING IS DISABLED`* *`BAGUETTE SERVER IS DISABLED`* *`BAGUETTE SERVER IS NOT RUNNING`*	
Operation Summary	**End Point:**	*/health*
	HTTP Method:	GET
Description	This operation enables to get the health status of the EMS, i.e., whether EMS server is up and running as well as whether connections to this server can be established.	

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

Parameters	N/A
Request Example	N/A
Response Example	HTTP response code: 200 OK

In the following table, we explain the REST APIs consumed by EMS.

Table 13: The REST APIs consumed by EMS

Operation Summary	**End Point:**	*/notification-uri*
	HTTP Method:	POST
Description	After the Camel model operation is executed, the specified notification URL is called to signal the Upperware control process that EMS finished processing the respective CAMEL model and initialised itself. **Offered By:** *Control Plane*	
Parameters	*application-id*: String [INPUT] – The identifier of the application (also denoting the related CAMEL model) *result*: Notification result [INPUT] • *status*: "SUCCESS", "ERROR" • *errorCode*: Specific error code • *errorDescription*: Description of an error *watermark*: WaterMark [INPUT] – Watermark of the message. Stores information about origin of the message	

Request Example	<pre>{
	"applicationId": "/FCRWithDlms",
	"result": {
	"status": "SUCCESS",
	},
	"watermark": {
	"user": "EMS",
	"system": "EMS",
	"date": "2019-09-11T12:19:51+0000",</pre>	

<table>
<tr><td></td><td colspan="2"><pre>
    "uuid": "9d65b0b6-40a8-11e7-a919-92ebcb67fe33"
  }
}
</pre></td></tr>
<tr><td>Response Example</td><td colspan="2">N/A</td></tr>
<tr><td rowspan="2">Operation Summary</td><td>End Point:</td><td>/updateConfiguration</td></tr>
<tr><td>HTTP Method:</td><td>POST</td></tr>
<tr><td>Description</td><td colspan="2">This operation is used to send new Metasolver configurations, regarding the topics it must subscribe to, in order to receive new metric values (needed to update CP model), or receive events signaling that reconfiguration of the application must occur. Note that this operation's URL/endpoint is specified in the EMS control service configuration file.

Offered By: Metasolver</td></tr>
<tr><td>Parameters</td><td colspan="2">mvv: map [INPUT] – an associative array (i.e. map) where a current-config variable (contained in the deployed solution) is mapped to a CP metric in CP model. This mapping is used for copying the (actual) values of current-config. variables onto the corresponding CP metrics of CP model, in order to make them available to solvers for the next solving iteration.

subscriptions: array [INPUT] – an array of subscription objects. Each subscription object is used to instruct Metasolver, to connect to a specific event broker and subscribe to a certain event topic. Each subscription object contains the following information:
• topic: String [INPUT] – The name of event topic where Metasolver will subscribe
• url: String [INPUT] – the connection string of the event broker (including protocol, address, port and other connection parameters)
• client-id: String [INPUT] – An optional client name used to identify the Metasolver connection when listing the active connections of an event broker
• type: String [INPUT] – The type of events sent in this topic. Two values are expected:</td></tr>
</table>

<table>
<tr><td></td><td>
'MVV': meaning that values of the received events must be extracted and used to update CP model
'SCALE': meaning that when an event is received, Metasolver must request a new reconfiguration iteration to start.
</td></tr>
<tr><td>Request Example</td><td>
<pre>
{
  "mvv": {
    "AppCardinality":   "AppActCardinality"
  },
  "subscriptions": [{
    "topic": "GlobalReconfigurationRule",
    "client-id": "",
    "type": "SCALE",
    "url": "tcp://ems:61616"
  },
  {
    "topic": "AvgResponseTime",
    "client-id": "",
    "type": "MVV",
    "url": "tcp://ems:61616"
  }]
}
</pre>
</td></tr>
<tr><td>Response Example</td><td>Text/Plain string "OK"</td></tr>
</table>

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

4 Executionware Interfaces

The Executionware and its implementation uses the OpenApi[3] specification and the Swagger[4] toolset for its API documentation. A detailed overview of the REST API is available under http://cloudiator.org/rest-swagger/ and additional information about the concepts and functionalities can be found in D4.1 [2] for the cloud provider mapping, in D4.3 [9] for the resource management and in D4.5 [10] for the data processing. Please note that in order to keep this deliverable length short, we have not adopted the table-based structure to present the Executionware's API in a more compact way than its Swagger documentation. Nevertheless, the portion of this API consumed by the MELODIC platform has been already presented in the respective API consumption tables of the CP Generator and the Adapter.

5 CAMEL 2.0

In the context of the Melodic project, CAMEL [4] has evolved to a new version by considering feedback obtained from user studies in the PaaSage project but also aligning with the goals of Melodic with respect to big data management. Further, such an evolution was supported through the experience of the CAMEL developers who were also aware of some initial weaknesses of CAMEL 1.0.

In its new version, CAMEL is able to cover multiple aspects which are related to the capturing of all appropriate information for the management of big data and multi-cloud applications. A summary of all these aspects, which take the form of particular meta-models/sub-DSLs is given by the following table, which also showcases the respective entities responsible for producing the models conforming to such meta-models. As it can be seen, the devops is responsible for modelling the application, its topology and requirements as well as the data that it manipulates. Further, the devops can also specify a metric and scalability model in order to cover the monitoring and reconfiguration of his/her multi-cloud application. On the other hand, the system is responsible for producing instance models for the deployment and monitoring aspects, which conform to the respective (type) models that have been specified by the devops. It is also responsible for producing and maintaining the application's execution model which covers the application's history over time. Finally, the admin is another kind of user/entity, who has the responsibility to maintain the organisation model of the respective user organisation (operating an instance of the Melodic platform) as well as the metadata model (with the rationale that the

[3] https://www.openapis.org/
[4] https://swagger.io/

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

respective features that can be selected by the devops in formulating platform/resource requirements or data properties should have been realised in the platform).

Table 14: Overview of the coverage of CAMEL 2.0

Name	Coverage	Editor / Modeller	Design / Runtime
Core	Top model, container of other models, application, attributes	devops / system	Both
Deployment	Application topology (components with their configuration & communication / placement dependencies)	devops / system	Both
Requirement	resource, platform, security, location, OS, provider, scaling, QoS (SLOs & optimisation) requirements	devops / system	Design
Metric	Metric, Sensors, Variables, Metric Templates + Mathematical metric/variable formulas	devops / system	Both
Constraint	Metric & variable (single) constraints, logical and if-then composite constraints	devops	Design
Scalability	Scalability rules, Scaling actions (horizontal), Events (single & logical/temporal event patterns)	devops	Design
Data	Data & Data sources	devops / system	Both
Location	Physical and cloud-based locations	devops / admin	Design
Unit	Units of measurements (single, composite, dimensionless), dimensions	devops	Design
Type	Types (numerical ranges, lists, range unions) & values (int, double, String, boolean)	devops	Design
Organisation	Organisations, users, roles, policies, user groups, role assignments	admin	Design / Runtime
Execution	state transitions and related actions along with their causes, measurements, SLO assessments	admin	Runtime
Security	Security domains, controls, metrics, metric templates, attributes	devops	Design

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

Metadata	Ontology-like structure with concepts as well as data & object properties at type & instance level	admin	Design

More details about the different meta-models in CAMEL 2.0 and how they can be used for the modelling of multi-cloud applications can be found in D2.2 and the project's confluence site[5]. In the latter case, a rather complete documentation is supplied for almost all CAMEL meta-models, including nice examples with CAMEL's textual syntax in order to raise the understandability of the user as well as assist him/her in his/her application modelling task.

To cover all the above types of entities involved in the manipulation of CAMEL models, CAMEL offers different kinds of interfaces. For the devops and admin users, CAMEL offers two editors to support them in the editing of CAMEL models. For the system (developers), CAMEL relies on Eclipse code generation facilities (from ecore models) in order to produce its domain code which coupled with the CDO[6]-based model management facilities from the PaaSage project that have been evolved in Melodic, enables both the generation, updating and storage of CAMEL models in the CDO model repository.

Focusing on CAMEL editing for devops and admins, the two editors offered by CAMEL include:

- *Textual editor*. This is an offline editor which enables users to specify CAMEL models that conform to the textual syntax of CAMEL. The editor offers some nice facilities like syntax and error highlighting, autocompletion and XMI model generation. The edited models can then be uploaded in an Melodic platform instance to support the respective application deployment. To reduce the modelling effort, certain template models have been also produced, taking the form of metric, unit, type, location and metadata models, which can be re-used by the users in the modelling of their applications in CAMEL. Such models are available from CAMEL's code repository in bitbucket[7] and can be imported into the textual editor's workspace in order to be immediately exploited. The textual editor's code is also available from the CAMEL repository[8] while its installation instructions are available from the project's confluence site[9].

[5] https://confluence.7bulls.eu/display/MEL/11+Modelling
[6] https://www.eclipse.org/cdo
[7] https://bitbucket.7bulls.eu/projects/MEL/repos/camel/browse/camel/examples?at=serverless-oxygen
[8] https://bitbucket.7bulls.eu/projects/MEL/repos/camel/browse/camel?at=refs%2Fheads%2Fserverless-oxygen
[9] https://confluence.7bulls.eu/display/MEL/%5BCAMEL%5D+Camel+2.0+Eclipse+%28oxygen%29+editor+installation

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

- *Web editor*. This is a web-based application developed through Eclipse's RAP technology which enables the form- and tree-based editing of CAMEL models while catering for the coverage of both devops and admin users. Especially for the latter, it enables the editing of organisational models which then also regulate the access to the CDO repository for devops users. Installation details about this web editor can be found in the project's confluence site[10]. A respective documentation based on CAMEL 1.0 for this editor has been incorporated in deliverable D3.3 while an update on this documentation for CAMEL 2.0 will be implanted in the project's confluence site soon. The code of the editor can be found in project's bitbucket[11] which is currently integrated as part of the Melodic platform.

A comparison between these two editors is summarised in Table 15

Table 15: Comparison between the two CAMEL editors

Editor	Model Valid.	Aspect	CDO Integr.	CDO Access Control	Format	Roles	Version
Textual	✓	All apart from execution			Textual (edit.), XMI (transf.)	devops	2.0
Web	✓	All apart from execution	✓	✓	Both	devops & admin	2.0

As it can be seen from the above table, the two editors are quite comparable as they support model validation and all aspects apart from the system-focused execution one while they at least cover the devops users. They also support the latest version of CAMEL. The textual editor allows the editing of models in the CAMEL's textual syntax while it produces an XMI form of the

10 https://confluence.7bulls.eu/display/MEL/%5BCAMEL%5D+CAMEL+2.0+Web+Editor+Installation?src=contextnavpagetreemode
11 https://bitbucket.7bulls.eu/projects/MEL/repos/camel_web_editor/browse

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

model which is readable but not writeable. On the other hand, the web editor does not directly enable the editing of CAMEL models in any form but is able to store the models in CDO as well as to export or import them in any format. Further, it has been integrated with the access control mechanisms of the CDO repository thus having the ability to control the access to CDO models in that repository based on the organisation model that is edited by the admin user.

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

6 External Interfaces and Extensibility

6.1 Overview

The Melodic platform has been built in a way that it can cater the needs and plans for the extensibility in the future. Each of the main component groups in the Melodic, Upperware, Executionware, and Modelling interfaces, are designed with clear interfaces so that new functionality can be plugged in by both the platform maintainers and third party developers. At the component level, ESB integration defines consistent interfaces through which components interact with each other. This simplifies communication as the individual components need to conform only to the standard communication interface, and not implement direct communication between each other. Integration through ESB provides platform for the extensibility, and reusability.

In the following, we discuss extensibility features of the Melodic platform for each of the main component groups.

6.2 Upperware

The main extensibility areas in the Upperware involve adding new optimizations solvers for calculating Cloud application placement and optimization solutions as well as new DLMS algorithms for data management algorithms, supporting new storage technologies, writing custom utility functions to optimize solutions as per user defined utility criteria, and extending CAMEL for, e.g., the coverage of new cloud-related domains/aspects. Each of these Upperware areas are discussed in detail in the following sub-sections.

Adding a new optimization solver

The solvers all have the same interface towards the Metasolver, as described in Section 3.2. This makes it easy to insert new solvers and this makes Melodic an attractive platform for further research. There are currently on-going student projects to build and test two new solvers:

- One algorithm based on a Markov Chain Monte Carlo (MCMC) sampling method called *Parallel Tempering*, like simulated annealing, that runs a set of parallel sub-searches over separate domains. For this reason, it is better than simulated annealing to avoid being trapped at local minima.

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

- The other solver uses evolutionary methods and Genetic Algorithms to search for a better, near-optimal solution.

The "*No free lunch theorem*" simplistically states that there is no optimization algorithm that is uniformly the best for all different optimization problems [11] . Having an extensible architecture and demonstrating that new solvers may be easily added if needed by certain cloud application management problems are therefore essential features of the Upperware for the acceptance of the Melodic platform as a generalized tool. Further, some exciting research directions can be also followed like establishing a sophisticated solver selection logic which could even include considering combinations of solvers to better solve a given optimisation model over cloud application placement. Such a logic could further increase the added-value of the Meta-Solver as the implementer of this logic as well as of the Melodic platform being able to produce truly optimal cloud application placement solutions according to the current context at hand.

Implementing a new DLMS algorithm

DLMS employs algorithms to assign utility values to the proposed deployment solutions by considering the characteristics of the current application and data components deployment, the candidate deployment topology, and an internal knowledgebase kept by the DLMS algorithms (such as historical data access patterns). Each of the DLMS algorithms returns a utility value to the UtilityGenerator component that are used in the application's utility function in order to select the optimal deployment solution among the proposed candidate ones.

Each algorithm is a *plugin* in the DLMS controller. In this way, new DLMS algorithms can be designed and incorporated in the DLMS. A new DLMS algorithm can be added by implementing the `AlgorithmRunner` interface defined in the DLMSController, as shown in Figure 2: The DLMS Controller interface.

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

```
/**
 * Interface for algorithm runner classes.
 */
public abstract class AlgorithmRunner {

    /**
     * Initializes the runner instance with a reference to the application to make the use of Spring injected repositories etc. possible.
     */
    public abstract void initialize(DlmsControllerApplication application);

    /**
     * Returns an utility value of all the collected results.
     * It is the runner's responsibility to make sure that the results are cleared afterwards (if necessary).
     */
    public abstract double queryResults(DlmsConfigurationConnection diff);

    /**
     * Method to run an algorithm.
     * It is the runner's job to keep the result(s).
     * Takes an individual number of parameters.
     */
    public abstract int update(Object... parameters);

    /**
     * Get DlmsConfigurationElement matching the connection component name
     */
    public DlmsConfigurationElement getComp(Collection<DlmsConfigurationElement> deployed, String toComp) {
        return deployed.stream()
                .filter(dlmsConfigurationElement -> dlmsConfigurationElement.getId().equals(toComp))
                .findFirst()
                .orElse(new DlmsConfigurationElement());
    }

    /**
     * Check DlmsConfigurationElement is not empty
     */
    public boolean isEmpty(DlmsConfigurationElement element) {
        return element.getId()==null;
    }

}
```

Figure 2: The DLMS Controller interface

Once an algorithm has been implemented, it can be *plugged* into the DLMSController by providing the respective name and class implementing the algorithm in the DLMS properties file. For example, the DLMS properties file will be appended as follows in order to integrate a new algorithm with the class name `eu.melodic.dlms.algorithm_runners.Algo_SourceAwarenessRunner`:

```
dlms.algorithms[AlgoNUM].name=AlgorithmName
dlms.algorithms[AlgoNUM].className=eu.melodic.dlms.algorithm_runners.Algo_SourceAwarenessRunner
```

where AlgoNUM is the current number of DLMS algorithms, including the new one.

Supporting a new storage technology

The DLMS uses Alluxio[12] (formerly Tachyon [12]) as the middleware for the storage technologies supported by the Melodic cross-Cloud platform.. Alluxio is a rapidly growing open source virtual

[12] https://www.alluxio.io/

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

distributed storage system enabling big data applications to interact with data from a variety of storage systems and technologies. A new storage technology can be supported in the Alluxio by writing an implementation for the *Under Store* using Alluxio's *Under File Storage Extension API.*

Building a new under storage connector involves implementing the respective under storage interfaces (`UnderFileSystem` and `UnderFileSystemFactor`interfaces), declaring the service implementation, and bundling up the implementation and transitive dependencies in a jar file. A step-to-step detailed guide is available at Alluxio's developer pages[13].

Utility functions

The utility function is defined by the application owner in the CAMEL model describing the application's architectural model. This utility function currently comprises three main metric variables which are added through a weighted sum approach: (a) one computing the utility as perceived by the user; (b) one calculating the data reconfiguration penalty; (c) one calculating the application topology reconfiguration penalty. As such, the user has the capability to change the weights involved in the computation of the overall utility to better match his/her requirements and preferences. For instance, in case of a non-big-data application, the weight of the data reconfiguration penalty can be zero. However, we do foresee in the near future, that it will be extremely convenient if the weights to the different variables of the utility function are dynamically inferred. For instance, if there is a significant violation of an application SLO, a higher weight could be given dynamically to the first variable, the utility perceived by the user, with respect to the weights of the two other variables as there will be a higher need to remedy this non-functional fault than to reduce any kind of reconfiguration penalty/cost.

Besides, there are three other extensions which are foreseen, related to some parts of the utility function or the way it is computed from node candidates:

1. The selection function for the matching Node Candidates. If there is more than one Node Candidate matching the requirements of an application's component to be deployed, only the Node Candicates' price is currently taken into account and the Node Candidate of the matching set with the lowest price is selected. There are plans under discussion for defining this selection function as a separate utility function in CAMEL, potentially covering additional criteria apart from price, and when such mechanisms are in place, then it will be easy to change or extend the selection function.

[13] https://docs.alluxio.io/os/user/stable/en/ufs/Ufs-Extension-API.html

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

2. The data reconfiguration penalty calculated by the Data Lifecycle Management System (DLMS) is implemented as an aggregation of a set of replaceable algorithms of DLMS, which take the form of DLMS plug-ins. Hence, to change the way the penalty is calculated requires to modify the weights given to the (metric) variables representing these algorithms as well as potentially implementing and adding a new DLMS plug-in algorithm in the form of a metric variable, if needed. Logically speaking, the use of a weighted sum approach for the calculation of the overall metric variable representing the data reconfiguration penalty gives the strength to the user to, e.g., activate and de-activate different DLMS penalty calculation algorithms as needed through the use of corresponding (relative) weights as well as to modify their relative importance. While the incorporation of new DLMS plug-in algorithms witnesses the fact that the Melodic platform is extensible to cover the requirements of any kind of user. Nevertheless, as in the case of the overall utility, a promising future work direction could be to dynamically derive the weights given to the different DLMS (penalty) algorithms based on the characteristics of the data being manipulated by the user application at hand and the current application context.
3. The application topology adaptation penalty calculated by the Adapter. This expresses how difficult it is to move from the current application topology to the application topology induced by the new, proposed configuration. This is a multi-dimensional decision problem where factors like the number of machines that must be stopped or started counts, where the Cloud providers hosting the old and the new machines matter, where the difficulty in establishing secure connections among virtual machines in the topology must be calculated, and where the state the components to be redeployed may count; e.g. some must be restarted, which is alright for stateless components, some can be check-pointed albeit as a cost in time and storage and re-started, and some must be closed and re-started. Calculating a single penalty value in the unit interval must be done by the Adapter as it is intrinsically connected with the adapter's operation. To this end, by considering the current solution adopted by Melodic, further research is needed in order to explore if the relevant concepts can be generalized and what the generic problem dimensions are. Then one faces the challenge how to define this in CAMEL through the exploitation of the corresponding problem dimensions.

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

As indicated in section 6.4, CAMEL can be easily extended by following a certain process. It is a subject of further research whether CAMEL needs to be enhanced to cover the modelling of any kind of reconfiguration or utility-based metric variable. However, it needs to be stated that based on the experience from the Melodic project, CAMEL seems to be generic enough to cover any kind of metric or (metric) variable. As such, the main challenge then remains whether the Melodic platform is able to support the CAMEL metric/variable specifications. This maps to the ability to implement and integrate the relevant leaf component metrics/variables. Thus, this goes down to the ability of the application owner to implement such metrics, when needed, as well as of the Melodic platform to integrate such metrics, irrespectively of whether these are internal or external to the application topology.

6.3 Executionware

The modular architecture of the Executionware (i.e., the Cloudiator framework[14]) relies on the well-established software technologies OpenAPI[15] for its central REST interface and Apache Kafka for the internal message based communication between Cloudiator agents [2, p. 1], [9, p. 3], [10, p. 5]. Each agent runs within in the Cloudiator framework and follows the micro-service paradigm with Apache Kafka as central communication channel. Consequently, each individual agent is taking over a small part of the overall Cloudiator feature set, e.g. allocating new nodes, deploying Spark applications or orchestrating the monitoring. In this way, the feature set of the Executionware can easily be extended by enhancing the functionalities of existing agents or adding new agents to the Cloudiator framework.

Supporting a new Cloud provider

The Node-Agent of Cloudiator presents the central agent to interact with cloud provider APIs. Therefore, it builds upon the concepts of the provider agnostic interface mapper [2] that are implemented as the Sword abstraction layer in Cloudiator[16].

Consequently, adding new cloud providers requires the extension of the abstraction layer component Sword with the mapping of the provider specific API calls and the generic interface provided by Sword.

In case that the added cloud provider supports extra heterogeneous resource kinds besides virtual machines, such as containers, the Node-Agent needs to be extended accordingly.

[14] http://cloudiator.org/
[15] https://swagger.io/docs/specification/about/
[16] https://github.com/cloudiator/sword

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

Adding a new data processing framework

The Executionware can be easily extended with additional data processing frameworks. The features of each data processing framework are encapsulated in its dedicated agents, i.e., the Spark-Agent provides the required features to orchestrate an Apache Spark cluster [10]. Consequently, adding a new data processing framework is twofold.

First, it requires the implementation of a new data processing agent. The respective agent needs to provide two functionalities; (i) orchestrating the deployment of the data processing framework and (ii) implementing the business logic to submit data processing processes to the data processing framework.

Second, the task description of the Executionware's REST-API needs to be extended by adding the new data processing interface[17] in the Swagger description[18]. The Swagger tool support enables the automatic creation of the respective REST endpoints at the Executionware REST-API. Finally, the required Protobuf messages[19] need to be defined in Cloudiator[20] to enable the communication between the REST interface and the new newly created data processing agent.

D4.5 [10] details the extension of the Executionware with the support for the data processing framework Apache Spark[21] by implementing the respective Spark Agent.

6.4 Extending CAMEL

CAMEL is a live language that is being maintained by a very active community of users. In this respect, it undergoes constant changes, especially in the context of European projects like Melodic. In this section, we will attempt to explicate the actual process for updating CAMEL that starts from its meta-model (ecore model) and goes until the editors produced for it. This process is depicted in the Figure 3. As it can be seen, the process comprises 4 main tasks, the first of which are sequentially executed while the last two can be in parallel. Please note that as the last two steps concern the CAMEL editors, a potential adopter of CAMEL might choose to support the evolution of just one editor from the two. In this case, one of these two steps will be actually executed.

[17] http://cloudiator.org/rest-swagger/#operation/addJob
[18] https://github.com/cloudiator/rest-swagger
[19] https://developers.google.com/protocol-buffers/
[20] https://github.com/cloudiator/common
[21] https://spark.apache.org/

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

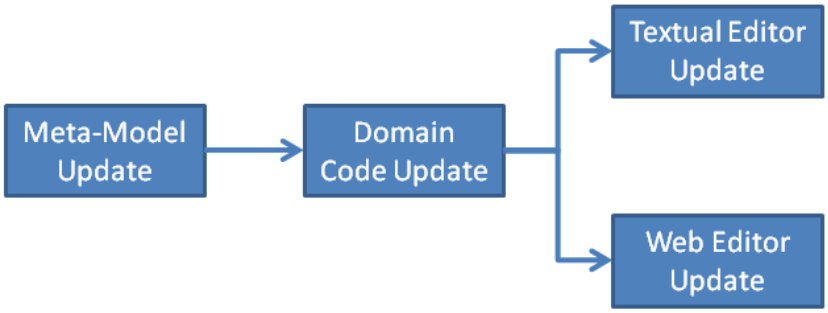

Figure 3: CAMEL update process

In order to support the updating of the CAMEL meta-model (and its textual editor), the respective code[22] needs to be imported in the Eclipse Environment. This importing should include all the relevant directories (having the camel prefix).

The meta-model updating / evolution of CAMEL includes the editing of its ecore model. For this editing, the Eclipse tree-based (named as "Sample Ecore Model Editor") or OCL (named as "OCLinEcore Editor") editors can be used. The tree-based editor visualises the meta-model in the form of a tree and allows its updating and expansion by supplying respective differentiated forms for the editing of different types of meta-model elements. On the other hand, the OCL editor is a textual editor which enables both the updating of the meta-model and its respective OCL rules. For non-expert users, it is recommendable to start with the tree-based editor and then, when they have finished the changes in the meta-model, switch to the OCL editor in order to update or extend the set of OCL rules already defined for CAMEL. While an expert user could adopt immediately the OCL editor in order to both update CAMEL's meta-model and its OCL rules. To open any of these two editors, the user should right click on the camel.ecore file, choose the "Open With" option and then the option mapping to one of these two editors.

We should highlight at this point that depending on the meta-model updating, it might not be required to change the OCL rules. This could be checked by launching the CAMEL ecore model in the OCL editor. In that case, this editor features an error highlighting facility which immediately identifies problematic OCL rules (that might be invalidated by the performed meta-model changes). However, it should be also noted that it could be also the case that OCL rules need to be modified at the semantic level. In this case, the modeller should be able to check all the relevant OCL rules in order to assess the necessity for updating them.

We foresee that there can be two types of changes that can be performed in the CAMEL meta-model:

[22] https://bitbucket.7bulls.eu/projects/MEL/repos/camel/browse/camel?at=refs%2Fheads%2Foxygen-new

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

- *change of existing packages* (mapping to respective aspects covered by CAMEL): in this case, the modeller should reassure that the changes are correct and appropriate and do not invalidate any from the packages involved. Eclipse can also assist in this case as both the tree-based and OCL editors highlight syntactic errors. At the semantic level, the modeller should be careful not to make changes that break the aspect/sub-DSL integration in CAMEL. For instance, moving one concept from the domain/aspect it actually belongs to another aspect is wrong. Similarly, creating a respective concept in a wrong domain is a mistake. In this respect, it is advocated that the modeller first reads the CAMEL documentation and the extent of each of its domains before making any changes.
- *incorporation of an additional package*: this is a quite possible scenario in the case of extending CAMEL through the coverage of a new aspect. In this case, this new aspect should be modelled as a sub-package (sub-DSL) of the main package of CAMEL. As in the context of the previous case, the modeller should be careful not to create any overlapping with the other packages/domains. In addition, he/she should create cross-references to both intra-package and other package elements to support the proper integration of the new package in CAMEL with respect to the existing ones. Finally, to the extent possible, the modeller should create OCL rules that cover both the new package as well as its relation to the other packages.

It should be noted that based on the new design of CAMEL, any CAMEL model is assorted with template models, including those conforming to the meta-data schema. In this respect, in case that the user requires to modify some metadata related to his/her context, then he/she should update or extend the metadata (template) model. More details about how this can be done are supplied in deliverable D3.1. However, it should be noted that this metadata (template) model can also be updated through CAMEL's textual editor, due to the integration of the structure of the metadata schema in the form of a new CAMEL package.

After CAMEL's meta-model has been updated, including the incorporated OCL rules, if needed, the respective user should also update the domain code of CAMEL. This can be easily done by updating first the camel.genmodel file of CAMEL. This is included in the CAMEL's code repository[23]. Eclipse has also a great support for this. In the context of the current workspace, the

23 https://bitbucket.7bulls.eu/projects/MEL/repos/camel/browse/camel/camel/model/camel.genmodel?at=refs%2Fheads%2Foxygen-new

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

user has just to perform a right click on that file and choose the option "Reload". The user then needs to follow the UI-based instructions indicated to him/her. Once this update takes place, the genmodel is launched in the form of a new tab. The user should just right click on the Camel element displayed and choose the option "Generate Model Code". After that, the domain code is automatically updated. In case that Java-related errors pop up in the end, these can be due to the deletion of CAMEL elements. In that case, it is recommended to delete all the classes related to those elements. Alternatively, the user can delete the src directory before regenerating the model/domain code.

The update of the textual editor is threefold:

- on one hand, the user should update the textual syntax of CAMEL by going into the camel.dsl directory, then into the src/camel/dsl directory and opening the CamelDsl.xtext file. It should be noted that the user should have appropriate knowledge and expertise with respect to Xtext[24] in order to perform this.
- on the other hand, the user should update any textual editor feature, when needed, either due to the update of CAMEL (in case that this update impacts this feature) or due to the user requirements (such a feature was missing and needs to be enabled). Existing features can be updated by modifying the code that is included in the src/camel/dsl subdirectories. Again essential knowledge of Xtext is needed also for performing these changes.
- Finally, due to new textual editor features related to the visualisation of documentation-oriented details via hoving over the respective CAMEL elements, the user should attempt to modify the following, when needed, inside the camel.dsl.ui directory:
    - the documentation.xlsx file related to the supply of the documentation-oriented information for each CAMEL element (class in particular), situated inside the input subdirectory
    - the MyKeywordHovers.xtend file situated inside the src/camel/dsl/ui/hover directory by catering for the addition of new CAMEL classes and the deletion of respective code for those CAMEL classes removed

[24] https://www.eclipse.org/Xtext

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

Finally, the update of the web editor is quite involved as it requires the modification of the respective code in various places. In order not to go into many technical details, we just outline here what are the main changes needed at the higher level of abstraction:

- updating of existing queries for fetching information related to existing CAMEL classes
- addition of new queries for new CAMEL classes
- updating perspectives and views for existing CAMEL aspects
- creating new perspective and views for new CAMEL aspects and registering them
- updating of the respective access flow for the menu-oriented UI elements subject to the respective role that has been authenticated initially. Examples for this are the following: (a) for a devops user, any perspective apart from the organisation can be enabled only when a respective application is created or launched; (b) when a certain aspect-oriented model is launched, the respective option in the menu related to the aspect's perspective dealing with the opening or importing of such a model is deactivated/disabled.

In the cases of both the textual and web editor, there are also some template models (e.g., location model) that are re-used in order to reduce the modeller's effort. Depending on the places where the respective changes have occurred in CAMEL, such models need to be updated. In that case, it is recommended to use the updated textual editor to perform such changes. If the user has not opted towards updating that editor, then the default tree-based editor of CAMEL can be used for this purpose.

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

7 Conclusions

This document has attempted to detail the REST APIs consumed and offered by the MELODIC platform components. This enabled to raise the understanding with respect to: (a) the way these components interact with each other at the interface layer; (b) how external software products can be integrated with the MELODIC platform. It also highlighted what are main extension points of particular MELODIC platform components which could enable to enhance the current platform functionality and potentially enable to support new features. As the MELODIC platform is attempted to be enhanced in different ways in new European project proposals, this can be considered as the baseline for such an enhancement. Finally, this document shortly analyzed CAMEL 2.0, the newest version of CAMEL, which has been adopted by the latest releases of the MELODIC platform.

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

References

[1] Yiannis Verginadis *et al.*, "D3.4 Workload optimisation recommendation and adaptation enactment," Melodic Project Deliverable, Jan. 2019.
[2] Daniel Baur and Daniel Seybold, "D4.1 Provider agnostic interface definition & mapping cycle," Melodic Project Deliverable, Sep. 2019.
[3] Yiannis Verginadis *et al.*, "D2.2 Architecture and Initial Feature Definitions," Melodic Project Deliverable, Feb. 2018.
[4] A. P. Achilleos *et al.*, "The cloud application modelling and execution language," *J. Cloud Comput.*, vol. 8, no. 1, p. 20, Dec. 2019, doi: 10.1186/s13677-019-0138-7.
[5] Chappell, David, *Enterprise Service Bus*. O'Reilly Media, Inc., 2004.
[6] J. Sutherland and W.-J. van den Heuvel, "Enterprise application integration and complex adaptive systems," *Commun. ACM*, vol. 45, no. 10, Oct. 2002, doi: 10.1145/570907.570932.
[7] Pieter Hintjens, *ZeroMq: Messaging For Many Applications*. O'Reilly Media, 2013.
[8] Yiannis Verginadis *et al.*, "D3.5 Melodic Upperware," Melodic Project Deliverable, Nov. 2019.
[9] Daniel Baur and Daniel Seybold, "D4.3 Resource Management Layer Prototype," Melodic Project Deliverable, Aug. 2018.
[10] Daniel Seybold, Daniel Baur, Floran Held, and Paweł Skrzypek, "D4.5 Data Processing Layer Prototype," Melodic Project Deliverable, Jan. 2019.
[11] David H. Wolpert and William G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, 1997, doi: 10.1109/4235.585893.
[12] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Tachyon: Reliable, Memory Speed Storage for Cluster Computing Frameworks," in Proceedings of the ACM Symposium on Cloud Computing - SOCC '14, Seattle, WA, USA, 2014, pp. 1–15, doi: 10.1145/2670979.2670985.

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