Title:

D2.2 Architecture and Initial Feature Definitions

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

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

Grant Agreement No.:
731664

Duration:
1 December 2016 -
30 November 2019

www.melodic.cloud

Deliverable reference:
D2.2

Date:
15 May 2018

Responsible partner:
Simula Research Laboratory

Editor(s):
Feroz Zahid

Author(s):
Yiannis Verginadis, Geir Horn, Kyriakos Kritikos, Feroz Zahid, Daniel Baur, Paweł Skrzypek, Daniel Seybold, Marcin Prusiński, Somnath Mazumdar

Approved by:
Jörg Domaschka

ISBN number:
N/A

Document URL:
http://www.melodic.cloud/deliverables/D2.2 Architecture and Initial Feature Definiton.pdf

Abstract:

This document presents the final Melodic architecture and its initial feature definitions. We detail key Melodic components, describe their internal architecture and interfaces, and present how these components work and interact together to realise an efficient middleware platform enabling optimised deployment of data-intensive applications in Cross-Cloud environments.

The Melodic platform is conceptually divided into three main component groups: the *Upperware*, the *Executionware*, and the tools and interfaces for data modelling, application modelling, and user-platform interactions. At the heart of the platform lies a feedback-driven adaptation loop where the current application deployments are continuously monitored, analysed, and reconfigured, if needed, to ensure that the user-defined constraints and requirements are optimally satisfied for a given application. The user requirements and constraints, related to both applications and data, are captured using a domain-specific language, CAMEL, which encapsulates all relevant aspects required for modelling data-intensive applications and their configurations in heterogeneous Cross-Cloud environments. The job of the Upperware is to calculate the optimal data placements and application deployments on aggregated Cross-Cloud resources in accordance with the specified application and data models in CAMEL as well as in consideration of the current workload situation and involved costs. The actual Cloud deployments are carried out through the Executionware, which is capable of managing and orchestrating diverse Cloud resources. The Executionware also enables support of cross-cloud monitoring of the deployed data-intensive applications.

Expanding on the generic and use-case specific requirements gathered in the deliverable *D2.1 System Specification Document*, we also enlist key Melodic capabilities and features in this document. In addition, a brief description of the timeline of planned Melodic releases and the corresponding features is also supplied.

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

Document	
Period Covered	M1-14
Deliverable No.	D2.2
Deliverable Title	Architecture and Initial Feature Definitions
Editor(s)	Feroz Zahid
Author(s)	Yiannis Verginadis, Geir Horn, Kyriakos Kritikos, Feroz Zahid, Daniel Baur, Paweł Skrzypek, Daniel Seybold, Marcin Prusiński, Somnath Mazumdar
Reviewer(s)	Antonia Schwichtenberg, Daniel Baur, Amir Taherkordi
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	78

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

Table of Contents

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

List of Figures

1 Introduction

Modern enterprises increasingly rely on hybrid Cloud solutions to meet their computational demands by acquiring additional resources from public Clouds dynamically as per their needs. International Data Corporation (IDC), which is a leading market-research firm, in its CloudView Survey 2017[1], reported that 87% of Cloud users have adapted a hybrid cloud strategy and 56% of the users use more than one type of Cloud deployment. In general, Cloud federation [1] enables end users to integrate segregated resources from different Cloud systems. Federated Clouds offer more freedom to the Cloud users and increase the granularity of choices in the application deployment. The key objective of the Melodic project is to provide a middleware platform that enables data-aware application deployment on geographically distributed and federated Cloud infrastructures. The lack of data-awareness in Cross-Cloud deployments, in particular, leads to non-optimal application performance as well as hinders the full potential utilisation of the acquired resources from the Cloud infrastructures [2]. The Melodic middleware platform aims to act as an automatic DevOps solution for data-intensive Cloud applications covering modelling, deployment, configuration, and autonomic adaptation of such applications in distributed, heterogeneous, and dynamic Cross-Cloud environments.

In this document, we present the architecture of the Melodic middleware software platform and its key components. As also described in the Melodic System Specification Document [3] , the Melodic project reuses technology, components, and research results from selected open-source projects. The components of the selected open-source projects were evaluated carefully to assess their integrability in the Melodic, and the evaluation results are summarised in the System Specification Document. In particular, the architecture of the Melodic platform is greatly influenced by that of the PaaSage project [4] to ensure that the available components from PaaSage are utilised to the extent possible, and the target of the new developments in the Melodic project remains the unique requirements arising from the needs of data-intensive applications in Cross-Cloud environments. The PaaSage project lacked Cross-Cloud data management and data-aware application deployments limiting its use for the data-intensive applications in the Cloud, and data-aware deployments and processing in general. The Melodic project adds new components needed for satisfying requirements of data-aware Cross-Cloud deployments, introduces consistent and modular component integration, and rectifies design issues based on the lessons learned from the PaaSage project.

[1] https://www.idc.com/getdoc.jsp?containerId=prUS42878417

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

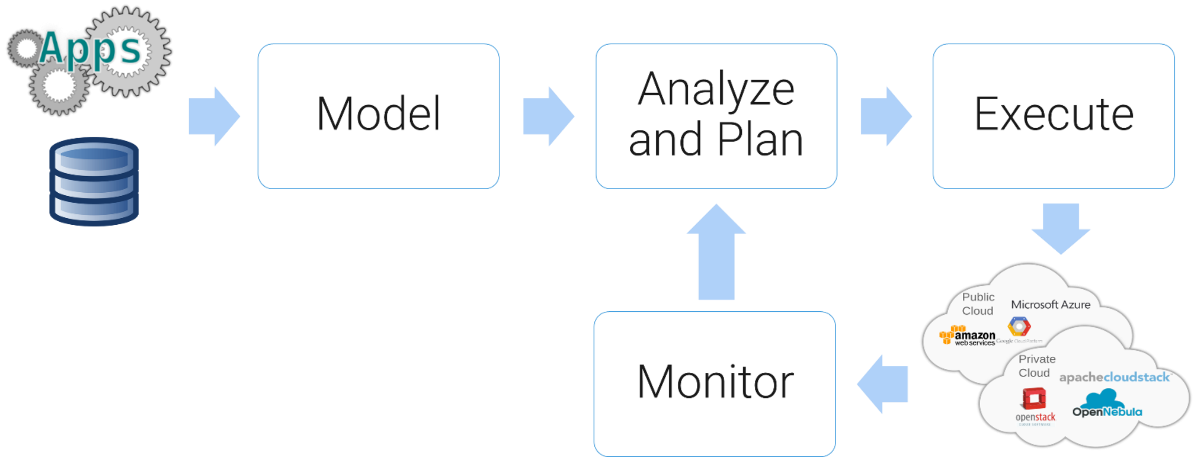


Figure 1: Melodic enables an automatic DevOps system for Cross-Cloud application deployment and adaptation

Cross-Cloud[2] application deployments comprise of resources acquired from multiple administrative domains, ranging from locally deployed private Cloud infrastructures to externally managed public Cloud offerings [5]. A data-intensive application, like any other Cloud application, corresponds to a specific component deployment topology, and has certain application- and user-specific deployment requirements, such as hardware/OS requirements, security and Quality-of-Service (QoS) constraints, allocated Cloud budget, and scalability policies and rules. The same goes for the data sources. The user data, for instance, may need to adhere to specific location constraints and confidentiality policies in place. Following a Model-driven Engineering (MDE) approach [6], before a given data-intensive application and corresponding data sources are ready to be deployed by the Melodic middleware platform onto dynamically acquired Cross-Cloud resources, they are modelled so that the aforementioned requirements and constraints can be formally specified, and hence utilised by the deployment reasoning process. As depicted in Figure 1, modelling of an application is the first step of the automated DevOps system offered by the Melodic middleware platform for data-intensive applications.

After applications has been modelled, the reasoning part of the Melodic middleware finds out most effective placement of the applications onto Cross-Cloud resources. Furthermore, to cater for performance unpredictability and dynamicity challenges in the Cloud, applications deployed through Melodic are continuously monitored and adapted, to make sure that the current deployment corresponds to the best possible configuration according to the current Cloud

[2] We use the term *Cross-Cloud* to refer to application deployments where multiple Cloud platforms are simultaneously used to deploy application components. The term *Multi-Cloud* is also popular, though. We differentiate Multi-Cloud scenarios from Cross-Clouds – in Multi-Clouds, applications are capable of being deployed on different Cloud platforms, but one at a time, contrary to the Cross-Cloud deployments of application components on segregated Cloud platforms at the same time.

resource availability, reliability, and performance, user requirements, constraints, and the execution context. The Melodic middleware platform implements a self-adaptive[3] deployment and reconfiguration system through a feedback-driven control loop. As shown in Figure 1, the feedback loop in Melodic is implemented through a Monitor-Analyse-Plan-Execute (MAPE[4]) [8] based adaptation loop. Application deployments are continuously monitored and analysed, and if the current deployment is no longer optimal, a new deployment solution is calculated, and the adaptation is planned and enforced. Thanks to the adaptation mechanism implemented by Melodic, data-intensive applications are both optimally deployed over Cross-Cloud resources and are also kept optimised when the application or Cloud context changes. In addition to applications deployment, data management is also performed in an intelligent way to cater for the unique Cross-Cloud needs. For large-scale distributed applications, optimisation of the complete data lifecycle, comprising of distinct phases including data acquisition, preparation, analysis, integration, aggregation, and its final representation, becomes complex and multi-dimensional [9]. Solutions targeting individual phases often yield contradictory management decisions. Moreover, as application deployment decisions are affected by data placement and migration methodologies in effect, and vice versa, it is important to couple data and computation modelling together. The Melodic platform enables the holistic management of complete data life-cycle by complementing the middleware platform with the holistic Cross-Cloud data life-cycle management solution.

1.1 Scope of the Document

This document presents the final Melodic architecture and its initial feature definitions. The document details key Melodic components, their internal architecture, key functionalities, and inter-component interactions and corresponding interfaces. A list of features targeted by Melodic, at different releases during the project tenure, are also provided. This document is intended for the general audience interested in learning about the architecture of the Melodic platform and its salient features. Parts of the document requires basic understanding of how Cloud computing systems and distributed applications work. The document, at places, refer to the System Specification Document, the Melodic project deliverable *D2.1, System Specification Document* [3], however, such references are clearly identified wherever possible.

[3] Self-adaptivity is defined as the property of a system to autonomously evaluate and change its behaviour in response to a changing environment [7].

[4] MAPE is also more precisely referred to as MAPE-K loop, with *K* representing the shared knowledge-based required to implement all stages of the monitor, plan, and execute sequence.

1.2 Structure of the Document

The rest of this document is structured as follows. In chapter 2, an overview of the Melodic architecture and its main components is provided together with an overview of the application and data modelling interfaces. The detailed architecture of the two core component groups of Melodic, the Upperware and the Executionware, is provided in chapter 3 and chapter 4, respectively. In chapter 5, we briefly analyse auxiliary services used by the Melodic components. In chapter 6, we list the main Melodic capabilities and provide its salient features together with a summary of the roadmap for the Melodic software releases. We conclude in chapter 7.

2 Architecture Overview

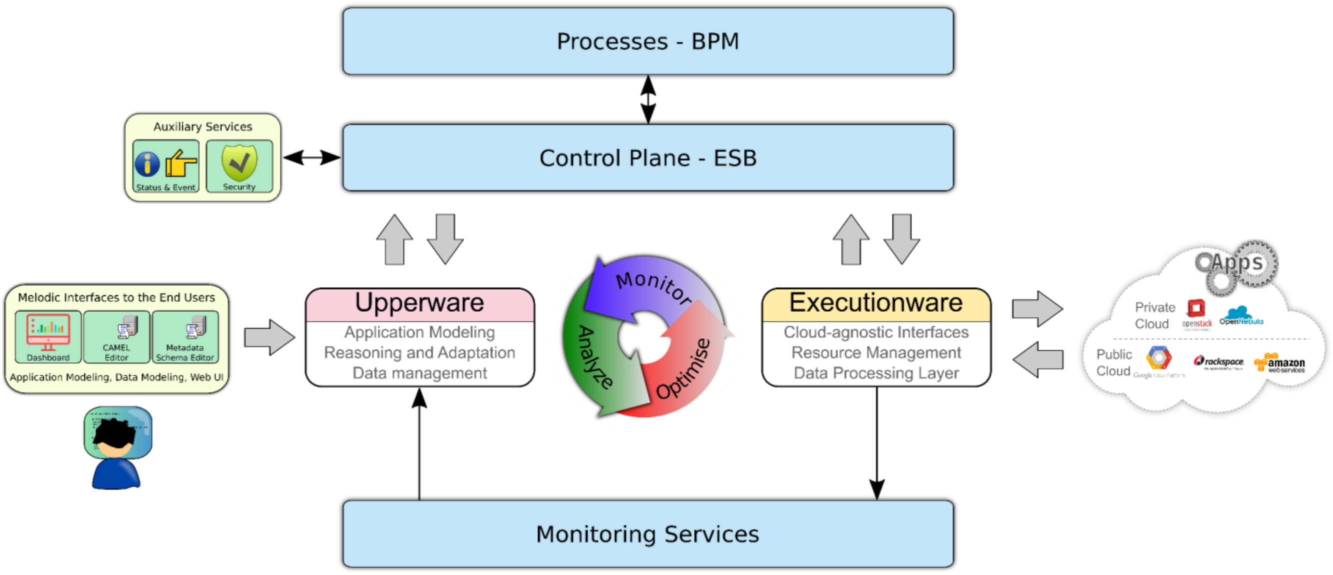


Figure 2: Overview of the Melodic architecture

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 [10], 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 as well as in consideration of 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, and it also enables support of cross-cloud monitoring of the deployed applications. Besides the three main component groups, two auxiliary services, for enabling unified and integrated event notification mechanism and to warrant secure operations with the Melodic platform, respectively, are also designed. An overview of the Melodic architecture is given in Figure 2.

2.1 Component Integration

To realise 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 the process, and thus, is reliable and transactional. The Monitoring Plane, on the other hand, deals with a large amount of monitoring data and thus requires fast data transfers. Based on a detailed evaluation of the integration and adaption requirements of each plane, a hybrid solution with two different integration methods is chosen to ensure that the requirements of the two different planes are fully fulfilled.

The Control Plane implementation is based on an Enterprise Service Bus (ESB) architecture [11] with process orchestration 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. ESB integration with BPM is the a flexible integration method allowing both easy modifications to the process workflow, as well as reusability of services exposed by a given component in various processes and features of the system [12]. For the Monitoring Plane, a queue based message broker, is employed ensuring fast message delivery [13].

2.2 Application and Data Modelling

As discussed in chapter 1, Melodic employs an MDE approach and the applications and corresponding data sources are first modelled before the Melodic Upperware can reason about their optimal Cross-Cloud deployments. Application and data modelling comprises formal specification of application components, their interactions, data sources, requirements and constraints pertaining both applications and data, as well as user-defined deployment goals and objectives.

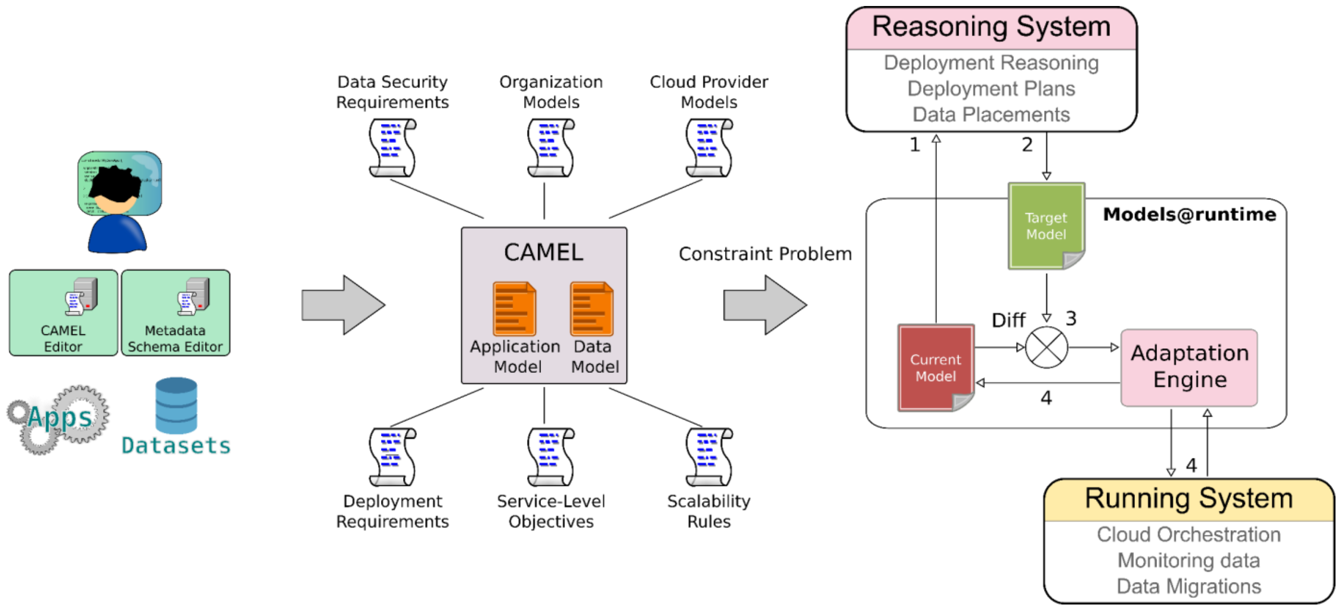


Figure 3: An Overview of the CAMEL modelling and the models@runtime approach taken by the Melodic platform

The PaaSage project created CAMEL, a domain-specific language that captures a rich set of design-time and runtime aspects like application deployment requirements, service-level objectives, scalability rules, and security considerations in Cross-Cloud deployments. CAMEL is similar to the Topology and Orchestration Specification for Cloud Applications (TOSCA) [14], which allows users to specify the components comprising the topology of Cloud-based applications along with the processes for their orchestration. However, a key difference between TOSCA and CAMEL is that while TOSCA can only be used at design-time as it supports specification of types of templates only, CAMEL can be used at both design-time and run-time because of its support of the specification of instances too. In the context of the Melodic project, CAMEL is being extended in order to support modelling of both data-intensive applications and datasets as well as the modelling of non-functional terms (properties or metrics) for both data and data-intensive applications. As depicted in Figure 3, CAMEL is a super-DSL which includes multiple DSLs, each focusing on a particular aspect. CAMEL has been designed based on EMF Ecore[5] and Object Constraint Language (OCL). EMF Ecore enables the specification of UML-based meta-models, while OCL constraints accompany such meta-model specification with the coverage of additional domain semantics. Using the provided Melodic interfaces to the end-users that includes a CAMEL editor, application developers create a CAMEL model which is

[5] http://www.eclipse.org/modeling/emf/downloads/

processed by the Melodic middleware. As shown in Figure 3, Cross-Cloud application deployments via CAMEL follow a *model@run-time* [15] approach that extends MDE [6] to the runtime system. In such an approach, the abstract representation of a system and related knowledge, in terms of models, is kept in synchrony with the *running system* so that when a modification to the model is done on-demand, a corresponding change in the system is automatically reflected. The modification to the model is a result from the *reasoning system* which, based on a MAPE loop as described in chapter 1, keeps the target model updated according to the monitoring data obtained from the running system and the specifications described in CAMEL. By exploiting models at both design- and run-time, and by allowing both direct and programmatic manipulation of models, CAMEL enable self-adaptive Cross-Cloud applications which automatically adapt to the changes in their operating environment.

We now briefly describe extensions to the application and data modelling in CAMEL to cover modelling needs of Melodic. An overview of the CAMEL modelling language is provided in [3], while the detailed specification is kept updated at the CAMEL website[6].

2.2.1 Application Modelling

The focus of the application modelling extensions in CAMEL is to capture a rich set of information about the data-intensive applications for allowing both current and future feature implementations in the Melodic platform. The Melodic component implementations, however, may ignore certain modelling aspects and focus on others according to the description of work. Nevertheless, the modelling *vocabulary* can also be extended by Metadata schema, as presented in deliverable *D2.4 Metadata Schema* [16].

The extension performed in the deployment meta-model part of CAMEL are depicted in Figure 4. Concerning internal application components, we have decided to cover the following additional information:

- The data which is consumed and generated by the component. A component can be a data consumer, a data producer, or both.
- Whether a component is long-lived or not. By default, a component is long-lived as this covers the case of ordinary and legacy application components (which live as long as the application that contains them). On the other hand, short-lived components can only be task-based.

[6] http://camel-dsl.org/

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

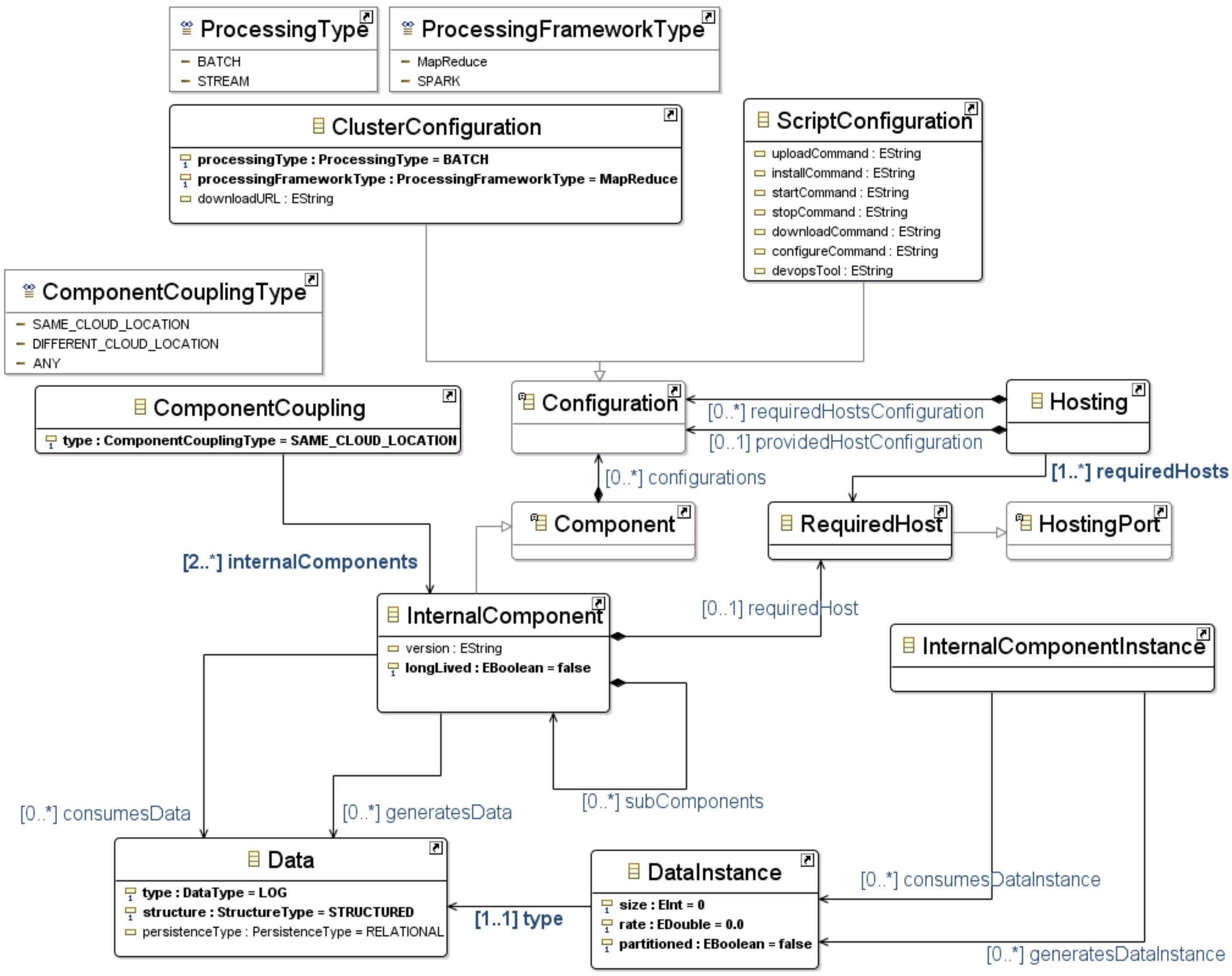


Figure 4: A snapshot of the CAMEL deployment meta-model focusing on the data-intensive application modelling extension

Concerning the configuration of an internal application component, the changes performed are extensive covering configurations of data-processing frameworks, such as MapReduce and Spark.

2.2.2 Data Modelling

The data modelling extensions in CAMEL enables modelling of salient aspects of the data that characterise data-intensive applications and are manipulated by the Melodic Upperware for the reasoning of data placement and migrations. These aspects are captured either directly by the data meta-model (Figure 5), or indirectly by supplying the possibility to the modeller to specify

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

features (i.e., new elements generated at the model level) or attributes (extra from those covered by the data meta-model in CAMEL) that come from the meta-data schema [16]. The coverage of the data aspect in CAMEL has led to the generation of a new meta-model, *Data*, a snapshot of which depicted in Figure 5.

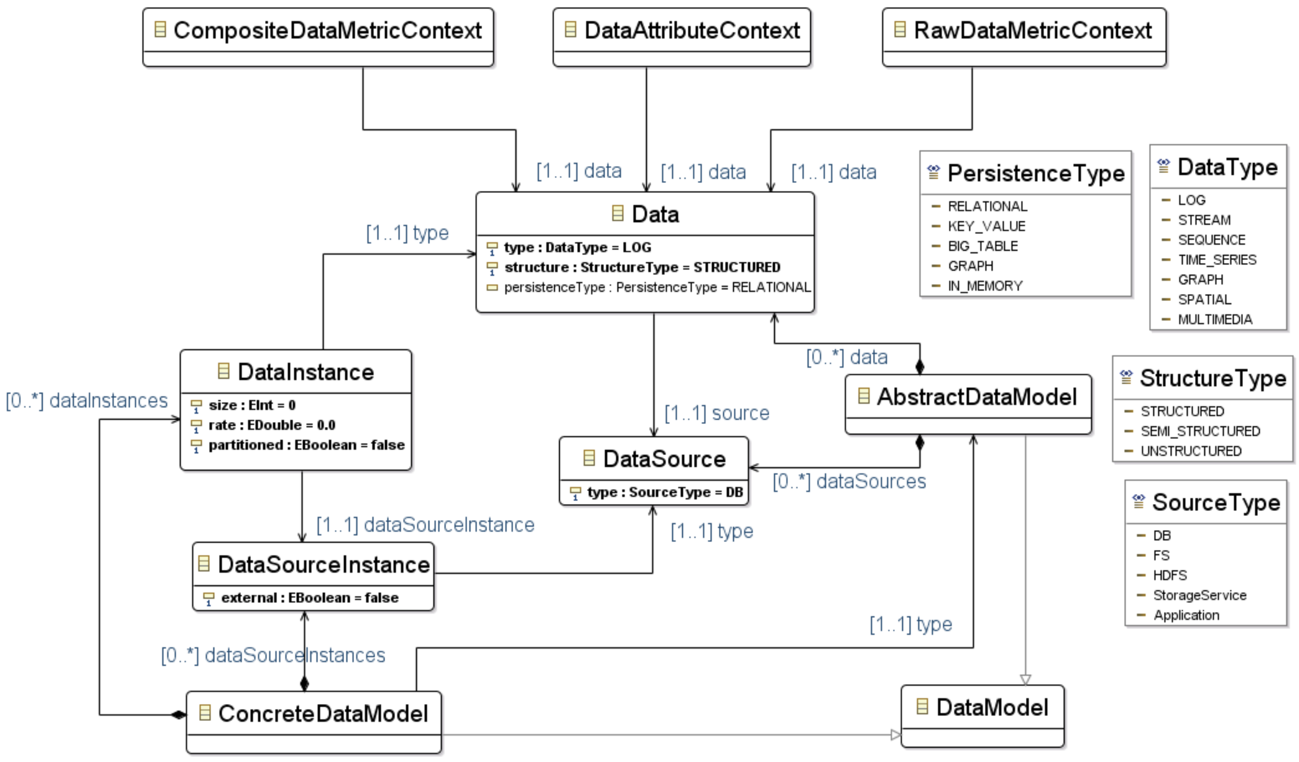


Figure 5: The data meta-model in CAMEL

2.2.3 Models Repository

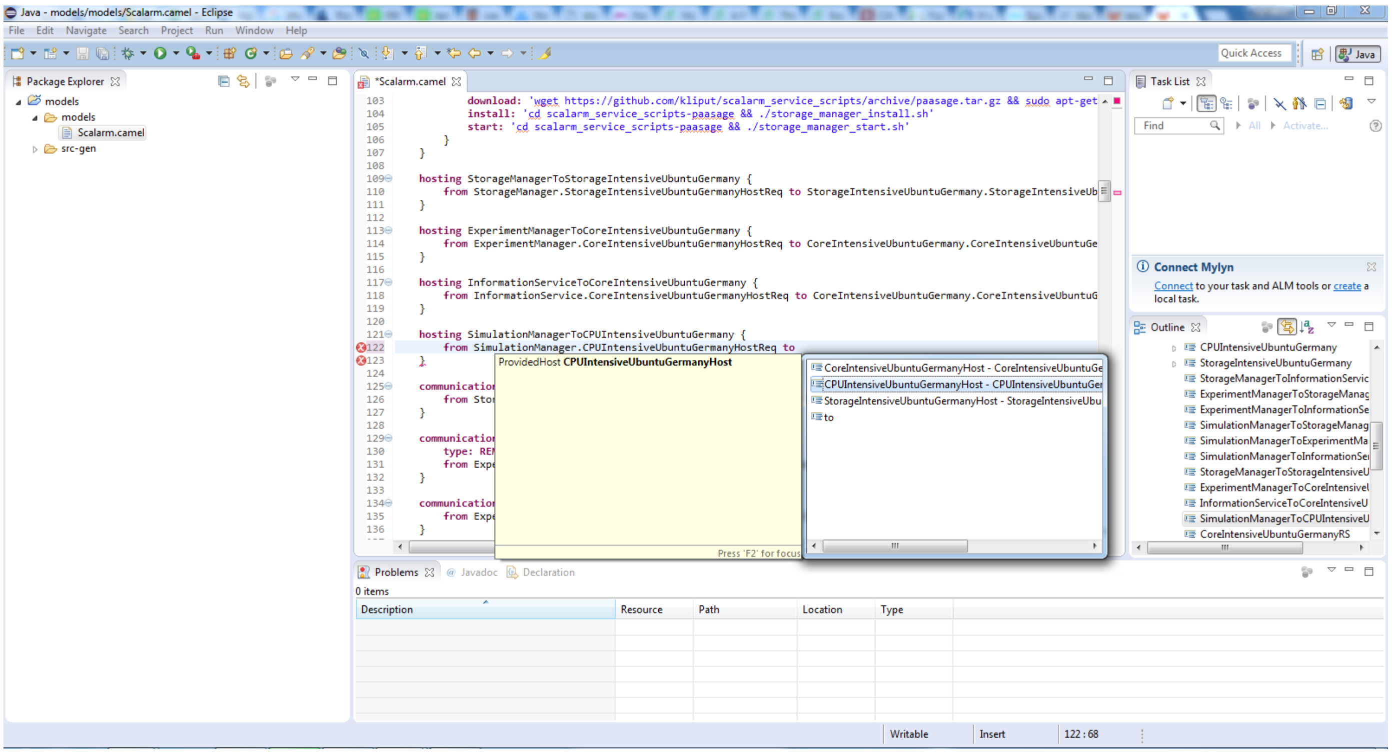
The Models Repository is part of the Upperware but is being presented in this section to emphasise its connection with the modelling part of the Melodic platform. The Models Repository stores the models manipulated by the Upperware components. The Melodic user-interfaces, in particular the editors, exploit this repository in order to store the models graphically generated by the users in order to enable their further processing by the core of the Melodic platform. The Models Repository mainly relies on an internal component called *CDOServer* which represents the server part in this repository enabling the storage and retrieval of models. The realisation of CDOServer is based on the Eclipse CDO technology[7] which enables the robust persistence of the manipulated models in the underlying storage which can take the form of a relational or a hibernate database. The technology also comes with additional features like the lazy loading of models and the support for transactionality. At the client side of this repository, the CDOClient lies offering an interface that enables users or programs acting on behalf of them to perform various actions over models, including in-memory loading as well as storing in the Models Repository, in cooperation with the CDOServer. The CDOClient is currently exploited by various components in the Upperware which require interaction with the models repository.

2.3 Melodic Interfaces to the End-Users

The users of the Melodic interact with the Melodic platform in two ways. First, application developers need to model their applications and data via an appropriate interface that enables valid CAMEL formal specifications utilisable by the Upperware components for reasoning application deployments. Second, the CAMEL language may itself need extensions based on the requirements of the Melodic adopters, and a formal way to enable defining extensions is required.

For modelling applications and data, two CAMEL editors are developed, a textual CAMEL editor and a web-based CAMEL editor. The textual CAMEL editor is based on Eclipse IDE and allows Create-Read-Update-Delete (CRUD) operations over the main CAMEL elements for describing, among others, the decomposition of the Cloud application into its components and for defining placement and scalability requirements that follow the required service level objectives (SLOs). Moreover, data sets are also modelled via the same editor. A screenshot of the textual editor is given in Figure 6.

[7] https://wiki.eclipse.org/CDO


Java - models/models/Scalarm.camel - Eclipse
File Edit Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
models
models
Scalarm.camel
src-gen
*Scalarm.camel
Task List
Find
All
Activate...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
CPUIntensiveUbuntuGermany
StorageIntensiveUbuntuGermany
StorageManagerToInformationServic
ExperimentManagerToStorageManag
ExperimentManagerToInformationSe
SimulationManagerToStorageManag
SimulationManagerToExperimentMa
SimulationManagerToInformationSe
StorageManagerToStorageIntensiveU
ExperimentManagerToCoreIntensivel
InformationServiceToCoreIntensiveU
SimulationManagerToCPUIntensiveU
CoreIntensiveUbuntuGermanyRS
Problems
Javadoc
Declaration
Writable
Insert
122 : 68

```
            download: 'wget https://github.com/kliput/scalarm_service_scripts/archive/paasage.tar.gz && sudo apt-get
            install: 'cd scalarm_service_scripts-paasage && ./storage_manager_install.sh'
            start: 'cd scalarm_service_scripts-paasage && ./storage_manager_start.sh'
        }
    }

    hosting StorageManagerToStorageIntensiveUbuntuGermany {
        from StorageManager.StorageIntensiveUbuntuGermanyHostReq to StorageIntensiveUbuntuGermany.StorageIntensiveUb
    }

    hosting ExperimentManagerToCoreIntensiveUbuntuGermany {
        from ExperimentManager.CoreIntensiveUbuntuGermanyHostReq to CoreIntensiveUbuntuGermany.CoreIntensiveUbuntuGe
    }

    hosting InformationServiceToCoreIntensiveUbuntuGermany {
        from InformationService.CoreIntensiveUbuntuGermanyHostReq to CoreIntensiveUbuntuGermany.CoreIntensiveUbuntuG
    }

    hosting SimulationManagerToCPUIntensiveUbuntuGermany {
        from SimulationManager.CPUIntensiveUbuntuGermanyHostReq to
    }

    communicatior
        from Stor
    }

    communicatior
        type: REM
        from Expe
    }

    communicatior
        from Expe
    }
```

```
ProvidedHost CPUIntensiveUbuntuGermanyHost
Press 'F2' for focus
```

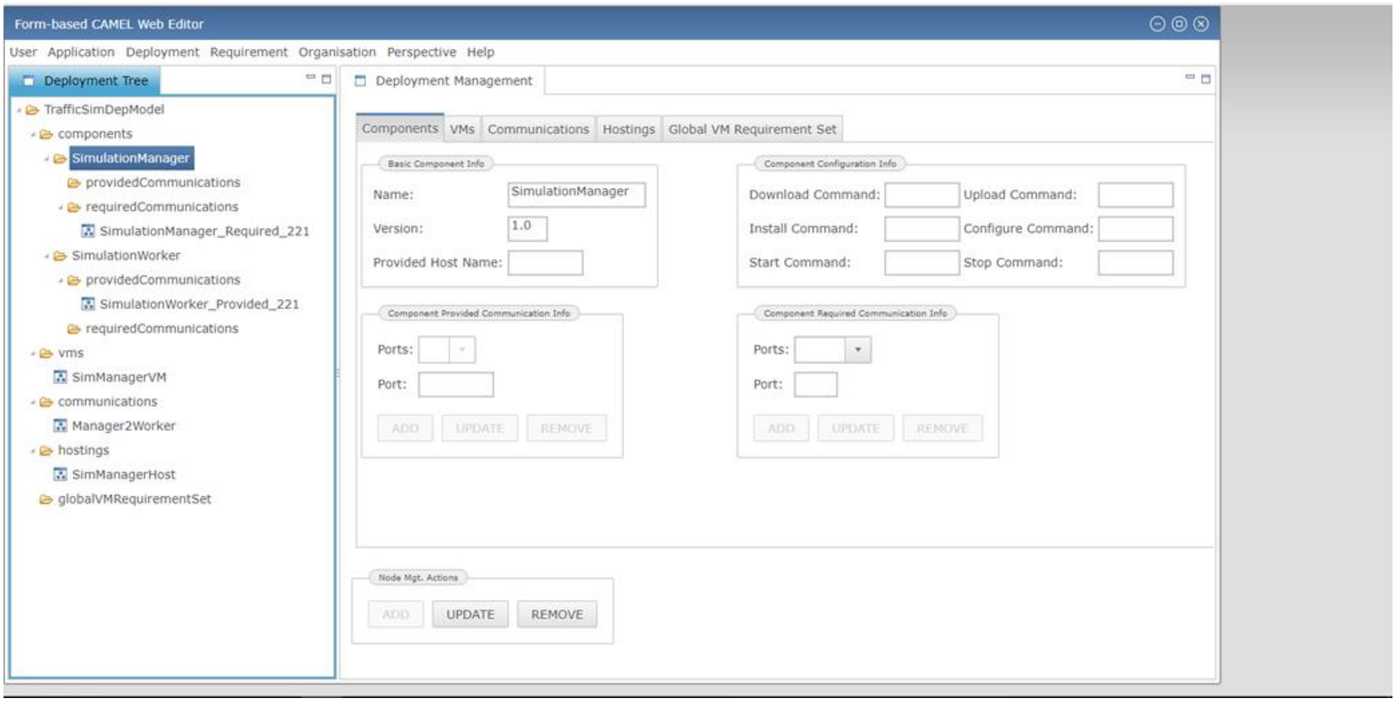
```
CoreIntensiveUbuntuGermanyHost - CoreIntensiveUbuntuGe
CPUIntensiveUbuntuGermanyHost - CPUIntensiveUbuntuGer
StorageIntensiveUbuntuGermanyHost - StorageIntensiveUbu
to
```

0 items

Description	Resource	Path	Location	Type

Figure 6: A screenshot of the textual CAMEL editor

The web-based CAMEL editor, as shown in Figure 7, provides a user-friendly way to edit CAMEL models through a form-based web interface.


Form-based CAMEL Web Editor
User Application Deployment Requirement Organisation Perspective Help
Deployment Tree
Deployment Management
Components VMs Communications Hostings Global VM Requirement Set
Basic Component Info
Component Configuration Info
Download Command:
Upload Command:
Install Command:
Configure Command:
Start Command:
Stop Command:
Component Provided Communication Info
Component Required Communication Info
Ports:
Port:
ADD UPDATE REMOVE
Node Mgt. Actions
ADD UPDATE REMOVE

- TrafficSimDepModel
  - components
    - SimulationManager
      - providedCommunications
      - requiredCommunications
        - SimulationManager_Required_221
    - SimulationWorker
      - providedCommunications
        - SimulationWorker_Provided_221
      - requiredCommunications
  - vms
    - SimManagerVM
  - communications
    - Manager2Worker
  - hostings
    - SimManagerHost
  - globalVMRequirementSet

Name: SimulationManager
Version: 1.0
Provided Host Name:

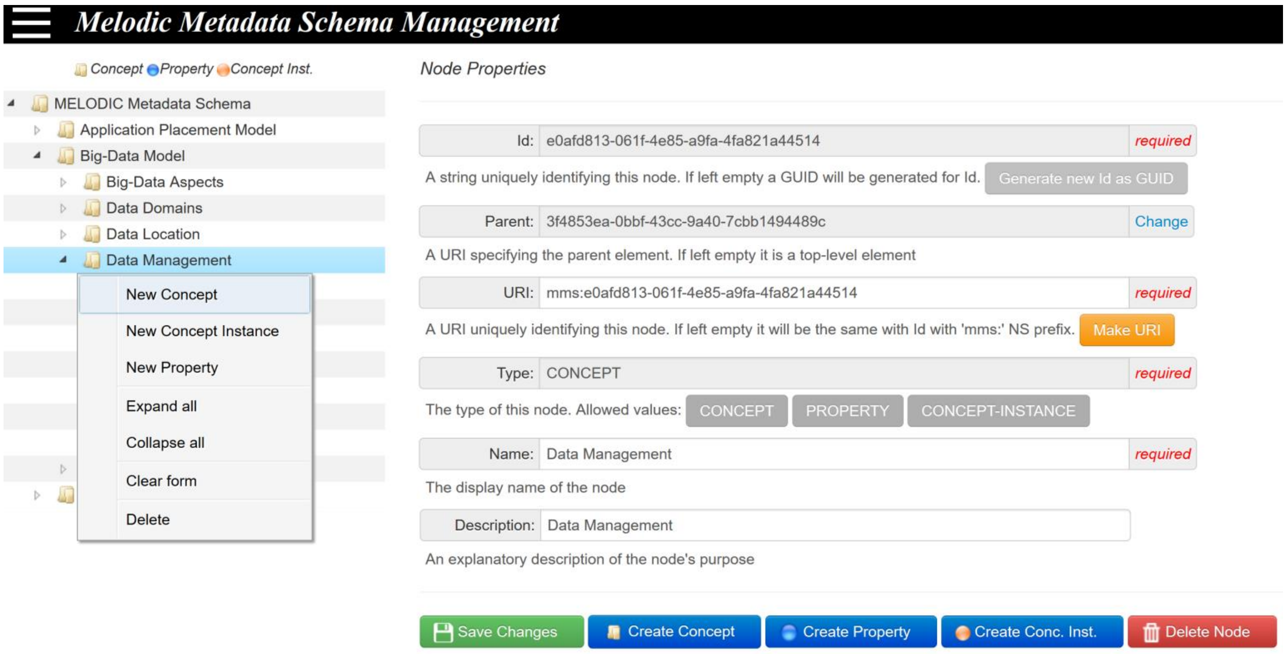
Figure 7: A screenshot of web-based CAMEL editor

Apart from the modelling based on the default CAMEL specification, we foresee that certain aspects of the CAMEL language may need extensions based on the requirements of the Melodic adopters. The extensible Melodic vocabulary has been described in terms of the Metadata Schema in deliverable *D2.4* [16]. In particular, we anticipate extensions on the Requirement, Metric, Scalability, Location, Provider and Security sub-models of CAMEL. Thus, certain aspects of the Metadata Schema has been seamlessly introduced in the CAMEL language and are available to the CAMEL editor. Any Melodic adopter is allowed to amend this Metadata Schema according to their organisational needs using the Metadata Schema editor. The editor will provide two functionalities:

- Enable CRUD operations over the Melodic vocabulary
- Provide the necessary functionalities for receiving Cloud application developers' or DevOps' preferences over a number of qualitative criteria based on the Metadata Schema.

Figure 8 shows a screenshot of the web-based Metadata schema editor.

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



Melodic Metadata Schema Management

Node Properties

Id:	e0afd813-061f-4e85-a9fa-4fa821a44514	required
Parent:	3f4853ea-0bbf-43cc-9a40-7cbb1494489c	
URI:	mms:e0afd813-061f-4e85-a9fa-4fa821a44514	required
Type:	CONCEPT	required
Name:	Data Management	required
Description:	Data Management	

A string uniquely identifying this node. If left empty a GUID will be generated for Id.

A URI specifying the parent element. If left empty it is a top-level element

A URI uniquely identifying this node. If left empty it will be the same with Id with 'mms:' NS prefix.

The type of this node. Allowed values: CONCEPT PROPERTY CONCEPT-INSTANCE

The display name of the node

An explanatory description of the node's purpose

Figure 8: A screenshot of the web-based Melodic metadata schema editor

3 Upperware

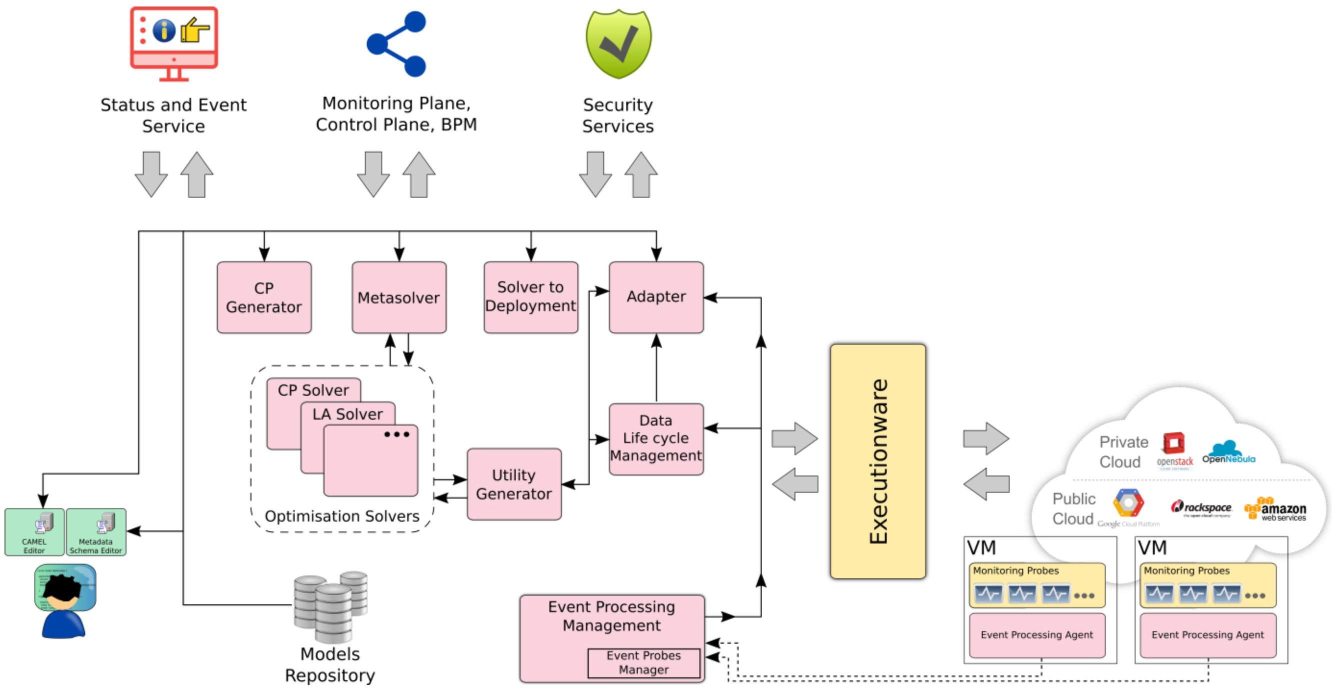


Figure 9: Overview of the Upperware Components

In this chapter, we focus on the details of the Upperware component group by providing a conceptual architecture and by describing its interactions with the other Melodic components. The Upperware comprises a number of software components that encapsulate all the necessary functionality for making timely decisions on appropriate Cross-Cloud data placements and application deployments. Basic aspects of this part of the Melodic architecture involve the appropriate components for finding optimal Cross-Cloud resources allocation and application placement in each situation, and coordinating the respective deployment. The output of Upperware will constitute the critical input for the Executionware component group (chapter 4), by propagating the best possible (re)configuration solution (translated into a number of deployment/configuration actions) based on the current status of the deployment topology in order to enact it.

Upperware also includes the appropriate user interfaces (as seen in Figure 9) for extending the critical aspects of the CAMEL language based on the extensible Melodic vocabulary (i.e., Metadata Schema) as discussed in chapter 2. Moreover, the Models Repository also constitutes a part of the Upperware, as described in section 2.2.3.

3.1 Upperware Components

In the following, we describe each of the Upperware component in detail along with the implemented interfaces and interactions with other components (apart from the already analysed Editors and Models Repository).

3.1.1 CP Generator

The CP Generator component is responsible for generating constraint programming (CP) models that are processed and solved by Melodic's solvers. Constraint programming models are extensively used to search feasible solutions from within large sets of candidate solutions by modelling the search problem in terms of arbitrary constraints [17]. The CP Generator component reads the CAMEL application model and all needed CAMEL provider models and based on them it creates a CP model that expresses a constraint equation needed by solvers. The CP Generator also executes preliminary filtering of the virtual machine offerings to limit the solution space for the solvers based on the various types of requirements given by the user in CAMEL application model. Furthermore, the CP Generator sets up and creates a Utility Generator instance which will be used to evaluate the value of the utility function for a given problem. The input interface of CP Generator will be exposed on ESB as a REST API interface. The interface accepts the path to the model in CDO Server. As output the path to the generated CP Model in CDO Server is returned.

3.1.2 Utility Generator

As mentioned previously, Melodic is the application owner's automated DevOps tool that aims to optimise the application deployment according to the application owner's goals and requirements. The concept of *utility* has since long been used in economic theory to capture someone's preferences and guiding decisions [18]. In autonomic computing it has been extensively used to express the *goodness* of a particular application configuration as seen and perceived by the application owner [19]. The Utility Generator is therefore an object acting on behalf of the application owner in the system, and it can use *an arbitrary method* to assign a *utility value* to a given deployment configuration proposed by the solver.

Once the solver has a solution candidate configuration, i.e., a vector of values assigned to the *variables* of the CP model that satisfies all the CP model constraints, it publishes this configuration to the Utility Generator. The Data Life-cycle Management System (DLMS),

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

discussed in detail in section 3.2, and the Adapter also see this configuration. The former assigns a penalty for adopting this configuration based on the effect this configuration will have on data location and use. The Adapter will assign its penalty value based on the deviation from the currently running application to the new configuration proposed using the models at runtime approach of dynamic software product lines [20]. The reconfiguration penalties from the DLMS and the Adapter will then be used by the Utility Generator to assign a *utility value* in the closed interval [-1, 1] to the candidate configuration. If the *utility value* is positive it means that the candidate configuration is *better* seen from the application owner's perspective than the currently running baseline configuration; conversely, if it is negative, it is then worse than the baseline configuration.

It should be noted that the solver generates a sequence of candidate configurations, and if a certain configuration's *utility value* is better than the currently running baseline configuration, it will be compared with the candidate configuration that has currently received the highest *utility value* compared with the currently running baseline configuration. If this comparison is positive for the new candidate configuration it is taken to be the *best* configuration seen until now, and retained as a deployment candidate configuration. This solution is then forwarded to the Solver-to-Deployment component and further to the Adapter, which may decide to reconfigure the running baseline configuration to this new deployment candidate, or reject the new deployment candidate for some reasons. The solver could then store the deployment candidate configuration and start its search for a better configuration based on this solution. The *utility value* assigned to the deployment candidate configuration will in this case be maintained if the Adapter rejects it because the running baseline configuration used for the utility comparison will then stay as before, or the *utility value* of the deployment candidate configuration is set to zero if the Adapter accepts this configuration so that it becomes the new baseline for comparing other future candidate configurations.

There are several approaches the Utility Generator may take to assign a *utility value* to a given candidate configuration. The application owner could, in theory, provide a mathematical function, often called **Utility Function**, taking a configuration vector as argument and returning a *utility value*. However, experience from several relevant major research projects clearly shows that it is a very difficult exercise even for advanced DevOps to formulate a good and usable utility function, and alternative configuration evaluation methods may prove equally effective [21]. For instance, given that the application owner's goals and preferences are often imprecise, **fuzzy logic** seems as a good candidate for comparing the configurations [22]. As the fuzzy base functions can easily be parameterised, it may be easier for the user to specify and tune a configuration comparison to match approximately the perceived goodness of a particular configuration. Fuzzy rules have previously been applied for autonomic computing [23], but to our knowledge fuzzy preference modelling as a way to capture the application owner's true *utility* has not been previously applied, and this approach will be further explored in Melodic.

The candidate configuration could be **simulated** in a Cloud platform simulator in order to evaluate its performance on key indicators, and then assess the goodness of the candidate configuration relative to the running baseline configuration. CloudSim[8] is one of the best-known Cloud simulators, with multiple variants and forks [24]. However, conducting an extensive simulation for every candidate configuration proposed by the solver will make the decision process prohibitively slow, and therefore this option remains an alternative to explore only for specific applications beyond the scope of Melodic. Alternatively, a way to assess a candidate configuration is of course to **deploy** the configuration regardless of adaptation cost, and then actually measure if it behaves better or worse than the baseline configuration it has replaced. This is probably as slow as conducting a full simulation of the deployment, and more intrusive to the running application; and it will introduce additional deployment cost. One could, of course, solicit the application owner's opinion about a given candidate configuration, and based on the iterative **user feedback** obtained learn the application owner's preferences and goals. This approach could certainly be used during the CAMEL model tuning phase to ensure that the stated requirements and constraints result in a desired deployment, and therefore serve as a starting point for the autonomic Melodic deployment. It may not be easy for the application owner to judge the utility of a configuration, although this approach remains a possibility to explore with the use case partners of Melodic.

3.1.3 Metasolver

The Metasolver constitutes the responsible software component that orchestrates the operation of the Solvers and ranks their outputs based on the application developer's or DevOps' needs. As explained below, the Melodic platform may accommodate any number of Solvers. These Solvers may use different methods and algorithms, such as constraint and linear programming or reinforcement learning [25] in order to reason on the best possible use of Cross-Cloud resources for placing data and applications. The role of Metasolver includes the following responsibilities:

1. Selection and invocation of the appropriate Solver(s) according to:
    a. the (non)-linearity of the constraint problem at hand
    b. time available for finding a (re)configuration solution

[8] http://www.cloudbus.org/cloudsim/

2. Collection of Solver(s) solution(s) according to the predefined time thresholds set by the DevOp
3. Comparison of available local optimum solutions based on the preferences defined by the DevOps or the application developer over a number of qualitative and quantitative criteria (e.g. amount of security services supported)

The main inputs for the Metasolver are the description of the Cloud application in CAMEL (e.g. scalability rules described) along with the CP model defined by the CP Generator. The output of the Metasolver is one solution produced based on one of the following two alternatives.

1. The optimum solution calculated by a certain selected solver (i.e., solution propagation) in case the constraint problem's size (i.e., both in terms of the number of variables used and the multitude of the possible solutions) is small enough to allow the calculation of the optimum solution faster than the time threshold set (I.e. the solver will have the time to find the optimum solution)
2. The identification of the best solution based on the weights on certain predefined qualitative and quantitative criteria (e.g. provider's reputation, resource's ecological footprint) in case the time threshold set was not enough for any of the available solvers to derive the optimum solution (e.g., on adaptation scenarios). The approach for selecting the best solution will follow an appropriate multi-criteria decision making technique (e.g. Analytic Hierarchy Process)

In either case the Metasolver's output is stored on the Models Repository (realised through CDO server) and a reference to it is relayed to the Solver-to-deployment component for constructing the CAMEL deployment model which will drive the initial placement or reconfiguration implementation. Based on the selected solution to be implemented the appropriate information is sent to the Utility Generator for consideration during the future utility function evaluations.

In case where a new adaptation is triggered, due to a scalability rule that was fired (e.g., >80% CPU usage THEN Scale-out) or a service level objective (SLO) violation (e.g., number of users >1000) the Metasolver invokes one or more solvers, requesting the calculation of a new optimum solution. It is important to note that in such scaling-out situation, two additional instances of a VM offering (e.g., m1.medium) might be automatically created but then the solver may reason about the need to change into just one VM with more VCPUs (e.g., m1.large) because it is the optimum option from the cost point of view. As such, to appropriately address such situations, it is advocated that the time available for finding the best reconfiguration should be much more limited with respect to that of the initial placement.

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

3.1.4 CP Solver

The CP Solver is responsible for solving a certain deployment reasoning/optimisation problem that is encoded in the form of a CP model. Such a deployment reasoning problem is expressed in the form of a constraint model (along with the utility function handled by the Utility Generator) which encompasses the following information.

- A set of constraints mapping to the SLOs defined by the user in the same CAMEL model
- A set of variables which denote the number of instances that an application component should have over a certain VM offering that satisfies the quantitative hardware requirements posed by the user for that component
- A set of constraints denoting further baseline restrictions over the variables; for instance, that the number of instances of an application component should not overpass the horizontal scalability requirement posed by the user in the same CAMEL model

Before solving the CP model, the CP Solver transforms it to the form expected by the CP Engine exploited internally by it which is the Choco solver engine[9]. Once the end result is produced, i.e., the solution, this result is then written back to the CP model processed.

The major features of this component include the following:

- Ability to also handle non-linear constraints
- Ability to incorporate in the constraints arbitrary, user-defined functions. Such functions could, for instance, correlate different variables together or they could enable the derivation of the non-functional capabilities at a higher-level (e.g., component level) from those exhibited at a lower level (e.g., the infrastructure one). For example, the execution time of a certain application component could be expressed in the form of a linear function which encompasses the characteristics of the VM on which that component is deployed along with the current workload.
- Ability to solve problems containing various types of variables, including integer-, and real-based. Integer-based variables are those mapping to the main variables of the CP model which express the number of instances of a component per matching VM offering. A real-based variable could express a certain non-functional parameter for an application component or the user application as a whole which might be involved in the constraints of the CP model mapping to the SLOs posed in the user CAMEL application model as well as on the main optimisation objectives of the CP model. As an example, the memory size for the whole application could be expressed as a variable which is derived from the

[9] http://www.choco-solver.org/

memory size of all of its components which in turn are derived via the weighted sum of the number of instances that each component has for a certain VM offering multiplied by the memory size of that VM offering. Then, a user might have provided a SLO which denotes that the application memory size should be greater than, e.g., 256 GB, which could then be expressed as a constraint in the CP model that involves the aforementioned variable.

- Ability to set a time threshold for the solving process, especially in case that the CP model obtained is complex and thus may take significant time to be solved
- Ability to concretise some parts of the CP model and especially the mapping between application components and Cloud offerings via the exploitation of best deployment knowledge that can be obtained by enforcing the execution of respective rules incorporated in a Knowledge Base that operates over the corresponding application history. Such partial concretisation of the CP model solution accelerates the solving of that model.

Currently, the CP Solver takes as input the CDO path to the CP model as well as a corresponding timestamp which denotes a certain part in the CP model where some constant values, mapping to measurements to be assigned to metric variables, can be found. Such values vary with respect to each solving of the CP model within the same deployment session of a user application so the timestamp enables the CP Solver to obtain the right variation. The CP solver returns as output the result of the solving in terms of whether a solution was found or the CP model was infeasible. It also writes the solution back to the CP model within the dedicated path in the CDO model repository where this model has been stored.

The solver exposes a REST interface with two methods that can be executed by issuing POST requests. These methods are more or less similar, as they map to the same core logic of solving a CP model and take as input similar parameters with the exception that the first obtains a path to the CDO model repository in order to read the CP model while the second a path in the local file system.

3.1.5 LA Solver

The Learning Automata (LA) solver is based on the realisation that the constraint mapping problem to solve is inherently stochastic [26]: Even if an application component is mapped to a virtual machine, that virtual machine is again mapped by the Cloud infrastructure provider to a physical server hosting multiple virtual machines such that the component performance can vary even in the same time period. This is the well-known issue of performance variation due to resource contention over the underlying hardware. Another example is when the application

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

utility is based on the average response time experienced by the application users where this measured response time is an implicit function of the unpredictable number of users and the amount of virtual resources provided to the application, and per the first example the amount of physical resources these virtual resources can access. The consequence of such examples is that both the application utility, which is the application owner's "objective function" to be maximised, and the constraints set for this optimisation can be stochastic and context dependent, in general.

This situation cannot be tackled by most solvers that will have to restart the search for a solution every time a measurement or context parameter changes value, and then must find a solution to the problem before the next change happens. Learning Automata are involved in a *game* with a stochastic *environment* where the environment has a certain unknown probability for rewarding a choice made by an automaton [27]. The goal of the automaton is to identify the "action" that maximises the reward it receives through a sequence of interactions with the environment. This theoretical framework fits well with the situation faced in Melodic, and a set of learning automata is used, one for each discrete variable of the CP problem like location, choice of machine type, and the number of cores, where the possible *actions* for each automaton correspond to the domain of the discrete variable it tries to optimise. The consequence is that the automaton gradually becomes more confident on its choice of action with every feedback it gets from the environment.

The LA solver therefore needs to run in parallel with the running application. Each automaton can be started with action probabilities selected from historical knowledge about the goodness of a choice, and then continuously propose assignments for the variable it tries to optimise aiming to improve the expected application utility. If no initial action probabilities are given, it will assume that all values in the variable's domain will be equally good initially with the same selection probability until the game with the environment makes it possible to identify out the better values. If the CP problem has *continuous* variables, these will be solved using the BOBYQA algorithm [28] with the discrete variables fixed by the set of learning automata. An assignment of all variables of the CP problem corresponds to a possible deployment *configuration*.

A configuration satisfying all the constraints of the CP-problem will be proposed to the *Utility Generator*, which will assign a goodness to this configuration relative to the currently running configuration. If the utility of the new configuration is higher than the currently running configuration, the new configuration will be proposed as a deployment candidate, and the stochastic search for an even better candidate will continue.

Hence, the CP-Generator will create a source file specifying the variables and their domains, the metrics to provide context dependent values, and the constraints involving variables and metric references. These definitions will then be compiled and linked with the LA solver code, and when the executable starts it will load persisted action set probabilities for the given variables from the Models Repository and start the search. Once it has a deployment configuration candidate, it will be proposed to the Utility Generator and the iterative game starts. The solver will subscribe to a binary shut-down flag, and once this flag is set the current value of the action probabilities for each discrete variable will be written back to the Models Repository, and the solver executable terminates.

3.1.6 Solver-To-Deployment

The Solver to deployment is a component which has the responsibility to apply a certain solution derived from the solvers to the CAMEL model of the application to be deployed. The main goal is to move from a provider-agnostic deployment model of the application at the type level to a provider-specific model of that application at the instance level. The CAMEL accommodates for the specification of both types of models. As such, the role of the Solver to deployment component is to check how many instances of application components and VMs are needed to be created as well as to make the respective connections between them (i.e., communication & hosting instances) in order to generate the provider-specific deployment model required. As a certain major consideration, in the context of this generation, this component also needs to map the instances of VMs modelled to the exact offerings from the respective model of the provider that they match.

In order to achieve its goal, the Solver to deployment needs to obtain as input the paths in the Models Repository of two models: (a) the CP model in which the solution has been imprinted; (b) the CAMEL model of the user application. Furthermore, optionally this component might obtain a timestamp in order to denote the part of the CP model where the latest solution resides. If this timestamp is not given, the Solver to deployment component attempts to consult the corresponding CP model part with the latest timestamp. As a result, the CAMEL model is updated within the Models Repository and a notification is sent about the success or not in updating this model.

3.1.7 Adapter

The Adapter is the component responsible for analysing and validating a new CAMEL deployment model and defining a number of reconfiguration action tasks to be executed in a specific order, i.e. a reconfiguration action graph. The validation might involve cost and time constraints aspects. Specifically, the adapter instructs the Executionware to execute these action tasks in order to implement the optimised deployment configuration for the current execution context found by the solver.

To guide the solver's search for the better deployment for the current context, the adapter assigns a *reconfiguration penalty* for the migration from the current deployment to a new deployment candidate proposed by the solver by considering the complexity in time and component placements of the difference between the current deployment and the candidate. This penalty will be used by the Utility Generator in order to compute the final utility of a proposed, new deployment; and thereby guide the solver to implicitly consider the reconfiguration in its decision on the next, better deployment configuration.

The main features of the adapter component are:

- To update the CAMEL model of the running application as soon as the actual deployment changes as a result of platform level scaling actions, or when the application execution context changes.
- To calculate the difference between the updated model of the currently running application and the new, optimised deployment proposed by the solver.
- To compute the sequence of reconfiguration tasks necessary to change the deployment from the current deployment to the proposed deployment.
- To assign a *reconfiguration penalty* based on the complexity of these reconfiguration tasks.
- To instruct Executionware to execute the reconfiguration tasks in the appropriate order as a series of REST API calls once the proposed deployment has been accepted, and monitor the outcome of these actions to ensure that the application has been properly re-configured by the Executionware.

The Adapter will expose its interface as a REST API on the ESB. The input to the Adapter is a reference to the proposed CAMEL model stored in the Models Repository by the Solver to Deployment component. The output is the result of the reconfiguration as returned by the Executionware.

3.1.8 Event Processing Management

Complex event processing has evolved into a dominant method for monitoring of reactive/adaptive applications [31]. In Melodic, the Event Processing Management, is the component responsible for the synchronisation and orchestration of Event Processing Agents, while it undertakes the setting up of complex event processing rules on each of these Agents.

We are considering Esper[10] engines for the implementation of these Event Processing Agents on each Virtual Machine commissioned for Melodic-enabled Cloud application deployment. Specifically, this component is responsible to instruct where to deploy new or additional Event Processing Agents that will undertake the task of processing hardware and application level monitoring events (coming from Monitoring Probes) in order to detect SLOs or scalability rules violations (i.e. complex event patterns). On a successful deployment, it also undertakes the configuration or enhancement of all the appropriate rules used in each Event Processing Agent. We note that the definition of these rules will be performed according to the needs expressed in CAMEL models as scalability rules and SLOs.

One of Event Processing Management mechanism's unique characteristics is that it materialises (through proper instructions to Executionware) and manages a distributed network of Event Processing Agents. This network of agents will be structured across three main hierarchy layers: the VM instance layer, the Cloud layer and the Global layer. The first one corresponds to the installation and configuration of Event Processing Agents on each VM instance (where application components are placed) in order to focus on the aggregation, filtering and propagation of application specific monitoring data and raw (hardware level) health status events. The second layer will involve the use of one such agent per Cloud for extracting higher level information on the placed data and Cloud application. This will allow for a valuable consolidated view of monitoring data per Cloud based on the aggregation and processing of the output of *local* Event Processing Agents that report from each VM. Consequently, the third layer (residing on the same resources used for the Adapter) will involve the aggregation of the "second level" Agents' output in order to allow for a global overview of the status of the whole topology used by the Melodic platform for monitoring and event generation (which leads to detecting the need to scale/adapt the application). Thus, the Event Processing Management component is responsible for setting up and maintaining a distributed Event Processing system bound to the DevOps' and/or application developers' requirements.

Due to the need for dynamic and adaptive deployments of various Virtual Machines, we are considering the use of a flexible type of messaging protocol to transfer the raw data coming from the Monitoring Probes to the Event Processing Agents (e.g., the use of Apache ActiveMQ[11] over Mule ESB). For this reason, the Event Processing Management component will use an interface which is based on the Publish/Subscribe model. The Event Processing Management component will mainly interact with the Adapter. Specifically, its input will involve the triggering from the Adapter in order to consider the rules, configuration and deployment of the Event Processing Agents based on the (new) CAMEL deployment model. Its output will firstly

10 http://www.espertech.com/products/esper.php
11 http://activemq.apache.org/

involve the submission of appropriate deployment and configuration instructions per each Event Processing Agent, to the Executionware. In addition, its output will involve the complex event patterns detection posted on the control plane for invoking placement adaptations. We note that among the responsibilities of the Event Processing Management mechanism is also the detection of the Monitoring Probes that should be also installed on each VM used by Melodic (based on the CAMEL model). This part of the work is handled by a dedicated sub-component, the Event Probes Manager as explained below.

3.1.9 Event Probes Manager

The Event Probes Manager is the component responsible for deciding and instructing (the Executionware) to deploy new monitoring probes and configuring their respective event topics used per metric and per node (i.e., VM). The objective is to install all the necessary sensing software in order to transmit application-level and hardware-level raw monitoring data that reveal the state of the application components and the health status of the Cloud resources used (e.g., CPU load, RAM usage, etc.). This mainly involves the Adapter, based on a devised action graph, to invoke the Event Probes Manager in order to configure and trigger the installation of monitoring software on selected nodes (VMs) through the Executionware. The raw monitoring data is collected and processed by Event Processing Agents (as explained in section 3.1.8 above). In case of reconfiguration, the Event Probes Manager will be capable of detecting only the delta of monitoring probes and instructing the installation of the missing ones while deregistering probes placed on decommissioned resources.

3.2 Data Lifecycle Management System (DLMS)

Data-intensive computing is characterised by challenges associated with complete data lifecycle comprising data collection, data storage, and data analysis of heterogeneous datasets [29]. The complexity of managing datasets also grows with their volume. In Cross-Cloud deployments, intelligent and pre-emptive data placement strategies are important for efficient data-intensive computing [30]. In addition, data placement in Cloud are also generally subjected to long-term storage selection, as migration may incur high costs, depending on the size, location, and other characteristics of the datasets. As a result, the initial Cloud selection for a given data source potentially affects the subsequent placement of applications requiring access to that data source, a phenomenon commonly referred to as *data gravity*. To efficiently manage data-sources, Melodic Upperware includes a dedicated component, DLMS, which enables holistic management of data lifecycle in Cross-Cloud environments.

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

The main functionality DLMS offers include:

- Management of data sources on behalf of the Melodic user
- Optimal data placement in the Cloud based on user-defined data placement requirements, constraints, and associated costs
- Keeping user-defined data requirements satisfied throughout the data lifetime
- Assignment of data transfer and access costs associated with data sources given an application topology and its data access requirements

All data sources available to the applications deployed through Melodic are modelled (See section 3.2, Data Modelling) and registered in DLMS. Once a data source has been registered in Melodic, it will be managed by DLMS throughout its life-cycle. This covers, when required, the selection of an appropriate location for the initial placement of the dataset on a Cloud based on the user-defined requirements and data storage costs in various Clouds, and subsequent migrations based on the application and data deployment solutions calculated by the Solvers. Note that both *external* and *internal* data sources are possible, where external data sources are referred to the ones *uncontrollable* by the Melodic platform and the DLMS will not be able to manage them.

From the monitoring plane, DLMS will subscribe to the data access metrics associated with the data processing by deployed application components. Making use of the historical information about the data access patterns of the application components, and the incurred monitory costs in the Cloud (data access and data storage costs), DLMS will create the business logic for assigning costs to the data sources, given the location of the application components accessing the data. As component data access patterns may be variable and only stabilise over time, DLMS will employ machine learning techniques to improve prediction accuracy over time.

In the Upperware workflow, for each solution proposed by a Solver, the Utility Generator will consult DLMS which will associate a cost given the particular Cross-Cloud application component topology and current location of the required data sources, as prescribed by the solution. For instance, if the proposed solution does not require any data migrations, DLMS will assign nominal costs to the solution. However, when a deployment solution requires moving very large datasets to another location, the costs assigned by DLMS will be very high affecting the subsequent utility of the solution. Moreover, DLMS will also ensure that the data placement requirements are not violated by providing an infinitely high cost when the respective user-defined data constraints are violated for a given solution. In addition to the cost assignment, when a reconfiguration to the existing deployment is required, DLMS will be consulted by the Adapter to trigger any data migrations required for the concerned datasets. The actual data migration though will require customised migration scripts provided by the user at the time of data registration in DLMS. DLMS interfaces will be exposed as REST APIs on ESB.

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

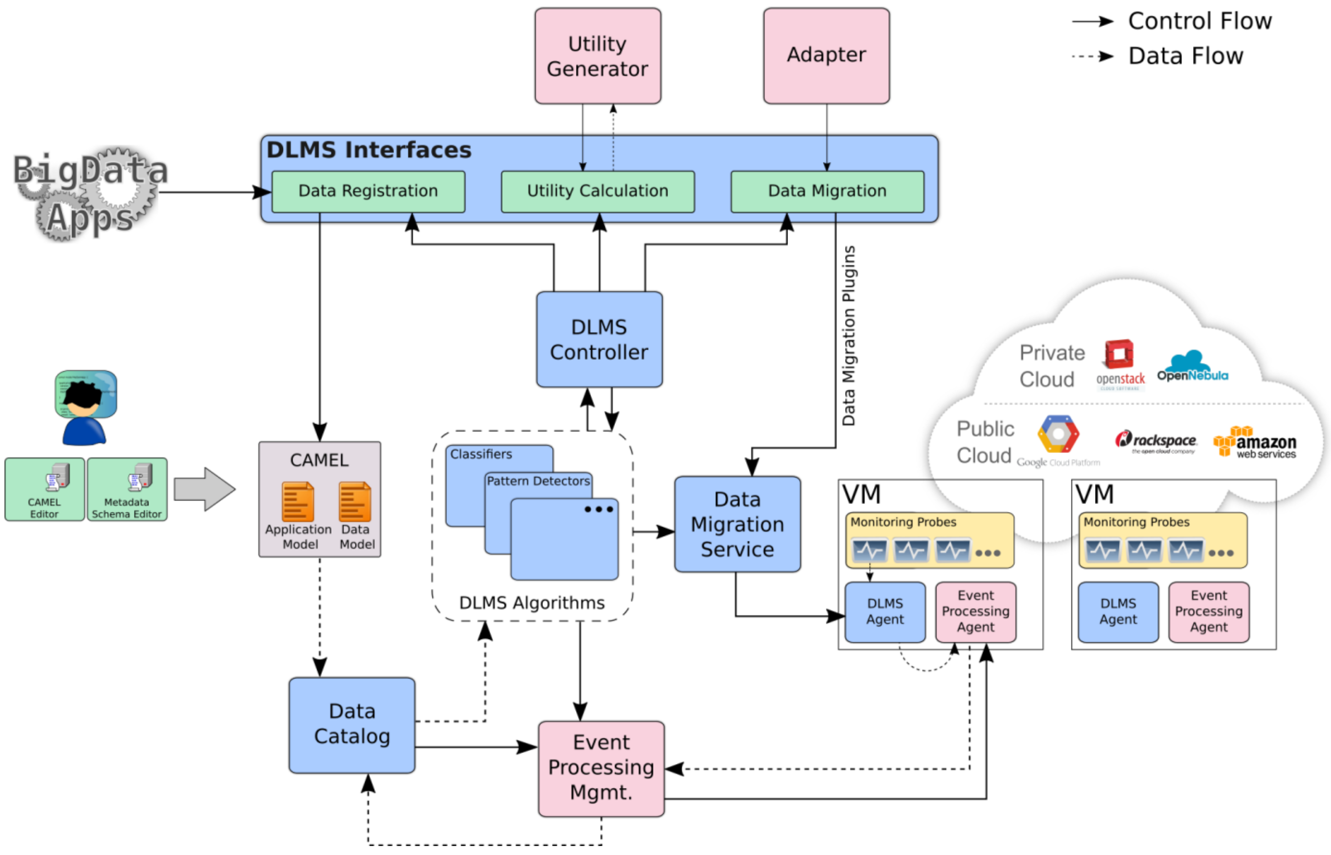


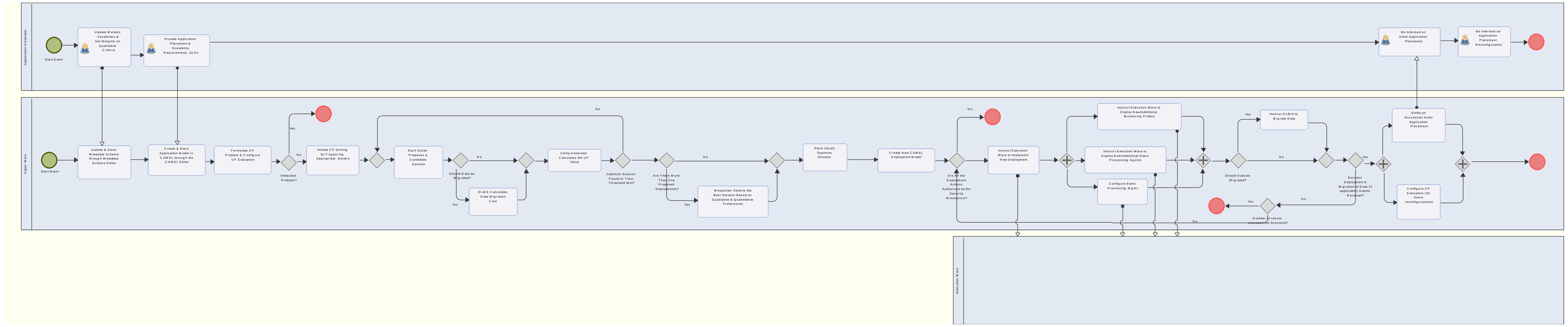
Figure 10: An overview of the data life-cycle management system

An overview of the internal architecture of DLMS is shown in Figure 10. At the heart of the DLMS is a DLMS controller that manages interaction between DLMS interfaces and the reasoning algorithms DLMS employs to assign costs to the proposed deployment solutions. DLMS interfaces include a data registration and access API that is used to register data sources, as well as interfaced by the big data applications to connect to the managed data sources. This is important as the data sources managed by the DLMS are subjected to the data migrations, if needed, according to the data life cycle management algorithms. As shown in the figure, the Utility Generator uses a *Utility Calculation* interface provided by the DLMS to query the data-related cost associated with each solution proposed by a Solver. In addition, DLMS also provides a *Data Migration* interface to the Adapter which is used to realise actual data migrations using a *Data Migration Service*. The data migration service uses *data migration plugins* provided by the user, which are executed through *DLMS Agents* that are deployed on each VM initiated by the Executionware. The DLMS agents, besides enabling data migrations, also install probes to record data access/transfers by the application components running on a VM. The monitoring data received by the DLMS agents is forwarded to the Event Processing Agents and is handled through Event Processing Management as discussed in section 3.1.8. A *Data Catalog* is maintained with both historical information about the data access patterns as well as updated application and data models, prescribed in CAMEL by the user. The DLMS algorithms include machine learning classifiers and pattern detectors to infer data access and migration costs and usage patterns according to the historical information stored in the data catalog, and the real-time monitoring data available through Event Processing Management.

3.3 Upperware Interfaces & Workflows

In this section we provide a bird's eye view of Upperware with the aim to provide an overview of its internal and external interfaces along with an explanation of the involved workflows that present how its objectives are attained. In particular, this presentation is based on the detailed descriptions provided in the previous sections concerning the individual software components of Upperware. It demonstrates how all these individual software components can interoperate for reaching timely decisions on the most suitable data and data-intensive application components placement over Cross-Clouds.

Figure 11 presents an outline of the workflow that involves the main tasks performed by Upperware components in order to reach an initial placement decision of both data and the Cloud application using the appropriate Cloud resources from single or multiple providers (the same figure is shown enlarged by Figure 12 to Figure 15). Specifically, the process begins with an opportunity for the application developer to enhance (through the Metadata Schema Editor) the already available Melodic vocabulary (which is presented in Melodic Deliverable D2.4 - Metadata Schema [16]) comprised of concepts that can be used for expressing placement and scalability requirements. Moreover, preferences over a number of selected qualitative criteria can be provided by the developer (Metadata Schema Editor) to be used in Metasolver's decisions on finding the best placement solution when only local optimum solutions are available (i.e., adaption scenarios) by the Solvers. In addition, the application developer with the use of a dedicated CAMEL editor, models her application along with its placement, scalability requirements and relevant service level objectives (SLOs). The sequence of the interactions involving the two editors are also depicted in the Upperware sequence diagram, while their provided and required interfaces are presented in the component diagram in Figure 21.



Application Developer
Start Event
Update Melodic Vocabulary & Set Weights on Qualitative Criteria
Provide Application Placement & Scalability Requirements, SLOs
Be Informed on Initial Application Placement
Be Informed on Application Placement Reconfiguration
Upper Ware
Start Event
Update & Store Metadata Schema through Metadata Schema Editor
Create & Store Application Model in CAMEL through the CAMEL Editor
Formulate CP Problem & Configure UF Evaluation
Infeasible Problem?
Yes
No
Initiate CP Solving by Triggering Appropriate Solvers
Each Solver Proposes a Candidate Solution
Should Data be Migrated?
No
Yes
DLMS Calculates Data Migration Cost
Utility Generator Calculates the UF Value
Optimum Solution Found or Time Threshold Met?
No
Are There More Than One Proposed Deployments?
No
Yes
Metasolver Selects the Best Solution Based on Qualitative & Quantitative Preferences
Store (local) Optimum Solution
Create New CAMEL Deployment Model
Are All the Deployment Actions Authorized by the Security Mechanism?
No
Instruct Execution Ware to Implement New Deployment
Instruct Execution Ware to Deploy New/Additional Monitoring Probes
Instruct Execution Ware to Deploy New/Additional Event Processing Agents
Configure Event Processing Mgmt.
Should Data be Migrated?
Yes
No
Instruct DLMS to Migrate Data
Success Deployment & Migration of Data (if applicable) Events Received?
Yes
No
Number of retries exceeded the threshold?
Yes
No
Notify on Successful Initial Application Placement
Configure UF Evaluation (for future reconfigurations)
Execution Ware

Figure 11: Upperware BPMN diagram (initial placement) – outline (enlarged by Figure 12 to Figure 15)

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

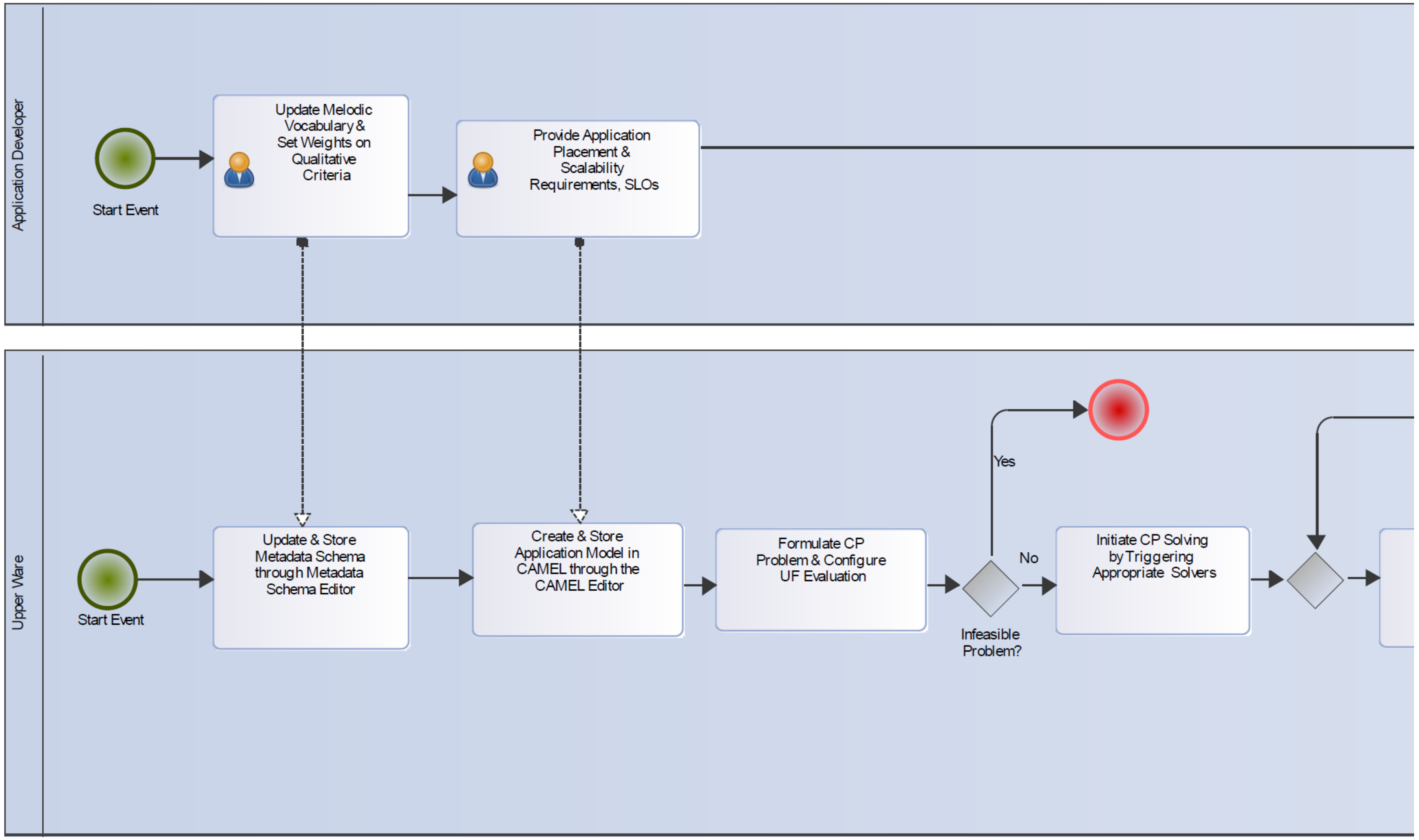

Application Developer
Start Event
Update Melodic Vocabulary & Set Weights on Qualitative Criteria
Provide Application Placement & Scalability Requirements, SLOs
Upper Ware
Start Event
Update & Store Metadata Schema through Metadata Schema Editor
Create & Store Application Model in CAMEL through the CAMEL Editor
Formulate CP Problem & Configure UF Evaluation
Yes
No
Infeasible Problem?
Initiate CP Solving by Triggering Appropriate Solvers

Figure 12: Upperware BPMN diagram (initial placement) – enlarged, part I of IV

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

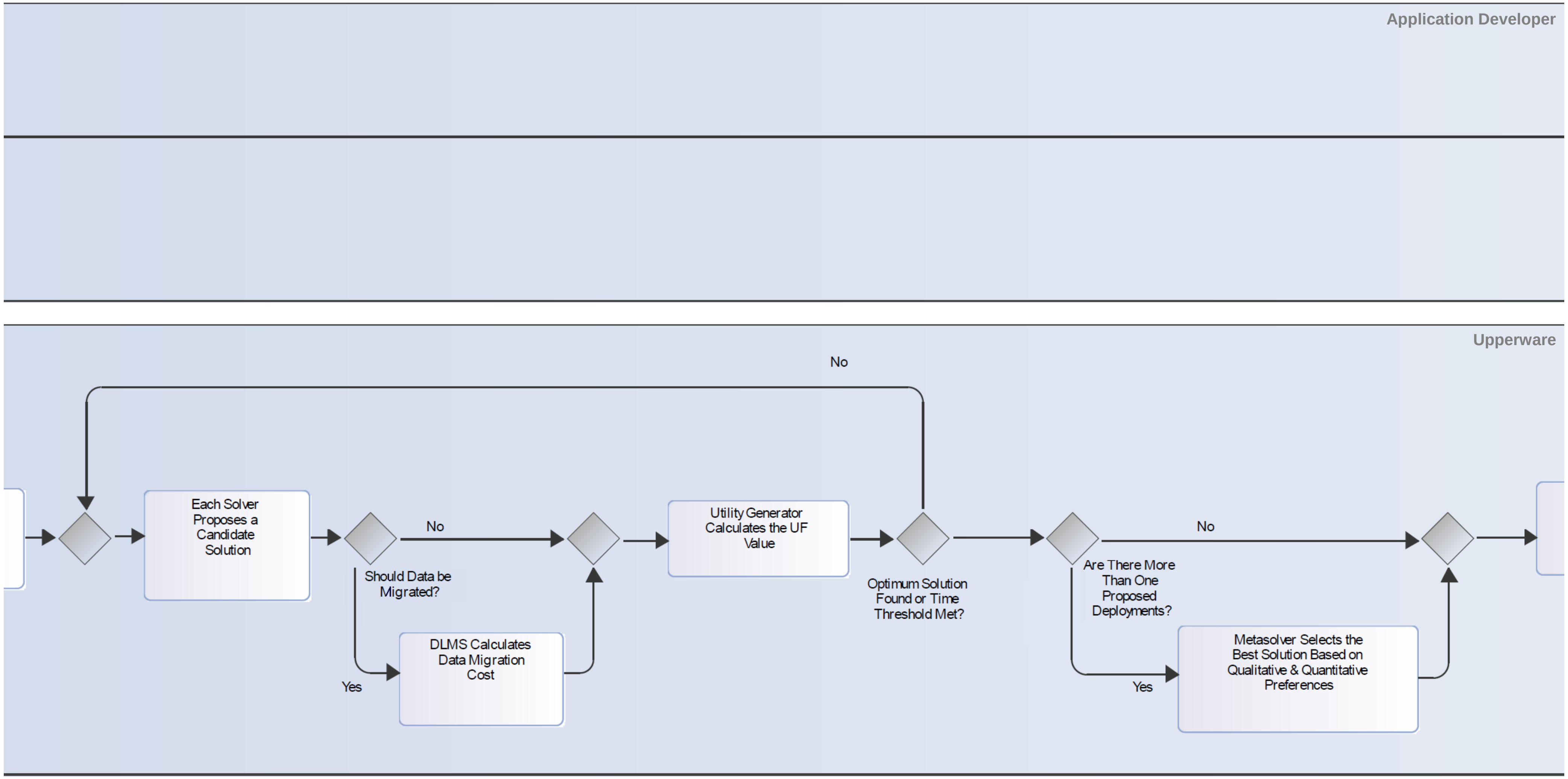

Application Developer
Upperware
No
Each Solver Proposes a Candidate Solution
Should Data be Migrated?
No
Yes
DLMS Calculates Data Migration Cost
Utility Generator Calculates the UF Value
Optimum Solution Found or Time Threshold Met?
Are There More Than One Proposed Deployments?
No
Yes
Metasolver Selects the Best Solution Based on Qualitative & Quantitative Preferences

Figure 13: Upperware BPMN diagram (initial placement) – enlarged, part II of IV

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

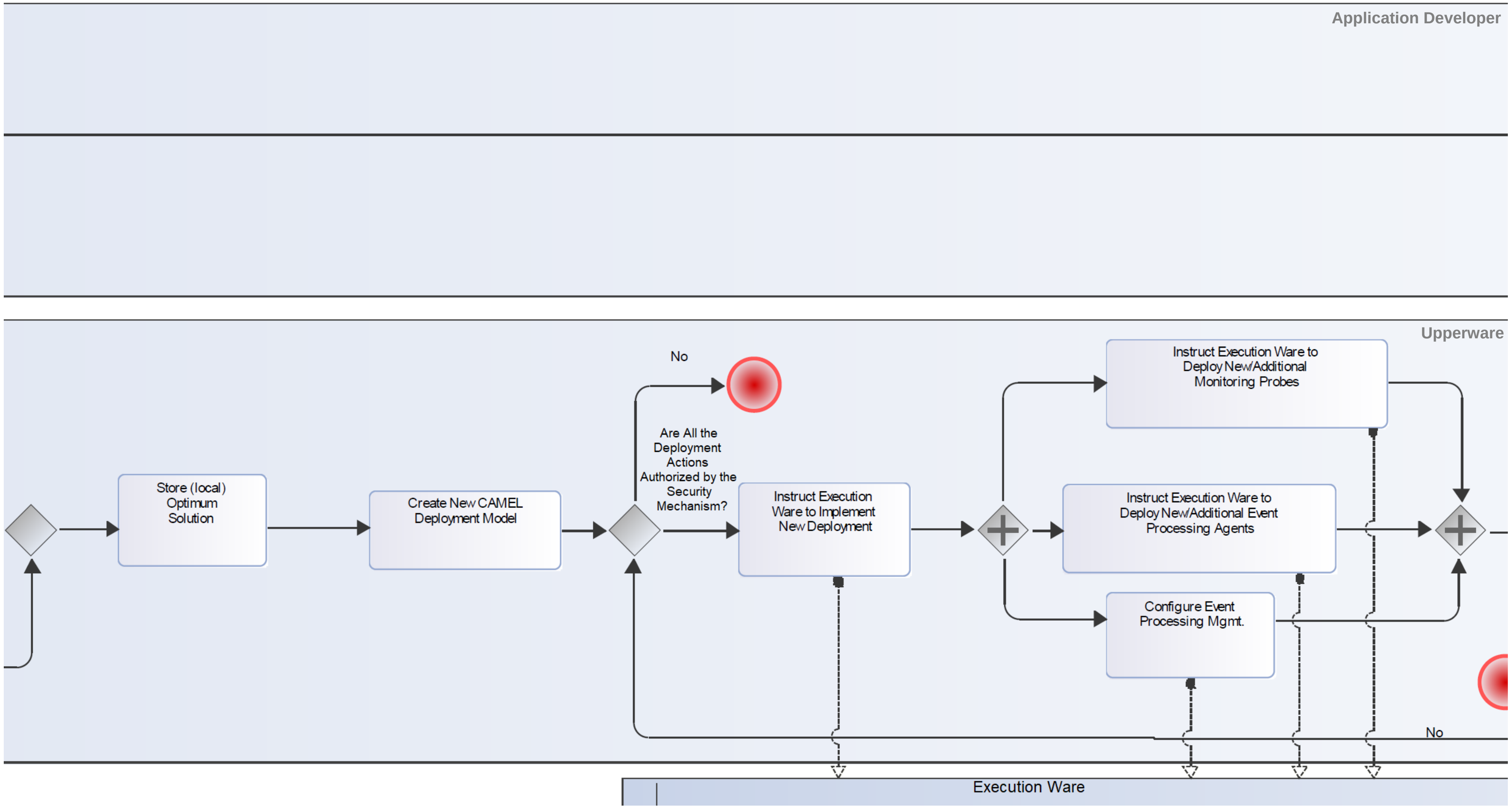

Application Developer
Upperware
No
Are All the Deployment Actions Authorized by the Security Mechanism?
Store (local) Optimum Solution
Create New CAMEL Deployment Model
Instruct Execution Ware to Implement New Deployment
Instruct Execution Ware to Deploy New/Additional Monitoring Probes
Instruct Execution Ware to Deploy New/Additional Event Processing Agents
Configure Event Processing Mgmt.
No
Execution Ware

Figure 14: Upperware BPMN diagram (initial placement) – enlarged, part III of IV

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

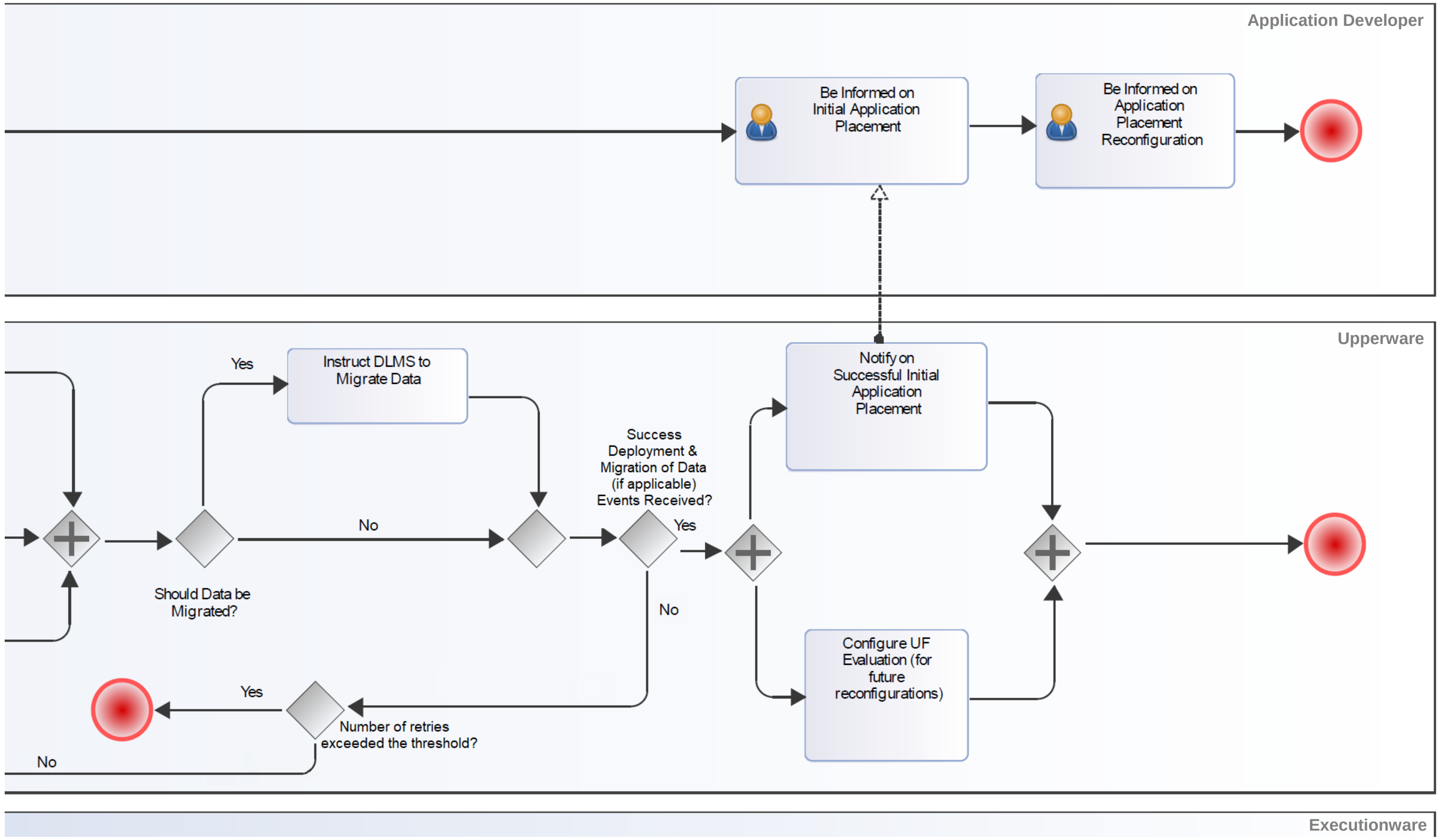
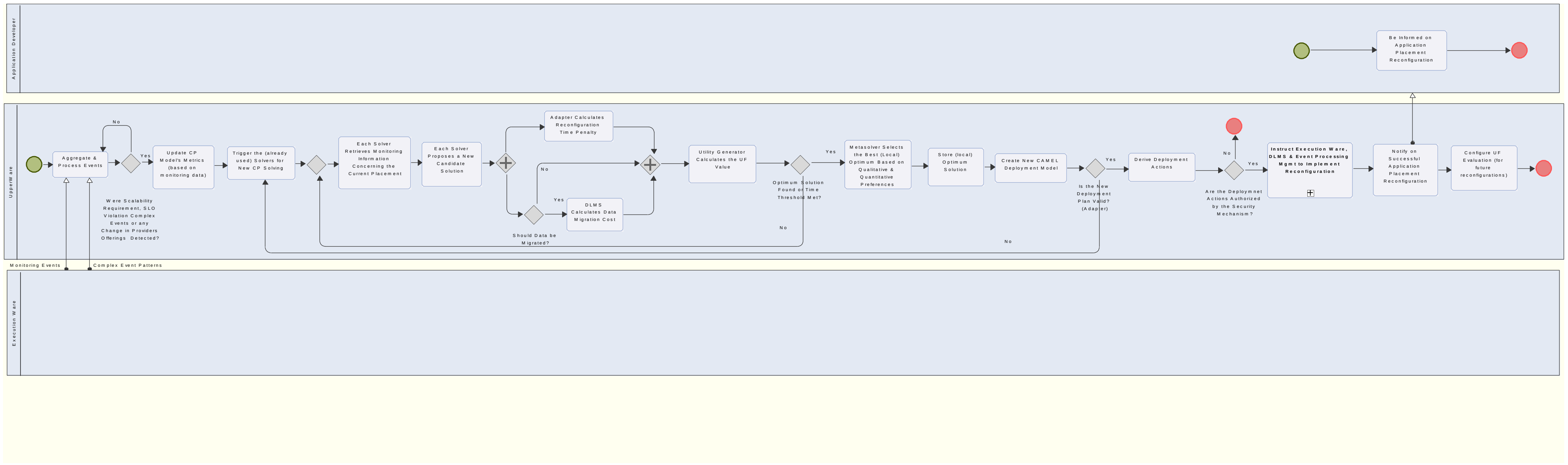

Application Developer
Be Informed on Initial Application Placement
Be Informed on Application Placement Reconfiguration
Upperware
Yes
Instruct DLMS to Migrate Data
No
Should Data be Migrated?
Success Deployment & Migration of Data (if applicable) Events Received?
Yes
No
Notify on Successful Initial Application Placement
Configure UF Evaluation (for future reconfigurations)
Yes
Number of retries exceeded the threshold?
No
Executionware

Figure 15: Upperware BPMN diagram (initial placement) – enlarged, part IV of IV

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

Once the initial placement problem has been formulated in CAMEL, the CP Generator creates the constraint programming (CP) problem and configures the Utility Generator for being able to evaluate the different candidate solutions that each solver may suggest. Based on the type of CP problem at hand (as it was discussed in section 3.1.3) the Metasolver triggers the appropriate Solver (e.g., CP Solver) while it starts any other solver (i.e., LA Solver) that is based on reinforcement learning and should continuously operate in the background. As shown in Figure 16 (enlarged by Figure 17 to Figure 20), every time the involved Solver(s) tries a new candidate configuration, the Utility Generator provides a utility value (using, e.g., Utility Function or Fuzzy Logic) that can be compared to the best configuration candidate already found. DLMS is invoked to provide an estimation of the data migration cost as described in section 3.2. In case that there are more than one proposed deployment solutions (i.e., time for calculating the optimum solution was not adequate, and more than one solvers were used), the Metasolver selects the best one based on the functionality described in section 3.1.3. The (local) optimum solution is stored in the Models Repository, while the Utility Generator is also informed to consider this solution in future UF evaluations that might be requested by the Solvers.

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



Application Developer
Be Informed on Application Placement Reconfiguration
UpperWare
Aggregate & Process Events
No
Yes
Were Scalability Requirement, SLO Violation Complex Events or any Change in Providers Offerings Detected?
Update CP Model's Metrics (based on monitoring data)
Trigger the (already used) Solvers for New CP Solving
Each Solver Retrieves Monitoring Information Concerning the Current Placement
Each Solver Proposes a New Candidate Solution
Adapter Calculates Reconfiguration Time Penalty
No
Yes
Should Data be Migrated?
DLMS Calculates Data Migration Cost
Utility Generator Calculates the UF Value
Optimum Solution Found or Time Threshold Met?
Yes
No
Metasolver Selects the Best (Local) Optimum Based on Qualitative & Quantitative Preferences
Store (local) Optimum Solution
Create New CAMEL Deployment Model
Is the New Deployment Plan Valid? (Adapter)
Yes
No
Derive Deployment Actions
Are the Deploymnet Actions Authorized by the Security Mechanism?
No
Yes
Instruct Execution Ware, DLMS & Event Processing Mgmt to Implement Reconfiguration
Notify on Successful Application Placement Reconfiguration
Configure UF Evaluation (for future reconfigurations)
Monitoring Events
Complex Event Patterns
Execution Ware

Figure 16: Upperware BPMN diagram (reconfiguration) – outline (enlarged by Figure 17 to Figure 20)

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

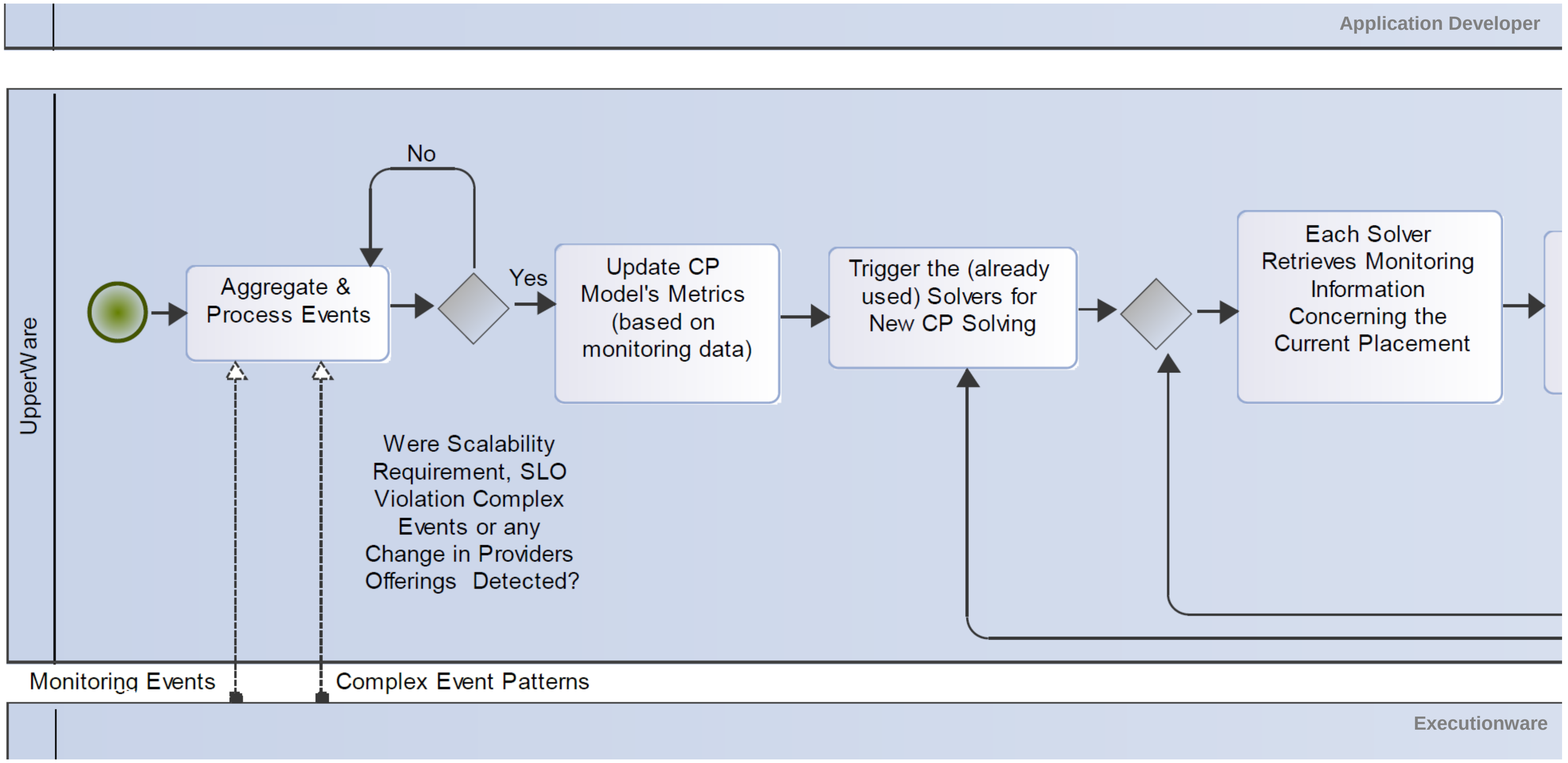

Application Developer
UpperWare
Aggregate & Process Events
No
Yes
Update CP Model's Metrics (based on monitoring data)
Trigger the (already used) Solvers for New CP Solving
Each Solver Retrieves Monitoring Information Concerning the Current Placement
Were Scalability Requirement, SLO Violation Complex Events or any Change in Providers Offerings Detected?
Monitoring Events
Complex Event Patterns
Executionware

Figure 17: Upperware BPMN diagram (reconfiguration) – enlarged, part I of IV

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

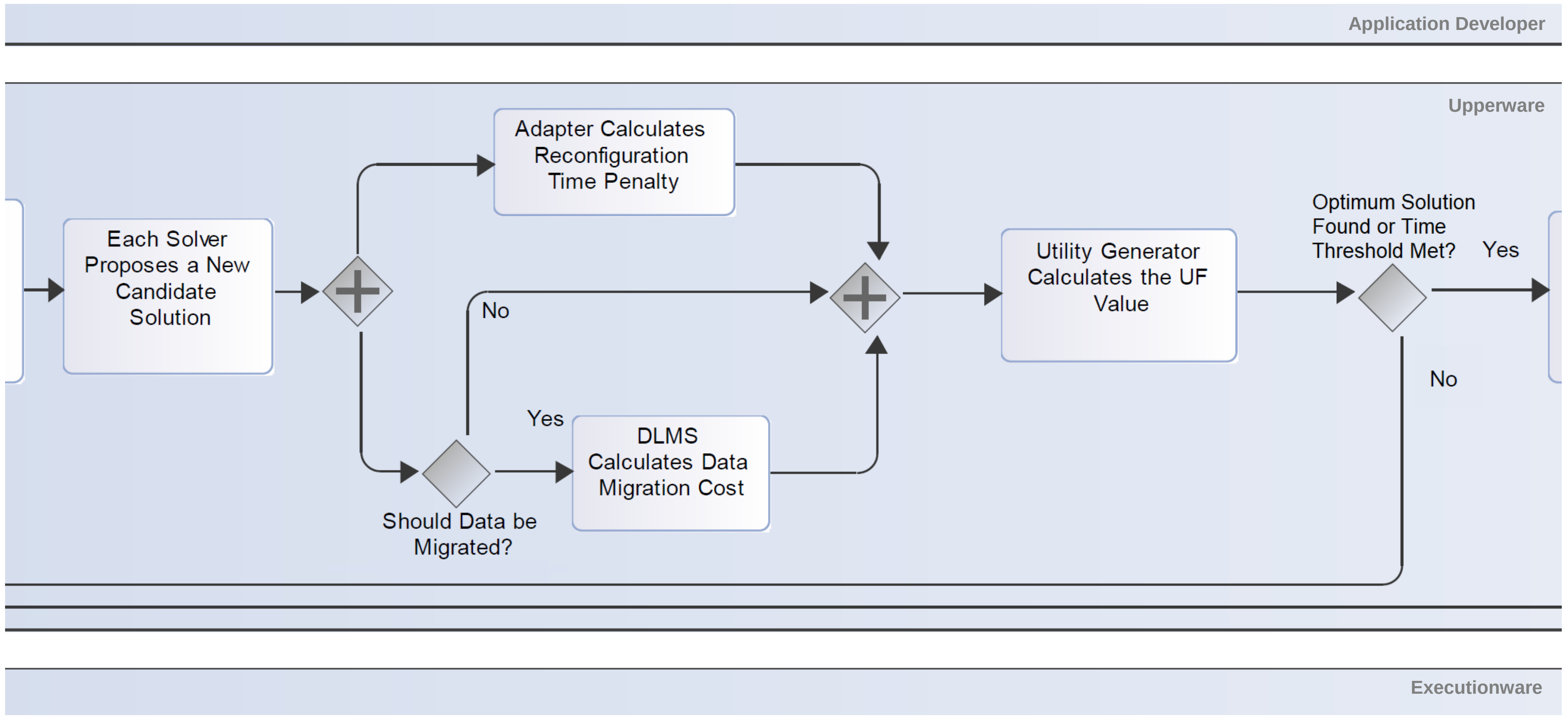

Application Developer
Upperware
Each Solver Proposes a New Candidate Solution
Adapter Calculates Reconfiguration Time Penalty
No
Yes
DLMS Calculates Data Migration Cost
Should Data be Migrated?
Utility Generator Calculates the UF Value
Optimum Solution Found or Time Threshold Met?
Yes
No
Executionware

Figure 18: Upperware BPMN diagram (reconfiguration) – enlarged, part II of IV

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

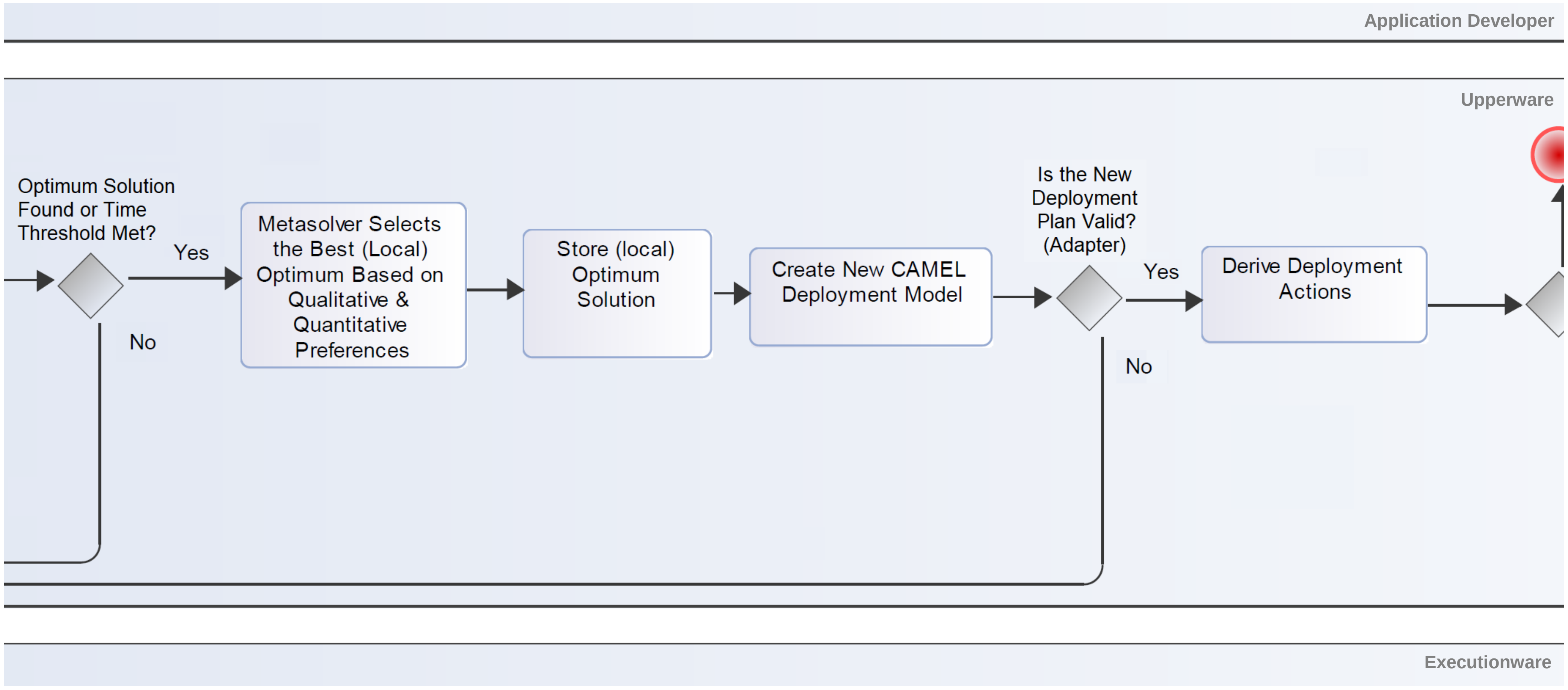

Application Developer
Upperware
Optimum Solution Found or Time Threshold Met?
Yes
No
Metasolver Selects the Best (Local) Optimum Based on Qualitative & Quantitative Preferences
Store (local) Optimum Solution
Create New CAMEL Deployment Model
Is the New Deployment Plan Valid? (Adapter)
Yes
No
Derive Deployment Actions
Executionware

Figure 19: Upperware BPMN diagram (reconfiguration) – enlarged, part III of IV

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

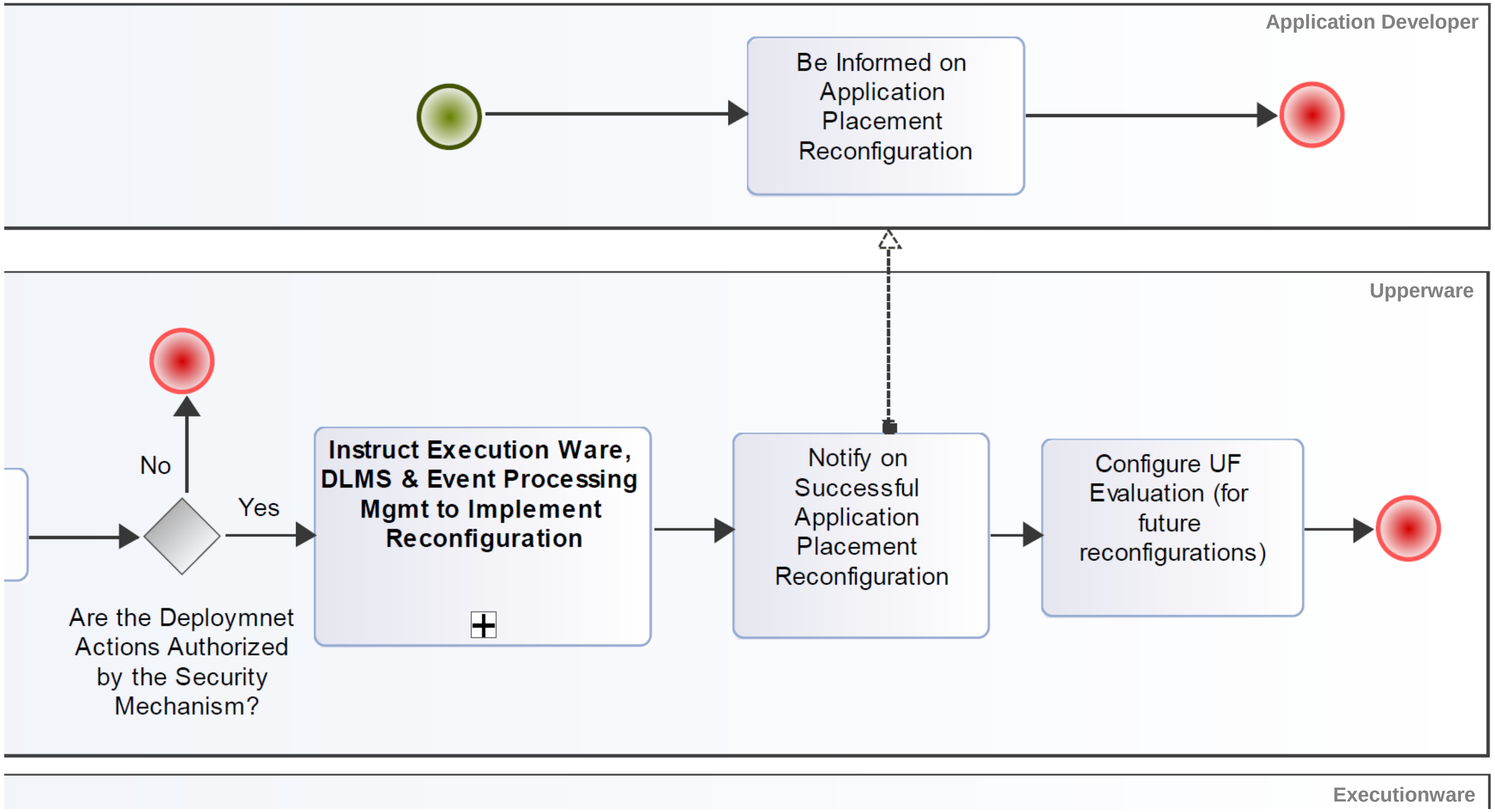

Application Developer
Be Informed on Application Placement Reconfiguration
Upperware
No
Yes
Are the Deploymnet Actions Authorized by the Security Mechanism?
Instruct Execution Ware, DLMS & Event Processing Mgmt to Implement Reconfiguration
Notify on Successful Application Placement Reconfiguration
Configure UF Evaluation (for future reconfigurations)
Executionware

Figure 20: Upperware BPMN diagram (reconfiguration) – enlarged, part IV of IV

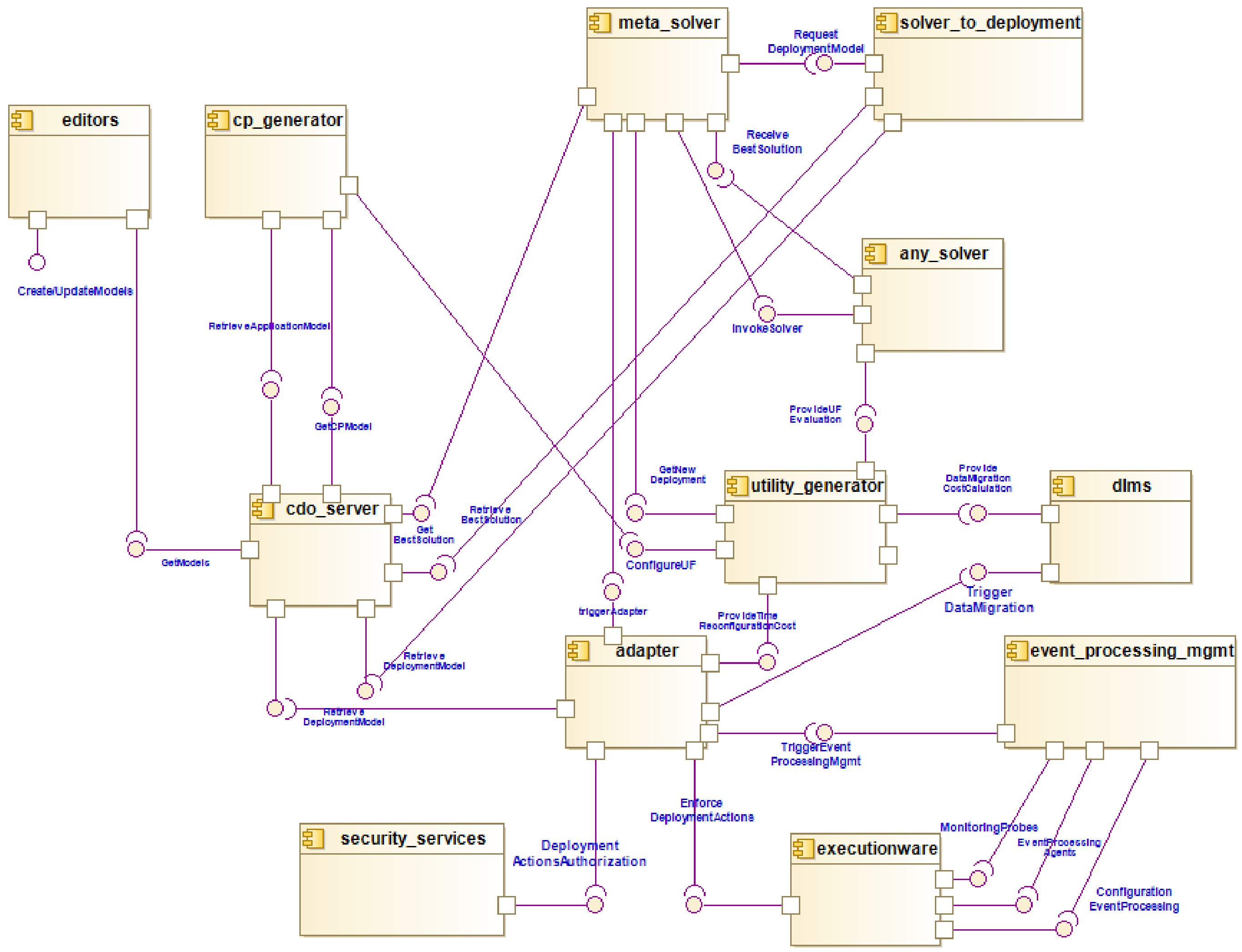


Figure 21: Upperware UML Component Diagram

Next, the Metasolver requests Solver-to-Deployment component to create and store the CAMEL deployment model that will drive the placement implementation orchestrated by the Adapter and executed by the components of the Executionware. We note that, before the Adapter is allowed to proceed with the new solution, there can be an optional manual validation step by the DevOp that may indicate his consensus for allowing the initial placement or the re-configuration of a big data intensive application. Then, the Adapter instructs the commissioning of Cloud resources and the data/application placement according to an action graph devised based on the CAMEL deployment model. After this, the Event Processing Management component is instructed to initiate the deployment of the monitoring probes and event processing agents as explained in sections 4.8 and 4.9. Besides, if any data artefacts should be moved from their current hosting environments, then the DLMS will undertake their migration. Both Adapter's and DLMS activities are authorised by the security services described in section 6.1. Upon a successful implementation of the initial application placement, all the necessary configurations (described in sections 3.1.8 & 3.1.9) will take place for detecting situations in which adaptation might be needed.

Based on the distributed event processing approach that Melodic will adopt, the Upperware will be informed and triggered on time about any scalability requirements or SLOs violations that will be intercepted as complex events. Other triggers for initiating the reconfiguration process may constitute any change in the Cloud provider offerings (e.g., provider outage, replacement of a certain VM offering). We note that based on how the application placement has been configured there are cases where small, automatic scaling actions (e.g., adding 2 new instances of a certain application component) can be directly performed by Executionware in order to guarantee the appropriate operation of the Melodic-enabled application. In either case, the Upperware will be informed and a process for calculating the new placement optimum solution will be triggered. During this reconfiguration process, the Metasolver updates the CP Model in the Models Repository (based on the relevant monitored data) and invokes again the relevant Solvers with (probably) a shorter time threshold. Solvers request the Utility Generator to provide UF values, while the latter invokes both DLMS and Adapter for receiving data migration costs (if it is implied by the candidate solution) and reconfiguration penalties, respectively. These costs are considered in the calculation of the UF value provided back to the Solver. After the time threshold is reached, the Metasolver compares the available local optimum solutions against the current (to be adapted) placement topology as it has been implemented by the Executionware. The best option is selected and if it does not match with the current implementation then once again the Adapter will undertake the task of devising an action graph based on which it will instruct the Executionware, DLMS and Event Processing Management components to implement the new placement solution for the application, its data and the monitoring and event processing infrastructure.

The provided and required interfaces of the Upperware components are depicted in the component diagram of Figure 21. We note that the control plane was omitted for improving the readability of this diagram. Moreover, the sequence of the interactions involving all Upperware components is also depicted in the Upperware sequence diagram (Figure 22) where more details are presented. We note that this sequence diagram is also provided in three parts for improved readability (Figures 23, 24, 25).

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

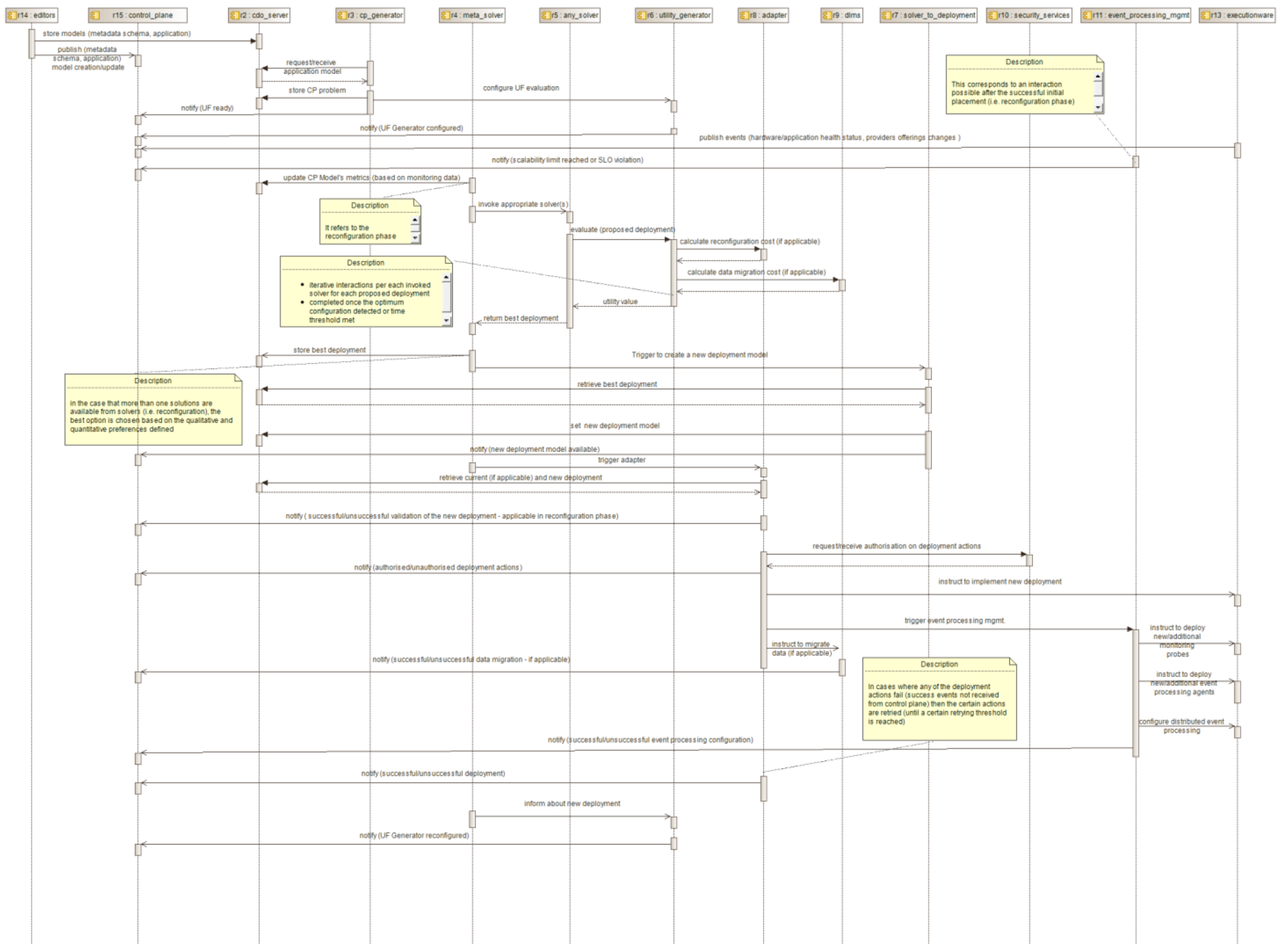

r14 : editors
r15 : control_plane
r2 : cdo_server
r3 : cp_generator
r4 : meta_solver
r5 : any_solver
r6 : utility_generator
r8 : adapter
r9 : dlms
r7 : solver_to_deployment
r10 : security_services
r11 : event_processing_mgmt
r13 : executionware
store models (metadata schema, application)
publish (metadata schema, application) model creation/update
request/receive application model
store CP problem
configure UF evaluation
notify (UF ready)
notify (UF Generator configured)
Description
This corresponds to an interaction possible after the successful initial placement (i.e. reconfiguration phase)
publish events (hardware/application health status, providers offerings changes)
notify (scalability limit reached or SLO violation)
update CP Model's metrics (based on monitoring data)
Description
It refers to the reconfiguration phase
invoke appropriate solver(s)
evaluate (proposed deployment)
calculate reconfiguration cost (if applicable)
calculate data migration cost (if applicable)
Description
iterative interactions per each invoked solver for each proposed deployment
completed once the optimum configuration detected or time threshold met
utility value
return best deployment
store best deployment
Trigger to create a new deployment model
Description
in the case that more than one solutions are available from solvers (i.e. reconfiguration), the best option is chosen based on the qualitative and quantitative preferences defined
retrieve best deployment
set new deployment model
notify (new deployment model available)
trigger adapter
retrieve current (if applicable) and new deployment
notify (successful/unsuccessful validation of the new deployment - applicable in reconfiguration phase)
request/receive authorisation on deployment actions
notify (authorised/unauthorised deployment actions)
instruct to implement new deployment
trigger event processing mgmt.
instruct to deploy new/additional monitoring probes
instruct to migrate data (if applicable)
notify (successful/unsuccessful data migration - if applicable)
Description
In cases where any of the deployment actions fail (success events not received from control plane) then the certain actions are retried (until a certain retrying threshold is reached)
instruct to deploy new/additional event processing agents
configure distributed event processing
notify (successful/unsuccessful event processing configuration)
notify (successful/unsuccessful deployment)
inform about new deployment
notify (UF Generator reconfigured)

Figure 22: Upperware UML Sequence Diagram

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

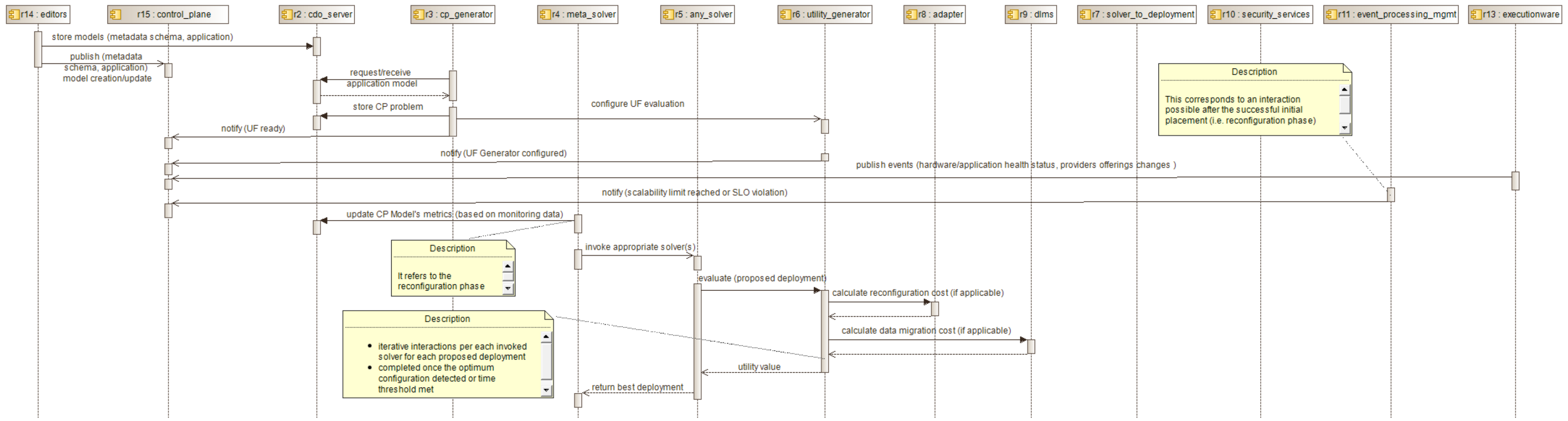

r14 : editors
r15 : control_plane
r2 : cdo_server
r3 : cp_generator
r4 : meta_solver
r5 : any_solver
r6 : utility_generator
r8 : adapter
r9 : dlms
r7 : solver_to_deployment
r10 : security_services
r11 : event_processing_mgmt
r13 : executionware
store models (metadata schema, application)
publish (metadata schema, application) model creation/update
request/receive application model
store CP problem
configure UF evaluation
notify (UF ready)
notify (UF Generator configured)
publish events (hardware/application health status, providers offerings changes)
notify (scalability limit reached or SLO violation)
update CP Model's metrics (based on monitoring data)
invoke appropriate solver(s)
evaluate (proposed deployment)
calculate reconfiguration cost (if applicable)
calculate data migration cost (if applicable)
utility value
return best deployment
Description
This corresponds to an interaction possible after the successful initial placement (i.e. reconfiguration phase)
Description
It refers to the reconfiguration phase
Description
iterative interactions per each invoked solver for each proposed deployment
completed once the optimum configuration detected or time threshold met

Figure 23: Upperware UML Sequence Diagram in parts (1/3)

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

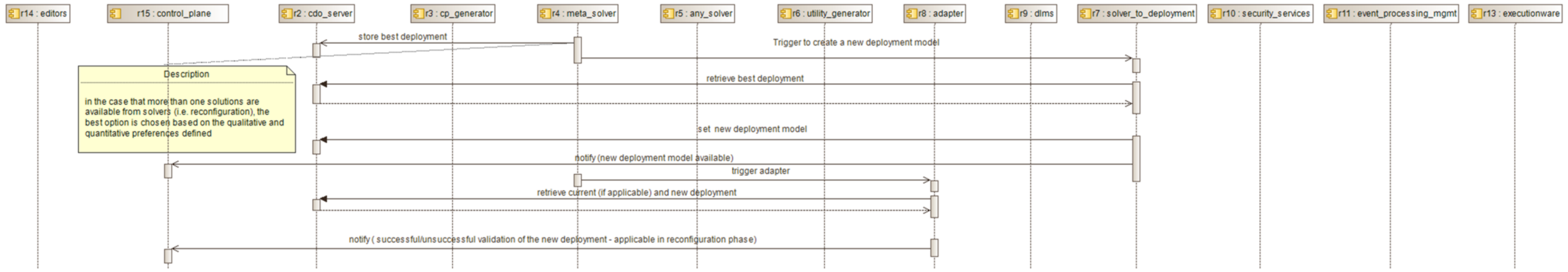
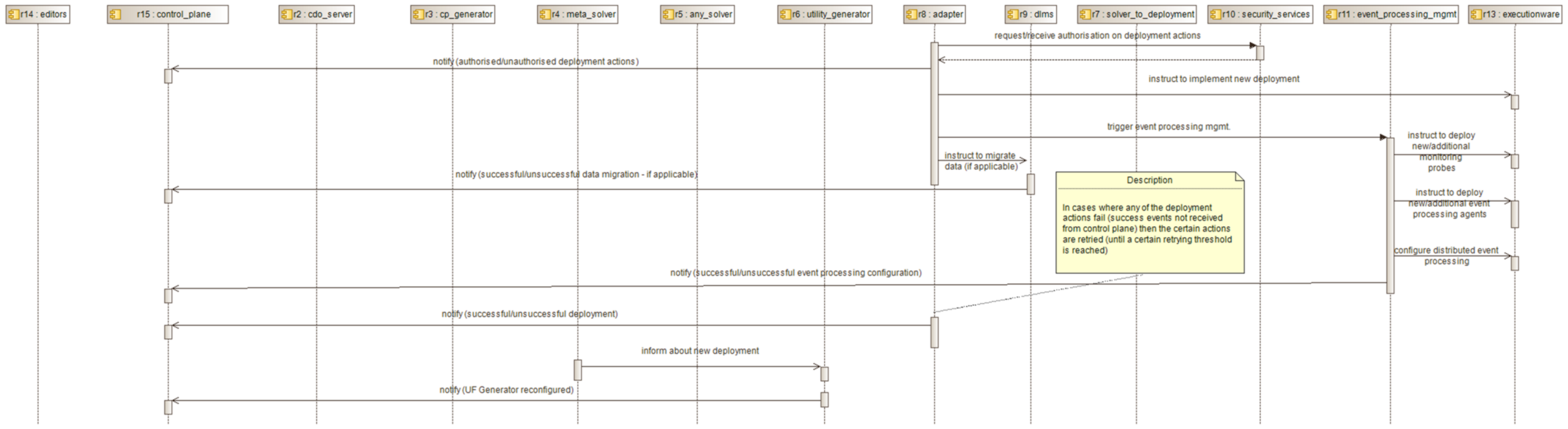

r14 : editors
r15 : control_plane
r2 : cdo_server
r3 : cp_generator
r4 : meta_solver
r5 : any_solver
r6 : utility_generator
r8 : adapter
r9 : dlms
r7 : solver_to_deployment
r10 : security_services
r11 : event_processing_mgmt
r13 : executionware
store best deployment
Trigger to create a new deployment model
Description
in the case that more than one solutions are available from solvers (i.e. reconfiguration), the best option is chosen based on the qualitative and quantitative preferences defined
retrieve best deployment
set new deployment model
notify (new deployment model available)
trigger adapter
retrieve current (if applicable) and new deployment
notify (successful/unsuccessful validation of the new deployment - applicable in reconfiguration phase)

Figure 24: Upperware UML Sequence Diagram in parts (2/3)


r14 : editors
r15 : control_plane
r2 : cdo_server
r3 : cp_generator
r4 : meta_solver
r5 : any_solver
r6 : utility_generator
r8 : adapter
r9 : dlms
r7 : solver_to_deployment
r10 : security_services
r11 : event_processing_mgmt
r13 : executionware
request/receive authorisation on deployment actions
notify (authorised/unauthorised deployment actions)
instruct to implement new deployment
trigger event processing mgmt.
instruct to deploy new/additional monitoring probes
instruct to migrate data (if applicable)
notify (successful/unsuccessful data migration - if applicable)
Description
instruct to deploy new/additional event processing agents
configure distributed event processing
notify (successful/unsuccessful event processing configuration)
notify (successful/unsuccessful deployment)
inform about new deployment
notify (UF Generator reconfigured)

In cases where any of the deployment actions fail (success events not received from control plane) then the certain actions are retried (until a certain retrying threshold is reached)

Figure 25: Upperware UML Sequence Diagram in parts (3/3)

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

4 Executionware

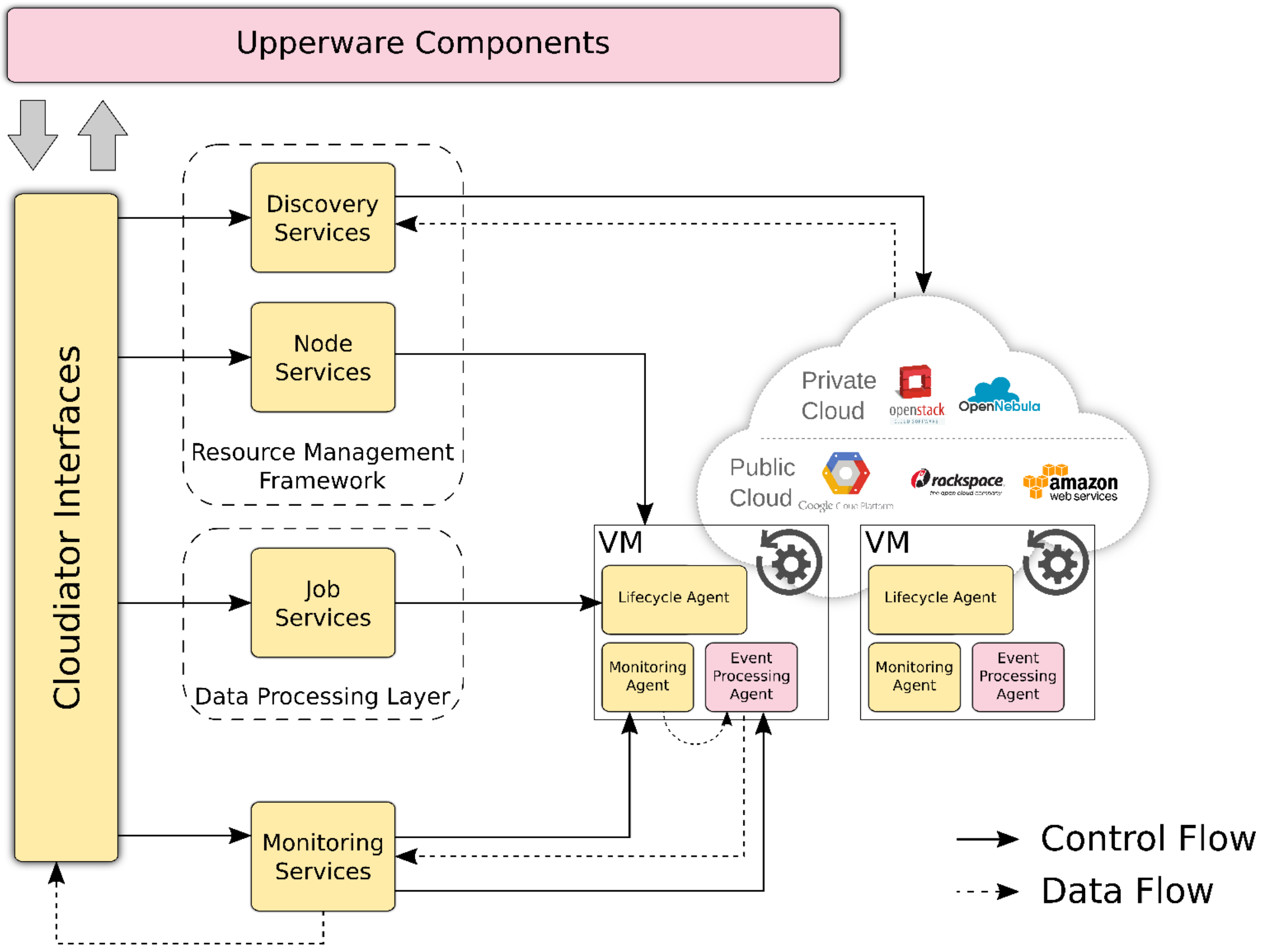


Figure 26: High-level Executionware Architecture

In the overall workflow of Melodic, the Upperware interacts with the Executionware in order to deploy data intensive applications, specified in the user-supplied CAMEL application model, to the Cloud. Hence, the main responsibilities of the Executionware are the management of diverse Cloud resources in a provider-agnostic way, the orchestration of data intensive applications on top of these Cloud resources and the deployment of the monitoring services to collect run-time metrics.

4.1 Cloud Orchestration

A high-level view on the Executionware components and its interaction is depicted in Figure 26. The core functionalities of the Executionware are built upon the Cloud orchestration tool (COT) [32], Cloudiator [33] [34]. Hence, the *Cloudiator Interface* represents the central entry point for the Upperware to interact with the Executionware. Cloudiator was initially developed within the PaaSage project with the focus on orchestrating web-based applications components in a Cross-Cloud context. Within the CACTOS project, Cloudiator was enhanced with respect to private Cloud deployment by adding functionalities, such as the discovery of private Cloud resource offerings or the optimised virtual machine placement by selecting the physical host to run the virtual machine. As Melodic requires the management of more diverse Cloud resources in conjunction with the orchestration of Big Data processing frameworks, Cloudiator will be redesigned in the context of Melodic in order to provide a more flexible and extensible architecture, adding a holistic *Resource Management Framework* and *Data Processing Layer* to Cloudiator. Further, the *Monitoring Services* of Cloudiator will be extended to support the integration of Event Processing Agents of the Upperware's Event Probes Manager.

As the enhanced architecture of Cloudiator is inspired by recent advancements in large-scale cluster management systems, its terminology for executable entities differs from the CAMEL terminology. The basic entities are depicted in Figure 27. While CAMEL features an application centric view, where the users model applications and their components, Cloudiator's entities are more execution centric. It, thus, uses a *Job entity* to group multiple independent, executable *Task entities* that require execution. One execution of a task is represented by a *Process entity*, therefore mapping exactly one task entity to a *Node* (e.g. representing a VM). We therefore map an application described by CAMEL to one *Job*, and each application component described in CAMEL to a *Task*. Instances described in CAMEL are mapped to process entities in Cloudiator.

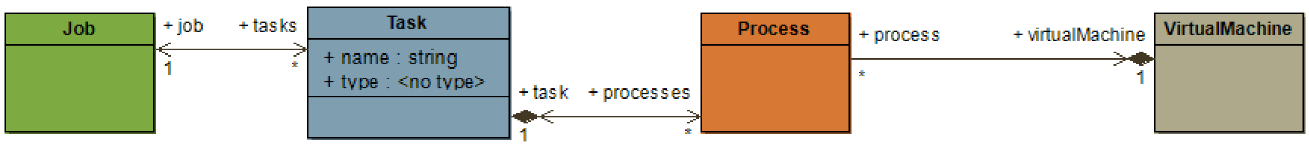


Figure 27: Cloudiator Executable Entity Terminology

The workflow of interactions between the Upperware and the Executionware is depicted in more detail in Figure 28 by explaining the workflow addressing the placement of both a job as well as the corresponding monitoring/event processing. The application deployment is initiated by the Upperware by requesting a new deployment by the Executionware. Thus, the Executionware provisions the required nodes and deploys the specified processes on top of the nodes. These processes comprise traditional web-based processes, data processing processes such as Apache Spark applications and PaaS processes. Cloudiator already offers the life-cycle agent Lance for web-based applications and will be extended with additional life-cycle handler to support data processing and PaaS processes. After the deployment of the processes, the Upperware is requesting from the Executionware the deployment and configuration of the Event Processing and Monitoring on the provisioned nodes at the Executionware. After executing these steps, the Executionware notifies the Upperware about the deployment state, i.e., success or possible error messages in order to retry the deployment with a different configuration.

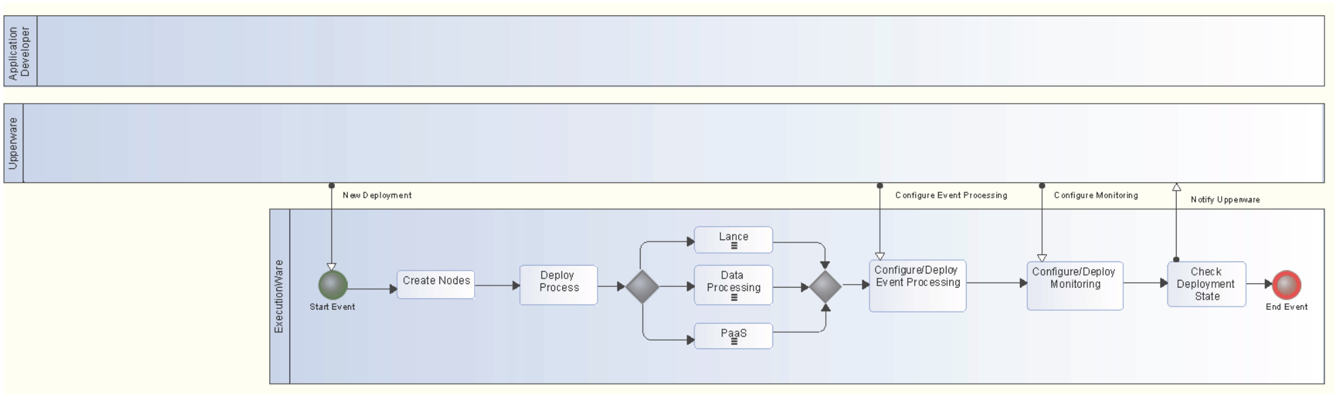


Figure 28: Executionware Deployment BPMN

A high level view on the Cloudiator components is depicted in Figure 14. As entry point to Cloudiator, a REST interface is provided. The interaction with the REST interface is enabled via a Web-based *UserInterface* and programmatically via API client libraries for multiple programming languages. Consequently, the programmatic interaction is exploited in the Upperware, which is interacting with the REST Server via the API client library.

Behind the REST Server, Cloudiator is built upon a message-driven architecture, following the publish-subscribe paradigm. Therefore, each call against the *RestServer* is transformed into a Cloudiator specific message and published to the *KafkaMessageQueue*. For the sake of clarity, the *KafkaMessageQueue* component of the Cloudiator architecture is depicted without the dedicated interaction between all Cloudiator components. Yet, all internal communication (behind the *RestServer*) of Cloudiator relies on messages over the *KafkaMessageQueue.*

4.2 Resource Management

Building upon the initial Cloudiator features, the new architecture enhances Cloudiator with the dedicated *Resource Management Framework* and Data Processing Layer components as depicted in Figure 29. In this respect, Cloudiator provides the automated discovery of Cloud resource offering of IaaS providers, which can be triggered via *DiscoveryQueries.* The *DiscoveryAgent* collects the resource offerings from the Cloud providers, while the

DiscoveryRegistry stores the collected resource offerings. The collected resource offerings can be enriched via additional metadata from third party providers, such as CloudHarmony[12], by the *MetaDataService*. The discovery and the actual provisioning of Cloud resources across multiple Cloud providers is enabled by the *Provider Agnostic Interface Mapper*, which is integrated into the *Resource Management Framework* of Cloudiator.

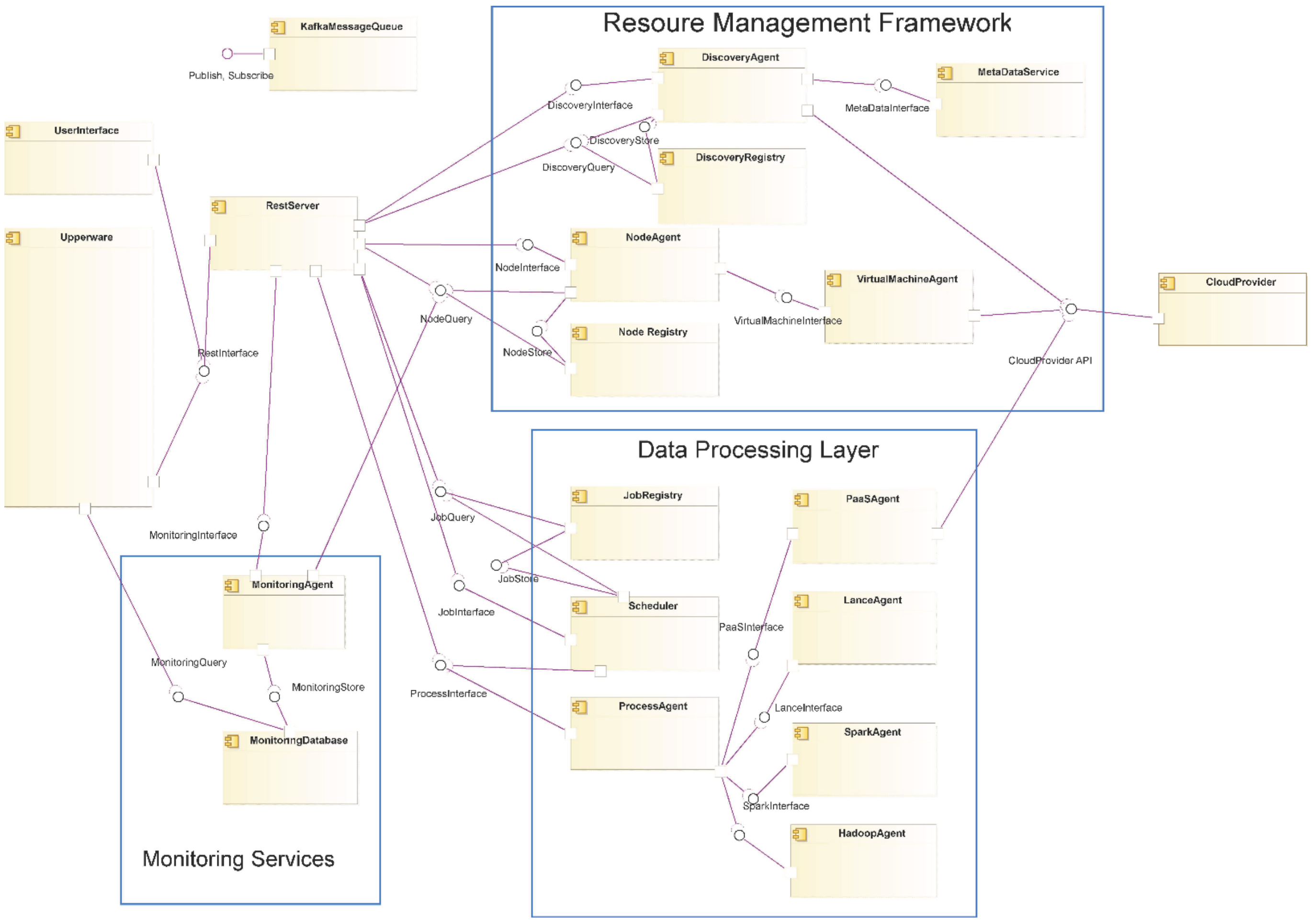


Figure 29: Cloudiator Architecture

As the Upperware requests a new deployment by the Executionware, Cloudiator executes a sequence of multiple steps across the Cloudiator components, which is depicted in the sequence diagram in Figure 30.

12 https://cloudharmony.com/

4.3 Data Processing Layer

A new deployment is requested by the Upperware at the *RestServer* component of Cloudiator. Consequently, the *RestServer* creates an internal message which is published via the *KafkaMessageQueue*, triggering the *NodeAgent* to allocate the required number of nodes at the specified Cloud providers. As depicted in Figure 14, this step can be executed multiple times, depending on the specified resources by the Upperware. The *NodeAgent* is responsible for allocating new Cloud resources, i.e., triggering the provisioning of VMs via the *VirtualMachineAgent* and the bootstrapping of the VM with internal Cloudiator tools, namely Lance for the lifecycle management of the processes and Visor (our *MonitoringAgent*) for their monitoring [23]. In addition, the *NodeAgent* will be able to include existing nodes into the orchestration workflow of Cloudiator, e.g., non-Cloud resources such as physical servers located a user site. All nodes are stored in the *NodeRegistry* and can be queried via the REST interface. Further, the modular architecture of the Resource Management Framework allows the integration of other resources management frameworks such as Apache Mesos[13] or Hadoop YARN[14] as special *NodeAgent* types.

After the provisioning of the required nodes, the *NodeAgent* publishes a message in the *KafkaMessageQueue,* indicating the initiated nodes and triggering the *Data Processing Layer*, which takes over with the *Scheduler* transforming the defined Tasks into deployable Processes and triggers their orchestration via the *ProcessAgent*. Due to the heterogeneity of the Processes, from traditional web-based Processes to data-intensive Big Data Processes, the *ProcessAgent* triggers the required agent to orchestrate the Process on the defined resources: the *LanceAgent* orchestrates Processes on IaaS level, the *PaaSAgent* orchestrates Processes running on PaaS level and the *Data Processing Process* for data processing clusters. Due to the modular architecture of Cloudiator, a Data Processing process can be implemented via *SparkAgents* to orchestrate Apache Spark clusters[15] or *MapReduceAgents* for Hadoop MapReduce[16] clusters. The information of the Jobs, their Tasks and the deployed Processes is stored in the *JobRegistry*, which can be queried via the *RESTServer*. In case of unsuccessful deployments due to resource or task failures, Cloudiator will inform the control plane of the Upperware to trigger a redeployment. The monitoring and event processing configuration and deployment of the nodes is managed via the *MonitoringAgent,* which configures and triggers the monitoring probes on selected nodes based on the instructions of the Upperware. Based on the provided configuration, raw monitoring data from the respective nodes is provided to the *Event Processing Agents* and stored in the *MonitoringDatabase*, which is queried by the Upperware.

[13] http://mesos.apache.org/
[14] https://hadoop.apache.org/docs/r2.7.2/hadoop-yarn/hadoop-yarn-site/YARN.html
[15] https://spark.apache.org/
[16] https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

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

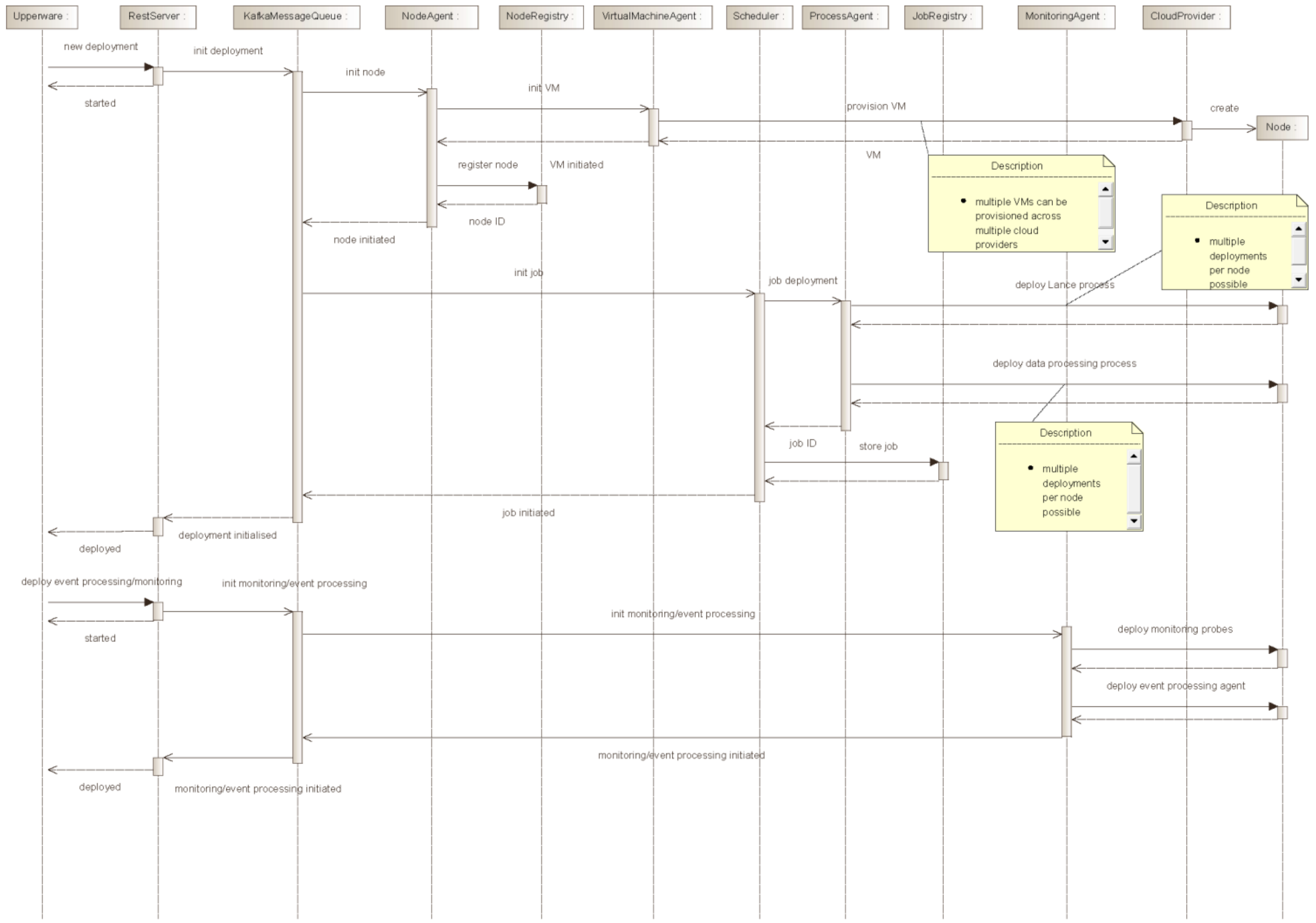


Figure 30: Executionware deployment sequence diagram

5 Auxiliary Services

In this chapter, we describe two important auxiliary components in the Melodic middleware platform, the *Security Services* and *Status and Event Service* (as shown in Figure 5). The *Security Services* component includes sub-components necessary to realise secure operations within the Melodic platform. The status and event Service is a utility service offering a unified and integrated event notification mechanism for the Melodic components.

5.1 Security Services

Security services encompass those architectural components responsible for authenticating and authorising the actions of core Melodic components, related to the implementation of data placement and application actions. They comprise of the Authentication and Authorisation services as well as a security and access policy repository.

Authentication service securely stores all credentials used for accessing Cloud providers services or for attesting Melodic components when making decisions, commissioning or decommissioning Cloud resources, placing or reconfiguring an application and migrating data. It may contain an actual credentials store or alternatively use an external identity system (e.g., LDAP or other).

Authorisation Service corresponds to a policy-based security mechanism that enables the adequate protection of sensitive Melodic platform resources (such as components, workflows and data), both from unauthorised access attempts as well as from compromised or misbehaving platform parts (due to cyberattacks). Based on this service, we are introducing an additional guarantee, in case of a cyberattack, that the Melodic platform will not deploy any application changes, which contradict the constraints set by the application developer or the DevOps. The policies involved are Attribute-Based Access Control (ABAC) policies, expressed using the eXtensible Access Control Markup Language (XACML) [35], which is the de facto standard in XML-based access control policies. The language is generic and extensible enough as it provides several syntactical artefacts that enable the expression of complex access control rules using contextual information. The authorisation service encompasses an XACML-compatible policy engine, as well as the corresponding XACML repository.

Security services communicate with the rest of Melodic components through ESB. They receive events about the availability of new placements, and publish events reporting the security check outcome (permit or deny, as well as the reasons of such denial). Security check reports can be stored in the Models Repository alongside the application models and other artefacts. Security services input can either be authentication requests or new configuration and deployment actions. The former may originate from any Melodic component whereas the latter may come from the Upperware's Adapter. In both cases, the output is routed back to the requesting component (originator).

From a conceptual perspective, Security services interface with Melodic Upperware and Executionware for authorisation, as well as those Melodic components that require authentication. In technical terms, they communicate with the ESB in order to send and receive events, the Models Repository in order to retrieve deployment models and other needed information while we may also consider the use of an external identity server.

5.2 Status and Event Service

The Status and Event Service is a utility service that encompasses operational status notifications and control events to the Melodic platform. Following operations will be realised by this service:

- GetStatusData – returning deployment/reasoning status of the given application with additional data (deployment plan, found solutions, etc.)
- UploadData – method which allows to upload CAMEL models into the platform
- StartReasoning – method which allows to start reasoning process
- StartDeployment – method which allows to start deployment process.

The service will be implemented as an ESB service. The communication between integration layer ESB and this service will be realised through a REST API. The list of exposed methods could be extended in the future. This service will be also used by User Interface component of the Melodic platform. Further, the usage by external systems (e.g., monitoring systems) will be possible as future extensions.

6 Initial Feature Definitions

In this chapter, we describe key Melodic capabilities and list its salient features. The feature definitions presented here correspond to the requirements gathered and presented in the deliverable D2.1 System Specification. To start the chapter, in section 7.1, we briefly present the timeline of the planned Melodic releases so that we can associate a target release to each of the features we describe later in the chapter. Next, we list the Melodic capabilities and the initial associated features, stemming from the generalised requirements of efficient execution of data-intensive applications in Cross-Cloud environments, in section 7.2. Next, in section 7.3, we move on to the list of Melodic features originated from the specific use-case requirements of our four demonstration use-cases. All feature requirements are gathered and documented in the Melodic's JIRA[17]. Once the features corresponding to the generic and use-case requirements have been described, we present features beneficial to satisfy non-functional requirements of the Melodic middleware platform in section 7.4. Note that the feature set presented in this document is initial based on the requirements gathered in the first iteration (as reported in *D2.1 System Specification*). As new requirements may come during the project, for instance as feedback from iterative use-case implementations, the final list of features will be presented in the deliverable '*D2.6 Final Features*' towards the end of the project.

6.1 Planned Software Releases

As per the description of action, the Melodic project plans to have three releases. The first release, due in M12, is the *integration release* integrating the identified components of existing platforms developed by the PaaSage, CACTOS, and the PaaSword projects. The first release will be able to deploy applications on resources acquired from multiple Cloud providers and monitor them, however, without any data-awareness. The second release, due in M24, adds both the data-awareness in the Melodic platform and the security component. This release also encapsulates the initial integrated platform with the big data processing frameworks for deploying user big data applications. The second release, due in M24, enhances the first platform release with additional capabilities spanning data-awareness and security as well as encompasses initial integration with selected big data processing frameworks, thus making it capable to support the deployment of big data applications. The third release, *final release,* includes the feedback from the second release and incorporates the final set of features at the project end (M36).

[17] https://jira.7bulls.eu/

Following a comprehensive evaluation and code assessment of the available components from PaaSage, CACTOS, and PaaSword (as described in *D2.1*), it was observed that some key integration functionalities are missing, as well as some component redevelopment and code refactoring is necessary to build a solid foundation for the upcoming releases, and to achieve efficient software maintenance. The features missing in PaaSage, for instance, the professional enterprise bus and flexible orchestration using BPM, in the long term would hinder both the flexibility and easy extensibility of the platform. Therefore, the consortium decided to pay special attention to building a solid code-base foundation in the first release. Yet, as some of the component integration is not possible in the tighter first release schedule, the Melodic consortium agreed to include another release between the integration release, and the prototype release. The added *intermediate release,* versioned as Release 1.5, will complete the integration of the available components from the parent projects on the solid foundation led by the first release. Additionally, it will introduce some new features (previously not planned before the second release) as described in more detail in section 7.3 and section 7.4. The key changes as featured in the integration layer over the parent projects are subject of *D5.02 Update to OSS frameworks.*

The updated release timeline is shown in Figure 31.

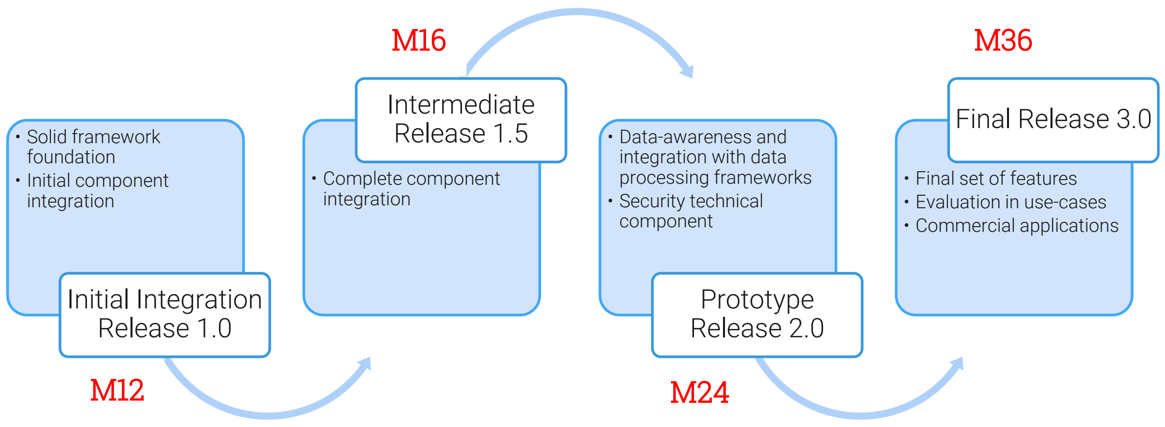

M12
Initial Integration Release 1.0
• Solid framework foundation
• Initial component integration
M16
Intermediate Release 1.5
• Complete component integration
M24
Prototype Release 2.0
• Data-awareness and integration with data processing frameworks
• Security technical component
M36
Final Release 3.0
• Final set of features
• Evaluation in use-cases
• Commercial applications

Figure 31: Updated release timeline for the Melodic middleware platform

6.2 High-Level Melodic Capabilities and Salient Features

The high-level Melodic capabilities and corresponding features based on the generalised requirements of deployment and execution of data-intensive applications on Cross-Clouds are provided in Table 2: Melodic capabilities and features corresponding to the generalised Cross-Cloud requirements.

Table 2: Melodic capabilities and features corresponding to the generalised Cross-Cloud requirements.

Capability	Features
Transparent Deployment and Execution of Data-Intensive Applications on Cross-Cloud infrastructures	• Ability to comprehensively model data-intensive applications and requirements, covering both design-time and runtime parameters • Support of provider-agnostic Cloud interfaces for IaaS and PaaS platforms • Ability of automated deployment of applications in distributed Cloud environments • Support for the deployment of two major big data processing frameworks, Apache Hadoop MapReduce and Apache Spark • Implementation of a Cross-Cloud resource management system for the data-processing frameworks • Realisation of efficient algorithms for data-aware application component deployments over Cross-Cloud resources
Holistic Data Management	• Ability to comprehensively model specification and requirements for heterogeneous data sources • Data-awareness in Cross-Cloud application deployment models • Efficient data placement and migration methodologies on Cross-Cloud platforms • Context-aware data access control • Metadata-schema with appropriate editors

Runtime Adaptation of deployed data-intensive applications	• Ability to calculate optimised Cross-Cloud resource allocations for data-intensive applications • Support for user-defined scalability rules for intra-Cloud application scaling • Ability to improve reasoning accuracy over time using historical data and statistical models • Availability of efficient re-configuration mechanisms • Support for proactive reconfiguration of applications
Privacy and Confidentiality for applications deployed on Cross-Clouds	• Context-aware access control mechanisms for distributed cross-domain deployments of data-intensive applications
Application Support	• Support for MapReduce and Spark big data processing frameworks • Application components correctly mapped on heterogeneous Cross-Cloud infrastructures according to the application requirements • Data-aware deployment of big data applications
Private Cloud Resource Management	• Optimal usage of private Cloud resources by exploiting, for instance, topology, and hardware-specific information unavailable on public Cloud platforms
Application Scalability and Availability	• Support for computational and data scaling • Support for application-defined Cloud-specific scalability rules • Utility-based scalability through global application reconfiguration • Transparent exploitation of geo-graphically dispersed Cloud locations to increase application availability

6.3 Features corresponding to the use-case requirements

In Table 3: Features corresponding to the use-case requirements, related components, and target releases.3, we list the features extracted from the use-case requirements. We also list the affected component or component groups in the Melodic platform as well as the target release for the realization of these features. Note that many of these features indirectly correspond to the capabilities and features listed in section 7.2.

Table 3: Features corresponding to the use-case requirements, related components, and target releases.

Feature Specification and Benefits	Components	Target Release
Ability to add application components/component instances during runtime Ability to add new application components and component instances during runtime will reduce reconfiguration downtime for the use-case applications.	User Interface, Upperware, Executionware	3.0
Running big data frameworks over aggregated resources from the infrastructures of different Cloud providers Transparent execution of data-intensive applications and components on Cross-Clouds will help taking full advantage of different offerings of the Cloud providers, and will optimise costs and running time for the user jobs.	Upperware, Executionware	2.0
Automatic configuration of the big data frameworks Automatic configuration of the big data frameworks on acquired Cloud resources, based on information specified in CAMEL, will substantially decrease deployment and configuration time for the big data applications.	Executionware	2.0

Efficient optimisation of the big data frameworks based on historical data from metrics Efficient optimisation of the job allocations and data placement during runtime, based on gathered performance data, will optimise application performance and cost in the Cloud.	Upperware, Executionware	2.0
Reconfiguration of big data frameworks based on user-specification Application-specific reconfigurations are only possible when the reconfiguration of big data frameworks based on user-specification is supported.	Modelling, Upperware	2.0
Ability to specify scalability rules for the big data applications in CAMEL model Application developers need to define scalability rules for their big data applications to adapt with the dynamic workload.	Modelling, CAMEL Editor	1.5
Ability to use predefined big data metrics For quickly setting up monitoring, the basic big data framework metrics need to be selectable in the CAMEL model as Melodic metrics.	CAMEL Editor	2.0
Ability to specify initial configuration for the big data frameworks The initial configuration of the big data frameworks greatly influences the performance for the data-intensive applications. For Melodic users, the CAMEL model should be rich enough to specify initial configurations, e.g., executors per node, cores per executor, memory per executor etc.	CAMEL Editor, Executionware	2.0
Ability to use existing big data cluster A melodic user can only benefit from available existing big data cluster from Melodic, if the support to deploy applications on existing big data clusters, such as Spark cluster, is provided.	Upperware, Executionware	2.0

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

Ability to use custom image of big data frameworks Many users benefit from customisation of big data frameworks for their applications. So, it is important to have the capability of specifying custom binary image for the big data framework installations in the Cloud infrastructure.	Modelling, Executionware	3.0
Ability to utilise existing Mesos cluster Existing data centre Mesos cluster should be able to be utilised from the Melodic platform.	Executionware Upperware	3.0
Support of ProfitBricks[18] as infrastructure Cloud provider CAS Software needs deploy applications on VMs provided and managed by ProfitBricks as they have good presence and support in Germany. Cloudiator needs to be extended with an additional adapter/driver to communicate with the ProfitBricks API to start/shutdown VMs.	Cloudiator Interfaces	1.5
Ability to define scalability rules in the web-based editor A web-based editor will greatly simplify defining scalability rules in an intuitive way without the necessity to understand to whole complexity of the underlying model. For instance, in the form of "if CPU load > 90%, then scale out.	CAMEL Editor	2.0
Ability to define application/component requirements in the web-based editor A web-based editor will greatly simplify application requirement modelling without the necessity to understand the complete CAMEL metamodel.	CAMEL Editor	1.5
Ability to model application components in web-based editor A web-based editor will greatly simplify application modelling without the necessity to understand the complete CAMEL metamodel.	CAMEL Editor	1.5

[18] https://www.profitbricks.com/

Cloud service provider models In order to find an appropriate infrastructure for the deployment, the Cloud providers and the services they offer need to be described.	CAMEL, Executionware	2.0
Ability to specify utility and cost function for the Melodic Upperware As the utility of a deployment is based on the perception of the user, it is important that the user themselves can specify utility and cost functions used in the reasoning process of the Melodic Upperware.	CP Generator	3.0
Ability to utilise template utility profiles With reference to the use-case requirements pertaining to rapid utility function placement, *utility profiles* will be offered where users can base their utility function on the base templates and specify basic parameters, for instance response time requirements, without the need to specify the whole utility function itself.	User Interfaces	3.0
Ability to view/monitor deployed applications through a user interface The Melodic users would like to view and monitor their application deployments through the Melodic dashboard.	User interface	3.0
Ability to optimise private resource allocations through hardware-level selection As private Cloud infrastructure offer greater administrative capabilities for the workload optimisations, hardware-level selection of resources, topology exploitations will greatly improve deployment solutions.	Upperware	3.0
Support of PaaS Many Cloud providers offer useful PaaS services for the Cloud users, such as RDMS, or ElasticSearch. Enabling PaaS-based deployment and exploitation of PaaS services will improve the overall user-experience and support array for the Melodic.	Upperware, Executionware, CAMEL Editor	3.0

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

Support for DevOps tools like Chef/Puppet/Ansible User-case partners would like to use DevOps tools like Chef/Puppet/Ansible for managing the lifecycle of their application components. A preliminary approach could be to add support for the Ansible recipes into CAMEL instead of installation/configuration scripts, based on predefined Ansible templates.	Executionware	3.0
Utility generator interface Utility generator interfaces will enrich the optimisation capabilities of the Melodic Upperware	Utility Generator	1.5
Support of multiple solvers Different application mappings and requirements leads to different problem sets. Ability to use multiple solvers help reaching solution faster as the appropriate solver can be used according to the problem description.	Meta solver	2.0
Support of Component Composition Full support of component composition is needed to correctly model applications using such attributes.	CAMEL, CP Generator	3.0
Ability to specify component co-location parameter in the application modelling Performance of communicating application components improves considerably when they are co-located.	CAMEL, CP Generator	2.0
Allow replication of the components across Cloud providers Replication of critical components across Cloud providers will improve the application availability. From modelling perspective, this can be done by adding an attribute *Replication* in CAMEL to mark components which should be replicated between defined numbers of Cloud providers.	CAMEL, Upperware, Executionware Upperware,	2.0

Ability to reconfigure applications based on the delta deployments To avoid frequent orchestration and re-orchestration of the Cloud resources, it is important to have the ability to deploy only changes/delta between current deployed deployment plan and the new one.	Adapter	1.5
Ability to evaluate reconfiguration utility and overhead before the actual reconfiguration To avoid frequent reconfigurations and resultant downtime and overhead, the new deployment plan needs to be validated based on the differences in value of the utility function. If there are already deployed applications with given deployment plan, a new deployment plan should be deployed only if the utility of the new plan is better than the previous one (including the consideration of the reconfiguration overhead).	Adapter	2.0
Ability to optimise, reconfigure, scale application components Apart from optimising / reconfiguring the big data processing frameworks, other legacy application components might also need to be re-configured at runtime. For CAS use-case, based on the monitoring information and usage statistics, the applications and the data storage need to be scaled.	All	1.5

Ability to integrate Melodic with other systems through an API In order to integrate the Melodic framework with other systems, Melodic needs to offer an external programming interface. CE-Traffic needs such an API to easily deploy and reconfigure their applications. Third-party developers also need to use such as API to connect to the CAS App Store in order to deploy their apps. The CAS app store will use the Melodic API and offer a dedicated user interface to their developers.	Melodic Interfaces to the End Users, Melodic External APIs	2.0
Ability to specify backup strategy for the data sources The ability to specify backup strategies through the Melodic platform will help avoiding losing data and possible downtimes. For CAS, in case of deploying an app that uses its own data and data store (not acting on data stored and managed on CAS premises), it would be useful to define a backup strategy and let Melodic manage the data replication and backup.	DLMS, CAMEL	3.0

Ability to deploy several applications on one Melodic framework instance The use-case partners CE-Traffic and CAS Software will benefit from the ability to deploy multiple applications on one Melodic framework instance. The approach here would be that several applications are modelled as components of one larger parent application. Then, it should be possible to include new instances of existing component types to the model during runtime. This should trigger the parsing and reasoning of the model, and the subsequent deployment process. In CET use case, Melodic will be used to deploy several applications or several instances of the same application but with different settings, often sharing the same data sources. Therefore, in view of general deployment management and usability, Melodic should run as one instance supporting several deployments. As part of the CAS use case, Melodic will be used in combination with the app store to support the deployment (and dynamic deployment changes) for several apps running on the same CAS Open platform.	CAMEL Editors, Upperware, Executionware	2.0

Ability to deploy docker containers The use-case partners CE-Traffic and CAS Software will benefit from the ability to deploy docker containers via Melodic. For CE-Traffic, *Kubernetes*[19] dockers may be used to run multiple instances of the same application for the evaluation of traffic control settings (using traffic simulation or machine learning algorithms) in parallel. Each instance will be run with different parameters. CAS will investigate the use of docker containers for the deployment of either the platform itself or the 3rd party apps. CAS would profit from reusing the information provided in the docker file for the app description in CAMEL.	CAMEL, Executionware	2.0

6.4 Features corresponding to the non-functional requirements

In Table 4, we list the features that are not directly visible to the end users but contribute towards achieving the non-functional requirements specified for the Melodic platform. We have also indicated key targets for the first release.

Table 4: Features corresponding to the non-functional requirements

Non-Functional Requirement	Feature and Benefits
Extensibility and Openness	*Component Integration via Enterprise Service Bus (ESB)* ESB integration provides a consistent interface through which components interact with each other. This simplifies communication as the individual components need to conform

[19] https://kubernetes.io/

	only to the standard communication interface, and not implement direct communication between each other. Integration through ESB provides easy component integration, and improves platform extensibility and reusability. ESB integration is targeted for Release 1.0.
Software Maintainability	*Key component refactoring* Key platform components inherited from selected European projects need to be refactored/rewritten from scratch to improve quality, maintainability, and extensibility. For instance, the Adapter is rewritten and the CP Generator would be refactored for Release 1.0.
Reliability	*High-Availability (HA) configuration* Due to redesigning the integration layer in the Melodic platform, the system is ready for HA configuration, even a distributed one. Thanks to that it is possible to use the platform for the deployment of critical applications. The key components of the system could be deployed in multiple instances and fault tolerance will be achieved by proper configuration.
Flexible orchestration	*Orchestration by Business Process Management (BPM)* A business process is used to describe the data and control flow and though its execution an appropriate orchestration of the Melodic components is achieved. Thanks to BPM process orchestration, most of the changes in functional requirements related to the flow of actions/data could be implemented without changing the underlying software. This will speed up implementation of the required changes as the project progresses. Flexible orchestration through BPM will be incorporated in Release 1.0.

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

System Monitoring	*Unified Logging* As Control and Data flow in the Melodic platform is integrated using ESB, BPM, and REST APIs, all requests are centralised and logged in one place. This helps with troubleshooting the system and discovery of bugs or misconfiguration issues. The initial development is targeted for Release 1.0, and will be subsequently improved in later releases.
Unified deployment	*Using Docker components* Starting from Release 1.0, all components of the Melodic platform are deployed in containers managed by Docker Swarm. Due to that, the deployment of the platform will be very easy and flexible.
Security	*Implementation of ESB level security* Communication between Melodic components goes through the ESB. The security, authentication and authorisation of the method invocation is unified and managed by the ESB. Most of the authentication and authorization features are the target of Release 2.0.
Platform Scalability	*Implementation of Micro-services* The Melodic platform, starting from Release 1.0, due to the use of Docker containers and ESB, is built around a micro-services architecture. Thanks to that, the platform is scalable, both at the component and integration level and could be used to provision large scale, big data applications.

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

7 Conclusions

In this document, we have presented the final Melodic architecture. We identified key Melodic components and presented their internal architecture. We also presented the interaction between different Melodic components and the corresponding control and monitoring data flow. Furthermore, the document also listed key Melodic capabilities, its initial feature definitions, and the corresponding non-functional requirements. This document will serve as a guiding reference to the implementation of the Melodic framework, its external APIs, and use-case demonstrators.

The architecture of Melodic is greatly influenced by the PaaSage project and we take a model-driven engineering approach where applications and corresponding datasets are first modelled using a domain-specific language, CAMEL, so that the Melodic platform can reason about their optimal Cross-Cloud deployments according to the formal specification of the applications in CAMEL. The platform is conceptually divided into three main component groups: the Upperware, the Executionware, and the interfaces for user-platform interactions. Application deployments are continuously monitored and analysed by the Melodic platform, and if the current deployment is no longer optimal, a new deployment solution is calculated and the adaptation is planned and executed.

Based on the requirements gathered in the *System Specification Document,* key Melodic capabilities and its salient features have been identified. Three main Melodic releases are planned and the target features for each of the release have been assigned. The feature list will be updated and extended from the feedback after the initial use-case deployments as well as from the feedback received from the external users.

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

References

[1] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze, "Cloud federation," *CLOUD Comput.*, vol. 2011, pp. 32–38, 2011.

[2] N. Grozev and R. Buyya, "Inter-Cloud architectures and application brokering: taxonomy and survey," *Softw. Pract. Exp.*, vol. 44, no. 3, pp. 369–390, Mar. 2014.

[3] Y. Verginadis *et al.*, "D2.1 System Specification." The Melodic H2020 Project Deliverable D2.1, 2017.

[4] T. Kirkham and K. Jeffery, "D1.6.2 Final Architecture Design." The PaaSage Project Deliverable D1.6.2, 2016.

[5] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "How to Enhance Cloud Architectures to Enable Cross-Federation," in *2010 IEEE 3rd International Conference on Cloud Computing*, 2010, pp. 337–345.

[6] D. C. Schmidt, "Model-driven engineering," *Comput.-IEEE Comput. Soc.-*, vol. 39, no. 2, p. 25, 2006.

[7] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, Jan. 2003.

[8] A. Computing and others, "An architectural blueprint for autonomic computing," *IBM White Pap.*, vol. 31, 2006.

[9] H. V. Jagadish *et al.*, "Big Data and Its Technical Challenges," *Commun ACM*, vol. 57, no. 7, pp. 86–94, Jul. 2014.

[10] A. Celesti and P. Leitner, *Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC 2015, Taormina, Italy, September 15-17, 2015, Revised Selected Papers*. Springer, 2016.

[11] D. Chappell, *Enterprise Service Bus*. O'Reilly Media, Inc., 2004.

[12] J. Sutherland and W.-J. van den Heuvel, "Enterprise Application Integration and Complex Adaptive Systems," *Commun ACM*, vol. 45, no. 10, pp. 59–64, Oct. 2002.

[13] P. Hintjens, *ZeroMQ: Messaging for Many Applications*. O'Reilly Media, Inc., 2013.

[14] D. Palma and T. Spatzier, "Topology and orchestration specification for cloud applications (TOSCA)," *Organ. Adv. Struct. Inf. Stand. OASIS Tech Rep*, 2013.

[15] G. Blair, N. Bencomo, and R. B. France, "Models@ run.time," *Computer*, vol. 42, no. 10, pp. 22–27, Oct. 2009.

[16] Y. Verginadis, I. Patiniotakis, C. Halaris, G. Mentzas, K. Kritikos, and K. Jeffery, "D2.4 Metadata Schema." The Melodic H2020 Project Deliverable D2.4, 2017.

[17] F. Rossi, P. van Beek, and T. Walsh, *Handbook of Constraint Programming*. Elsevier, 2006.

[18] Center for History and New Media, "Zotero Quick Start Guide." [Online]. Available: http://zotero.org/support/quick_start_guide.

[19] Jeffrey O. Kephart and Rajarshi Das, "Achieving Self-Management via Utility Functions," *IEEE Internet Comput.*, vol. 11, no. 1, pp. 40–48, Jan. 2007.

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

[20] Brice Morin, Olivier Barais, Jean-Marc Jézéquel, Franck Fleurey, and Arnor Solberg, “Models@run.time to Support Dynamic Adaptation,” *Computer*, vol. 42, no. 10, pp. 44–51, 2009.
[21] Jacqueline Floch *et al.*, “Playing MUSIC — building context-aware and self-adaptive mobile applications,” *Softw. Pract. Exp.*, vol. 43, no. 3, pp. 359–388, Mar. 2013.
[22] Witold Pedrycz, Petr Ekel, and Roberta Parreiras, *Fuzzy Multicriteria Decision-Making: Models, Methods and Applications*. Wiley, 2010.
[23] Mounir Beggas, Lionel Médini, Frederique Laforest, and Mohamed Tayeb Laskri, “Fuzzy Logic Based Utility Function for Context-Aware Adaptation Planning,” in *Modeling Approaches and Algorithms for Advanced Computer Applications: Proceedings of the 4th International Conference on Computer Science and Its Applications (CIIA 2013)*, Conference Location: Saida, Algeria, 2013, vol. 488, pp. 227–236.
[24] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya, “CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms,” *Softw. — Pract. Exp.*, vol. 41, no. 1, pp. 23–50, Jan. 2011.
[25] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.
[26] Geir Horn, “A vision for a stochastic reasoner for autonomic cloud deployment,” in *Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies (NordiCloud 2013)*, Conference Location: Oslo, Norway, 2013, pp. 46–53.
[27] Mandayam A. L. Thathachar and P. S. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization*, 1st ed. Boston, MA, USA: Kluwer Academic, 2004.
[28] M. J. D. Powell, “The BOBYQA algorithm for bound constrained optimization without derivatives,” Cambridge University, England, UK, Department of Applied Mathematics and Theoretical Physics, Centre for Mathematical Sciences, DAMTP 2009/NA06, Aug. 2009.
[29] C. L. Philip Chen and C.-Y. Zhang, “Data-intensive applications, challenges, techniques and technologies: A survey on Big Data,” *Inf. Sci.*, vol. 275, no. Supplement C, pp. 314–347, Aug. 2014.
[30] D. Yuan, Y. Yang, X. Liu, and J. Chen, “A data placement strategy in scientific cloud workflows,” *Future Gener. Comput. Syst.*, vol. 26, no. 8, pp. 1200–1214, Oct. 2010.
[31] A. Buchmann and B. Koldehofe, “Complex Event Processing,” *It - Inf. Technol. Methoden Innov. Anwendungen Inform. Informationstechnik*, vol. 51, no. 5, pp. 241–242, 2009.
[32] J. Domaschka, F. Griesinger, D. Baur, and A. Rossini, “Beyond Mere Application Structure Thoughts on the Future of Cloud Orchestration Tools,” *Procedia Comput. Sci.*, vol. 68, no. Supplement C, pp. 151–162, Jan. 2015.
[33] J. Domaschka, D. Baur, D. Seybold, and F. Griesinger, “Cloudiator: a cross-cloud,

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

multi-tenant deployment and runtime engine," in *9th Symposium and Summer School on Service-Oriented Computing*, 2015.

[34] D. Baur and J. Domaschka, "Experiences from Building a Cross-cloud Orchestration Tool," in *Proceedings of the 3rd Workshop on CrossCloud Infrastructures & Platforms*, New York, NY, USA, 2016, p. 4:1–4:6.

[35] E. Rissanen, *extensible access control markup language (xacml) version 3.0*. January, 2013.

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