MORPHEMIC

Architecture of pre-processor and proactive reconfiguration

Modelling and Orchestrating heterogeneous Resources and Polymorphic applications for Holistic Execution and adaptation of Models In the Cloud

H2020-ICT-2018-2020
Leadership in Enabling and Industrial Technologies: Information and Communication Technologies

Grant Agreement Number
871643

Duration
1 January 2020 –
31 December 2020

www.morphemic.cloud

Deliverable reference
D4.1

Date
19 March 2021

Responsible partner
7bulls.com

Editor
Marta Różańska

Reviewers
Nebil Ben Mabrouk, Alexandros Raikos

Distribution
Public

Availability
www.morphemic.cloud

Executive summary

This deliverable provides an initial, high-level architectural overview of the MORPHEMIC software platform and its underlying frameworks. The intention is to present and frame the high-level architecture and sequence flows of the MORPHEMIC pre-processor through three key areas:

1. The description and presentation of underlying frameworks in use by the MORPHEMIC pre-processor and their relation to the project's objectives.
2. A comprehensive analysis of the software platform's features, in relation to the project's requirements and objectives.
3. The high-level architecture of the MORPHEMIC pre-processor.

In general, this document serves as an initial point of reference for the software architecture of the MORPHEMIC platform and will provide the necessary guidelines for future work to be carried out by members of the consortium during the course of the project.

Author(s)
Marta Różańska, Katarzyna Materka, Alicja Reniewicz, Maxime Compastié, Kyriakos Kritikos, Ciro Formisano, Yiannis Verginadis, Chris Kachris, Kaïs Chaabouni, Łukasz Szymanski, Michał Semczuk, Paweł Skrzypek

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 871643

Revisions

Date	Version	Partner
15.12.2020	0.9	7bulls, UiO
19.03.2021	1.0	7bulls, UiO

Table of Contents

Revisions 2
List of Figures 4
List of Tables 4
Glossary 5
1 Introduction 7
1.1 Structure of the document 7
2 Baseline frameworks 7
2.1 MELODIC architecture 8
2.1.1 Application Modelling 8
2.1.2 Integration 8
2.1.3 Upperware 10
2.1.4 Executionware (Cloudiator) 10
2.1.5 Monitoring 10
2.1.6 Workflow 10
2.2 ProActive Scheduler architecture 12
2.2.1 Workload Scheduling 12
2.2.2 Workflow Model and Task definition 12
2.2.3 Job execution and Task Scheduling 13
2.2.4 Computing Resource Management and Monitoring 14
2.2.5 Components Integration 14
2.2.6 General ProActive Scheduler Workflow 15
2.2.7 Convergence with MELODIC Executionware 15
3 Requirements for the architecture 16
3.1 Replacing MELODIC Executionware 16
3.1.1 Requirements and decision 16
3.1.2 Analysis of replacing MELODIC Executionware 17
3.2 ProActive adaptation requirements and assumptions 21
3.2.1 Requirements 21
3.2.2 Integration within MORPHEMIC 22
3.3 Polymorphic adaptation requirements and assumptions 24

3.3.1 Requirements....24
3.3.2 Integration within MORPHEMIC....24
3.4 Hardware acceleration integration....27
3.4.1 Requirements....27
3.4.2 Integration within MORPHEMIC....28
3.5 Security requirements and assumptions....30
3.6 Self-healing requirements and assumptions....31
3.6.1 Requirements....32
3.6.2 Integration within MORPHEMIC....32
3.7 GUI for CAMEL modelling and UI extensions....34
3.7.1 Requirements....34
3.7.2 Integration within MORPHEMIC....34
3.8 Use-case application adaptation requirements....35
3.8.1 Requirements....35
4 Preliminary MORPHEMIC architecture....36
4.1 General MORPHEMIC architecture....36
4.2 High level component architecture....37
4.3 High level sequence flow....41
4.3.1 Initial deployment flow....42
4.3.2 Deployment reconfiguration flow....44
5 Summary....46
6 References....47

List of Figures

Figure 1: MELODIC high-level architecture....9
Figure 2: MELODIC high-level workflow....11
Figure 3: Implementation of control flow structures in ProActive Workflows....13
Figure 4: The architecture of the ProActive Scheduler....15
Figure 5: Integration of the MELODIC Upperware with the ProActive Scheduler, and the MORPHEMIC pre-processor....16
Figure 6: Preliminary architecture of the proactive adaptation feature in MORPHEMIC....23
Figure 7: The architecture of the polymorphic adaptation feature in the context of the MORPHEMIC pre-processor .26
Figure 8: High-level overview of the accelerated tasks deployed on the FPGA-based cloud resources using the InAccel FPGA resource manager....28
Figure 9: Offloading computationally intensive application on the FPGAs. A use case where a task is offloaded to the FPGAs through the InAccel resource manger....29
Figure 10: Core architectural elements of the MORPHEMIC self-healing capabilities feature....33
Figure 11: MORPHEMIC preliminary architecture....38
Figure 12: Key components of MORPHEMIC....39
Figure 13: MORHPEMIC Architecture sequence - Initial deployment....43
Figure 14: Architecture sequence - Reconfiguration....45

List of Tables

Table 1: Technical comparison between Activeeon's ProActive Scheduler and Cloudiator....18
Table 2: Security requirements....31
Table 3: Key components of the MORPHEMIC pre-processor and proactive reconfiguration high-level architecture.40

Glossary

Acronyms	Definition
ABAC	Attribute Based Access Control
API	Application Programming Interface
(S1) AP	(S1) Application Protocol
BPM	Business Process Management
CAMEL	Cloud Application Modelling and Execution Language
CP	Constraint Programming
CPU	Central Processing Unit
DoW	Description of Work
EMS	Event Management System
ESB	Enterprise Service Bus
FPGA	Field Programmable Gate Array
GDPR	General Data Protection Regulation
GPU	Graphic Processing Unit
GUI	Graphical User Interface
HPC	High Performance Computing
IP	Internet Protocol
MCTS	Monte Carlo Tree Search
PAMR	ProActive Message Routing
PNP	ProActive Network Protocol
PT	Parallel Tempering
RAM	Random Access Memory
RAN	Radio Access Network
SDK	Software Development Kit
SDN	Software Defined Network

SLO	Service Level Objective
TLS	Transport Layer Security
UI	User Interface
UML	Unified Modelling Language
VM	Virtual Machine

1 Introduction

This document provides an initial, high-level architectural overview of the MORPHEMIC software platform and its underlying frameworks. The detailed architecture of each components and interfaces between them will be provided in respective deliverables as described in section 4.1. The most important objective of the MORPHEMIC project is to provide two types of advanced adaptation of cloud applications: proactive adaptation and polymorphic adaptation. The goal of the proactive adaptation (also known as proactive reconfiguration) feature is to provide the ability to optimize the application's deployment in a given time horizon in the future. This means that the application will be optimized not only for the current execution context, but also for the future execution conditions. The goal of the polymorphic adaptation is to provide the ability to optimise the (multi-cloud) application's deployment by changing the application architecture at runtime. In particular, as a runtime decision, MORPHEMIC should decide which form or configuration of a component should be selected: virtual machine (direct deployment in a VM), container (deployment via container within a VM), serverless (container-based deployment of functions), hardware accelerators, etc. The selection will be based on the value of the utility function per each application architecture variant as well as the hard application requirements (like the SLOs).
To implement the features presented, the MORPHEMIC platform is developed based on two key pillars: (i) already existing frameworks that will be integrated and reused (MELODIC and ProActive Scheduler), and (ii) new components created from scratch. Members of the MORPHEMIC consortium have also decided to use the above terms for the described features, as the term "proactive reconfiguration" may suggest that all reconfiguration has already been conducted.

1.1 Structure of the document

In this document, despite its title, not only proactive adaptation is covered, but also the complete high-level architecture of the MORPHEMIC platform which contains integrated MORPHEMIC pre-processor and baseline frameworks. The intention is to present and frame the high-level architecture and sequence flow in MORPHEMIC. What is more, technical requirements and assumptions for each key feature of MORPHEMIC are described. Subsequently, the overall high-level architecture of the MORPHEMIC pre-processor and integration with the underlying frameworks is presented. The architecture is presented in the form of diagrams, which show the interaction between the key high-level components. Also, interaction with underlying frameworks is presented; assuming the communication between the MORPHEMIC and these frameworks is done through exposed interfaces.
The document contains the following sections which are intended for the target audience:

- Baseline frameworks – in this section the baseline frameworks used in the MORPHEMIC project are listed and described. A brief scope of the feature and an architecture over-view are presented for each framework. This section should be read by all technical partners in the project. A brief scope of the features and an architecture overview are presented for each framework.
- Requirements for the architecture – in this section the requirements for the architecture are presented by each feature of the MORPHEMIC pre-processor. A brief description is provided, following general architecture requirements. The details of the low-level architecture of each feature and component are not exposed, only the outside interfaces are listed and presented. This section should be read by technical and use case partners.
- Preliminary MORPHEMIC architecture – this section presents the initial MORPHEMIC architecture at a high level. The purpose is to frame the MORPHEMIC Pre-processor using the top-down approach. The detailed architecture of each feature will be described in the respective deliverables listed in this section. The overall architecture diagram and flow (sequence) diagram are presented. This section should be read by all partners in the project. It will also be interesting for the persons outside of the project who are interested in using or developing the MORPHEMIC platform.
- Summary – this section summarizes the deliverable and also draws directions for further work.

2 Baseline frameworks

In this section, the architecture of the baseline frameworks (MELODIC and ProActive Scheduler) in use by the MORPHEMIC platform is described. The technical requirements for the MORPHEMIC architecture are also presented. The technical requirements have been collected based on the following sources:

- The initial requirements and objectives of the MORPHEMIC platform as detailed in the project's Description of Action.

- The requirements of the use case providers as detailed in the deliverable D6.1 "Industrial requirements" [15] analysis. These requirements have been analysed and used as a foundation for the definition of the architecture.
- The decision regarding the replacement of the pre-existing Executionware framework (Cloudiator) from the MELODIC project with the Activeeon Proactive Scheduler component, which was agreed upon by consortium members of the MORPHEMIC project during the June 2020 plenary meeting. The details of this decision are presented in the relevant section 3.1.

2.1 MELODIC architecture

The MELODIC platform[1] is the software foundation of the MORPHEMIC project. In this section, the overall architecture of the MELODIC platform is presented. The MELODIC platform was developed under the scope of the H2020 MELODIC project[2]. This is a complete cross-cloud, cloud-agnostic deployment platform with advanced, utility-based optimization and multi-layer, real time monitoring features. This platform exploits the modelling language CAMEL [8], which advances the state-of-the-art through its ability to cover all aspects relevant to the application lifecycle in a rich manner, while catering for the models@run.time [10] paradigm at the deployment and monitoring aspects, enabling the adaptive provisioning of cloud applications. MELODIC also features an advanced Event Management System (EMS), which is able to monitor the cloud application across different clouds and abstraction levels, thus providing the right monitoring feedback for the continuous optimisation loop. The latter is realised through advanced optimisation using various methods of parallel tempering [12], Monte Carlo Tree Search Methods [13] and integration methods [14] in conjunction with constraint programming models.

2.1.1 Application Modelling

The application in MELODIC is modelled in the CAMEL modelling language [8], which can be interpreted as a superset of the Topology and Orchestration Specification for Cloud Applications (TOSCA) [9] standard from The Organization for the Advancement of Structured Information Standards (OASIS).
All components of the MELODIC platform operate on a common application model. MELODIC can therefore be seen as a particular implementation of the models@run.time part of a cloud modelling framework. The model itself is cloud agnostic and not dedicated to any particular cloud provider. The model of the application is translated to a constraint programming model (CP Model), which allows for the mathematical optimisation of the deployment, also in a cloud-agnostic way.

2.1.2 Integration

MELODIC integrates underlying frameworks into one platform [8]. Different frameworks use different integration methods, both in tools used and in communication types – synchronous versus asynchronous. The proper selection of the integration architecture is a crucial point for the success of MELODIC. An additional element to consider was the level of effort needed to implement the chosen integration method. ESB with BPM orchestration was chosen as the most flexible and easy method of integration [18]. ESB is a common integration method used to integrate enterprise grade systems. BPM is a standard for the description and execution of business processes. The integration layer of MELODIC contains two planes:

- Control Plane – for business logic integration and controlling.
- Monitoring Plane – for monitoring related activities.

The key benefits of this approach are:

- Flexible logic implementation in the BPM flow with no hard coding.
- Support for both synchronous and asynchronous communication.
- Support for most of the integration protocols.
- Reliability, configuration easiness, and high availability.
- Ability to integrate with other enterprise applications due to the use of the ESB integration method

The architecture of the MELODIC software platform is shown in Figure 1.

[1] https://www.melodic.cloud/
[2] https://h2020.melodic.cloud/

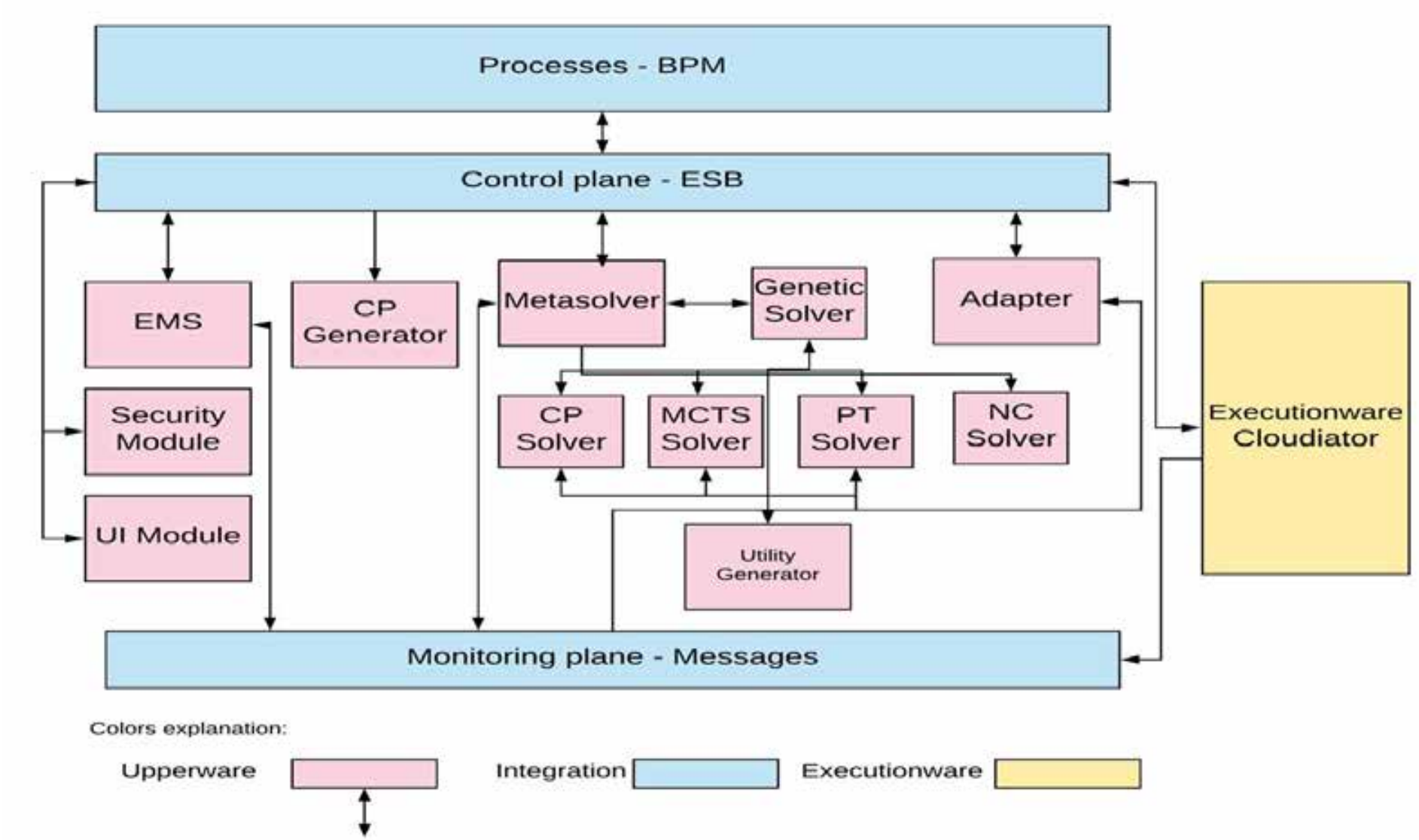


Figure 1: MELODIC high-level architecture

2.1.3 Upperware

2.1.3.1 Application profiling

The profiling part of the MELODIC platform handles the CAMEL model and prepares the CP Model for the reasoning process. It is implemented by the CP Generator.

2.1.3.2 Reasoning

The reasoning process is the core part of the MELODIC platform. It is responsible for finding the best deployment solution for the given application utility within the given constraints and current application context. The key components involved in the reasoning part are the following:

- The Metasolver, which chooses the right solver for a given problem.
- The CP Solver, which is a solver dedicated to solving nonlinear constraint problems.
- The Genetic algorithm-based [11] solver uses genetic algorithm to solve optimization problems.
- The Parallel Tempering (PT) [12] based solver, which uses the Parallel Tempering method to solve optimization problems.
- The Monte Carlo Tree Search (MCTS) [13] based solver, which uses advanced implementation of the MCTS algorithm to solve optimization problems.
- The Utility Generator which is the module responsible for the calculation of the utility value per feasible application configuration candidate proposed by a solver.

2.1.3.3 Adaptation

The adaptation part deals with the adaptation and orchestration of the deployment model to be deployed across the chosen Cloud providers. It contains one key component, the Adapter, which translates the CP model optimal solution to a CAMEL deployment model, then prepares, and finally orchestrates the plan of the deployment.

2.1.4 Executionware (Cloudiator)

The Executionware executes deployment commands orchestrated by the Adapter. It includes enacting the reconfiguration commands that the adapter has devised, in order to move data and components, and to ensure that the components are connected after the application is reconfigured.

2.1.5 Monitoring

The monitoring module of MELODIC collects the measurements on the current execution context using a hierarchical Esper event processing framework for gathering and aggregating the contextual metric values.

2.1.6 Workflow

The MELODIC workflow of actions is depicted in Figure 2:

1. The CAMEL model of the application and its requirements is prepared by the user.
2. The Constraint Programming (CP) model for the reasoning part is prepared based on the CAMEL model. The CP Model is enriched with the actual real-time metric values in case of reconfiguration. The most suitable solver for the CP model prepared is chosen by the Metasolver. The solver solves the problem. Each application configuration candidate is evaluated by the Utility Generator.
3. The most appropriate solution is communicated to the Adapter to be deployed. The solution and monitoring framework is deployed by the Adapter using the Executionware.
4. The monitoring component starts collecting metrics of the deployed solution.
5. Based on real-time metric values, the Metasolver decides if an application reconfiguration should be initiated.
6. If the process of reconfiguration is started, then the Metasolver updates the CP model with the current values of collected metrics, and the solver starts to look for the optimal solution. As before, the utility function value is calculated to evaluate and compare each application configuration candidate. After that that the deployment is repeated from step 3.

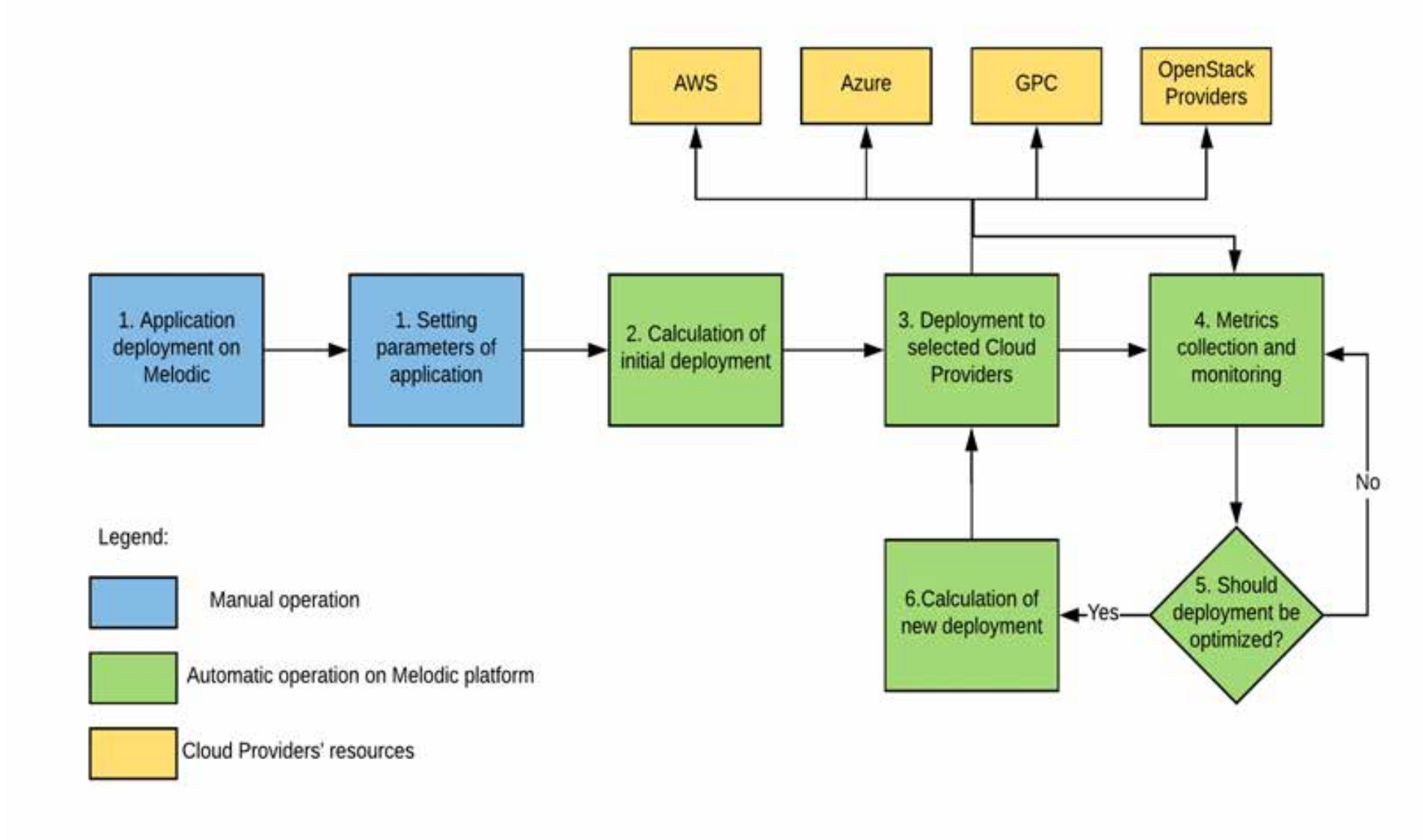


Figure 2: MELODIC high-level workflow

2.2 ProActive Scheduler architecture

In the MORPHEMIC project's description of action, the Activeeon ProActive Scheduler[3] is solely considered as part of the MORPHEMIC platform to support the pre-processor runtime: It minimizes the resource usage at runtime while taking into account the decisions made by the Upperware for adapting deployments. Its contribution to the runtime will be extensively detailed in deliverable D5.5 "Runtime optimization mechanism". In this subsection, we detail the internal architecture of the ProActive Scheduler. Its integration as Executionware with MORPHEMIC is covered in section 3.

2.2.1 Workload Scheduling

The workload scheduling involves the control and the dispatch of the execution of processing over a finite set of computing resources: this distribution aligns with high-level objectives, such as the minimization of the execution duration of the processing. However, the workload scheduling also accounts for the specification of computing resources. In the ProActive Scheduler, such processing is referred to as a scheduler Job.
A workload is a composition of atomic processes that can be allocated individually on resources. They can require the prior execution of other processing and have requirements over the computing resources they are allocated to. For instance, a process could be a computation to involving the result of another. In the ProActive scheduler, workloads are specified as workflows composed of tasks.
In ProActive terminology, a node models the capability of a computing resource to receive processing to execute in situ. Multiple nodes can be bound to the same resource if it supports multi-processing. In ProActive, nodes are declared by an agent on the computing resources and inform about the current usage of the resources.
In the next subsection, we overview how these concepts are defined and implemented by the Activeeon ProActive Scheduler.

2.2.2 Workflow Model and Task definition

In ProActive, a workload requiring scheduling is defined as a workflow model. It is an oriented graph composed of tasks (apexes) representing atomic processing and execution control flows (edges).
The file format specification of the workflow model is extensively defined in an XSD schema file[4]. Each task of the workflow is specified based on the following sets of properties:

- A task name description used to identify the processing at instantiation stage.
- The maps of variables specific to the task.
- The implementation of the processing: This can either be under the form of a Java class, or plain source code to be submitted to a language interpreter supported by ProActive.
- The definition of files to be input or output, respectively, before or after the task execution.
- A set of properties to locally constrain the execution of the task:
    - The handling strategy of errors during task execution (i.e., interrupt or cancel the whole workflow, interrupt/cancel only depending tasks, retry tasks, ignore errors).
    - The definition of custom logic for node selection.
    - The simultaneous execution on multiple nodes.
    - The isolation environment in which the task is executed, i.e., none, a dedicated Java virtual machine, software containers, or custom user implementations.
    - The generic information field, to locally constrain the execution of a task by exploiting a dedicated ontology (e.g., impose the execution of a task on a subset of nodes, delaying the execution at a given time).

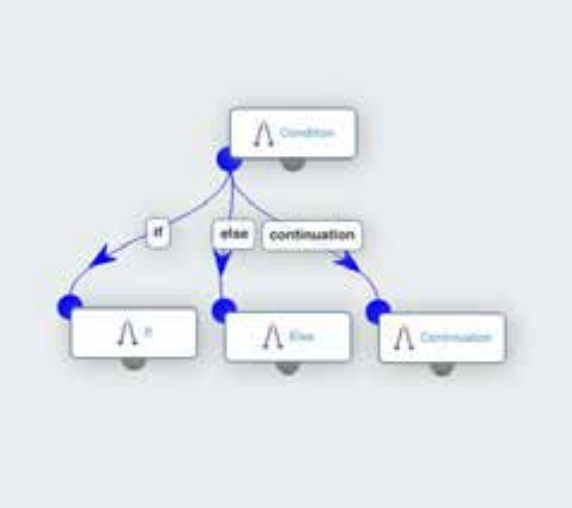
A ProActive workflow structures the execution of these tasks by defining dependency relations inspired by imperative programming:

- Sequential dependency − a set of tasks can be required to be run priorly to or following a task.
- Conditional branching − based on a condition, a set of tasks may have their execution confirmed or ignored.
- Iterative branching − the execution of a set of tasks is iterated after the positive re-evaluation of conditional statement.
- Replicating branching − a group of tasks can be scheduled for execution multiple times in parallel after a value is evaluated once.

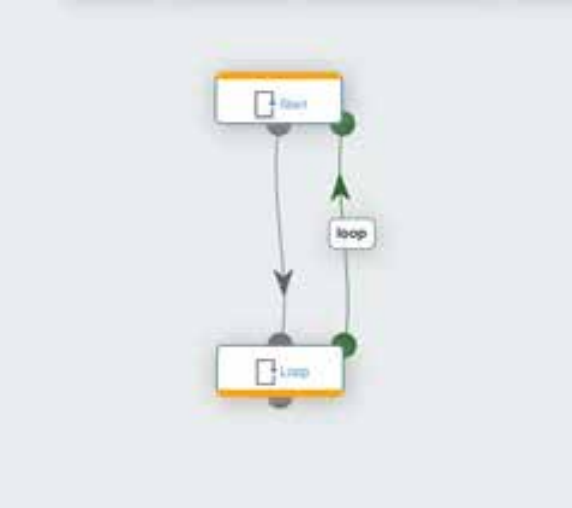
Figure 3 illustrates the control flow structures in ProActive workflows.

[3] https://www.activeeon.com/
[4] ProActive Workflow XSD specification. https://www.activeeon.com/publiccontent/schemas/proactive/jobdescriptor/3.3/schedulerjob.xsd (accessed Oct. 25, 2020)

a/ Sequential dependency



b/ Conditional branching



c/ Iterative branching

d/ Replicating branching

Figure 3: Implementation of control flow structures in ProActive Workflows

In Activeeon ProActive suite, a workflow and the tasks in it can be elaborated through the studio web interface or using ProActive Java APIs and SDKs.

2.2.3 Job execution and Task Scheduling

Once a workload has been fully specified as a workflow, it can be submitted to the scheduler for execution. Each submission is referred to as a "Job" in ProActive taxonomy. The Jobs can be managed independently and follow their own lifecycles.

Specifically, a Job can be paused, resumed, or killed from the Scheduler. A Job can be in the following states, once submitted:

- Pending − The scheduler is waiting for one compatible node to be available to dispatch a task of the Job.
- Running − The Job is currently having at least one of its tasks being executed on a node.
- Finished − The Job has all its tasks completed.
- Paused − The execution of the Job has been temporarily suspended from the scheduler.
- Killed − The Job has been terminated by the scheduler.
- In-Error − At least one task has caused a non-ignored error, terminating the Job execution.
- Failed − An internal error occurred in the scheduler during Job processing.

This list has been simplified for the sake of clarity. The complete list of Job states can be found in ProActive technical documentation[5].

[5] Activeeon, "ProActive Workflows & Scheduling — User Guide."
https://doc.activeeon.com/11.0.0/user/ProActiveUserGuide.html#_workflow_execution_control

Similarly, Jobs are defined with the same granularity as workflows: each workflow task execution can be monitored independently. Since the scheduler policy is in charge of managing the lifecycle of workflow tasks, no direct control is exposed on them. A task can be in the following states during its execution:

- Pending − The task has not been scheduled yet.
- Running − The task has been scheduled and is currently being run on a node.
- Finished − The task has successfully been scheduled and executed.
- Faulty − The task implementation has caused an error during task execution without interrupting the whole Job.
- In-Error − The task implementation has caused an error during task execution, terminating the whole Job.
- Aborted − The Job related to the task has been killed before the task was scheduled for execution.
- Skipped − The scheduler has skipped the allocation of a task. This behaviour occurs when the control flow evaluation dismisses the execution of tasks.
- Several policies can modulate the scheduling of Job's tasks:
- Regular policy − This is the default scheduling policy. Each task is scheduled on the first available node matching their characteristics in a FIFO (First-In, First-Out) fashion.
- Earliest deadline first (EDF) policy − The tasks can be assigned deadlines. The scheduler will prioritize the execution of Jobs having tasks with deadlines.
- Licence policy − The scheduler can be configured to limit the parallel execution of tasks using a specified software. In practice, this policy permits to ensure that the number of concurrent software instances will not exceed the maximum allowed by a licence.

The control and the monitoring of Job execution can be performed through a dedicated web and command-line interface. ProActive Scheduler also exposes a set of Java and REST interfaces to integrate with other software.

2.2.4 Computing Resource Management and Monitoring

The ProActive Scheduler assesses the status of nodes to select those available for task execution. The ProActive Resource Manager is responsible to provide such information and to provision additional nodes. Accordingly, the node lifecycle is composed of seven states:

- Deploying − The Resource Manager is deploying a new node on the infrastructure.
- Lost − The node deployment has failed, and the node will not be usable.
- Configuring − a node has been successfully deployed and is being integrated to the scheduler.
- Free − the node is available for receiving a task to process.
- Busy − the node is allocated a task for processing and is not available to receive another.
- To be removed − the node is currently allocated a task but will be removed when the task is fulfilled.
- Down − the deployed node is no more reachable by the Resource Manager and is no more usable.

In practice, a ProActive node agent is deployed on infrastructures to execute tasks and monitor its execution, as well as the underlying infrastructure. The acquisition or the removal of nodes is dealt with by a connector deploying or removing this agent. This is accompanied by VM management operations on IaaS infrastructures. The ProActive Resource Manager features connectors for remote hosts and for major public and private cloud providers.
A NodeSource defines how the Resource Manager can acquire and release nodes. It contains the setting to interface with an infrastructure (i.e., infrastructure type, credentials) and a policy to control node management and access.
The Resource Manager can be controlled via web based and CLI user interfaces. It also exposes a set of Java and REST APIs to integrate with other software.

2.2.5 Components Integration

The components of ProActive are integrated into a microservice architecture. They interact through REST API calls. ProActive agents are located on remote infrastructure. They interact with the Resource Manager using the ProActive Network Protocol (PNP) and the ProActive Message Routing Protocol (PAMR).
Figure 4 exposes an overview of the Community version of ProActive. We mention the user interface of the Scheduler and the Resource Manager but omit the workflow design interface (ProActive Studio) for the sake of clarity. The enterprise version comes with additional components not used in the MORPHEMIC project, and therefore not covered in this deliverable.

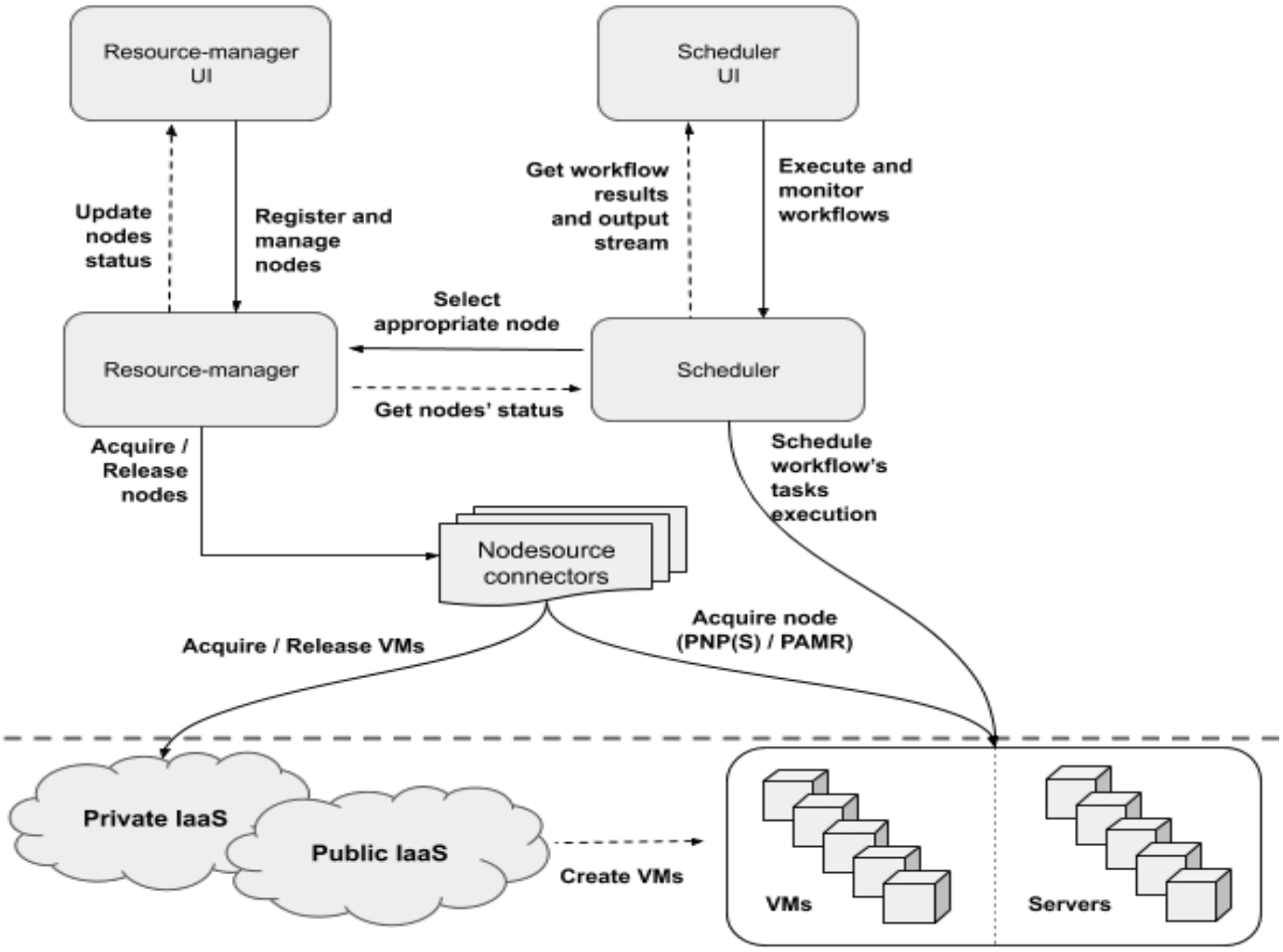


Figure 4: The architecture of the ProActive Scheduler

2.2.6 General ProActive Scheduler Workflow

To illustrate how ProActive's components cooperate to fulfil a workload scheduling, we detail the typical usage scenario of ProActive:

1. A user designs and submits a workflow file describing a workload via the scheduling interface.
2. The Scheduler interprets the workflow file into a Job and selects the first tasks to dispatch.
3. The Scheduler retrieves the nodes status.
4. Based on task characteristics, the scheduler selects the appropriate node to run the task and mark it as busy. Other tasks coming from the same or from other Jobs enter in pending state if they are dispatched to a busy node.
5. Based on a NodeSource policy, the Resource Manager requests the acquisition of additional nodes.
6. The NodeSource connector interfaces with an IaaS provider to acquire new VMs and deploy ProActive agents in there.
7. The agents report to the Resource Manager.
8. The scheduler gets the status of the new nodes and dispatches the remainder of the pending tasks to them.
9. The scheduler finishes the Job execution.
10. In compliance with the NodeSource policy, the Resource Manager requests the release of nodes
11. The NodeSource connector shutdowns ProActive agents, and interfaces with IaaS providers to remove unused VMs.

2.2.7 Convergence with MELODIC Executionware

The above subsections have shed the light on shared features between the ProActive Scheduler and Cloudiator. First, the workflow model can be used to orderly define the deployment plan of an application. The integration with the Adapter is a considerable option thanks to the API provided by ProActive. Moreover, the scheduler is also a viable

enabler for deploying applications in distributed environments with a per-component granularity. Finally, the Resource Manager provides more than monitoring capabilities, and permits the management of infrastructure resources (i.e., public cloud infrastructure, edge nodes).
Therefore, the ProActive Scheduler is reasonably equipped to enforce the decisions made by the MORPHEMIC reasoning process on the management of infrastructure resources and application components. In the next section, we evaluate its integration as the MORPHEMIC Executionware.

3 Requirements for the architecture

This section briefly describes all of the architecture requirements that were gathered during the first months of the MORPHEMIC project. The following subsections present the proposed features of the MORPHEMIC platform, such as the polymorphic and proactive adaptation capabilities, while also expanding upon their predetermined requirements and objectives.
It is important to highlight that the MORPHEMIC pre-processor aims to build upon and further extend the software component architecture that was first introduced with the MELODIC platform. At most, minimal changes are expected to take place in the original reasoning process introduced by the MELODIC platform, and the MORPHEMIC pre-processor will be integrated in a flexible way. Figure 5 presents the proposed integration of the MELODIC Upperware with the ProActive Scheduler, and the MORPHEMIC pre-processor.

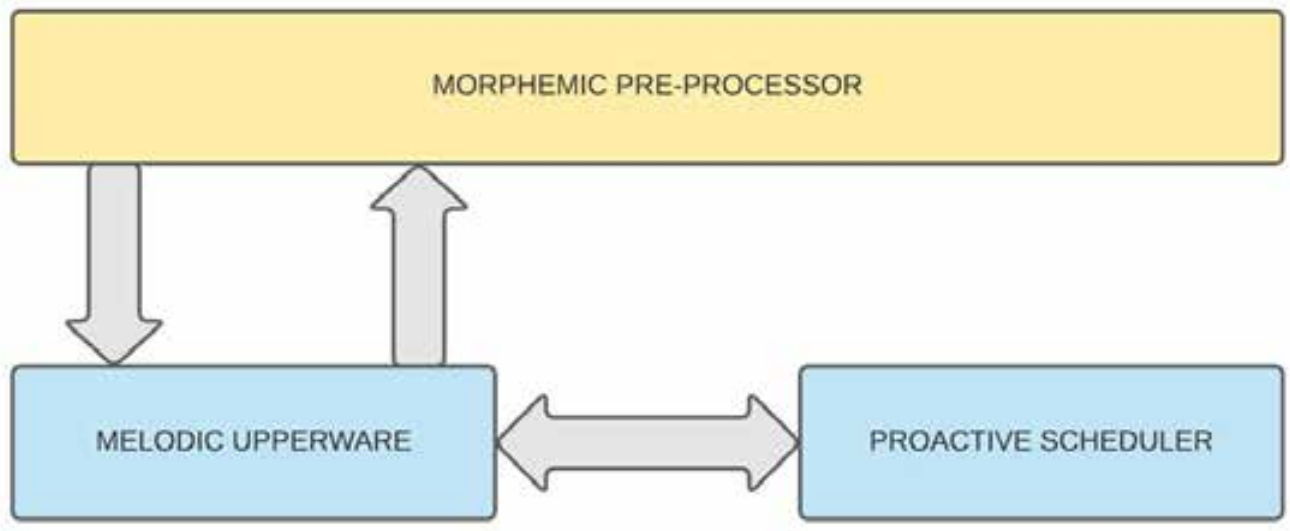


Figure 5: Integration of the MELODIC Upperware with the ProActive Scheduler, and the MORPHEMIC pre-processor

3.1 Replacing MELODIC Executionware

During early preparatory work on the architecture of the MORPHEMIC platform, many features were found to be common between Cloudiator and the ProActive Scheduler by Activeeon; both solutions provide scheduling and resource management capabilities supporting the planification and execution of the reconfiguration of distributed applications. Both solutions also participate in the: (i) provisioning and deallocation of infrastructure resources and, (ii) application component deployment, configuration and removal. Therefore, at an early stage of the MORPHEMIC project, the option of replacing the Executionware was eventually determined.
Next, we describe the reasoning, the process and the final decision for replacing Cloudiator by Activeeon ProActive Scheduler as the Executionware part of the MORPHEMIC platform.

3.1.1 Requirements and decision

During the first months of the MORPHEMIC project, a detailed evaluation of the ProActive Scheduler's capabilities in comparison with the cloud deployment-related requirements of the platform was carried out. A feature comparison of the ProActive Scheduler and Cloudiator has been presented during the consortium member's second plenary meeting (virtual). The ProActive Scheduler is a mature and commercially available product with full support provided by Activeeon. It fulfils the requirements for replacing the Executionware and it was decided that the replacement will be done within the project's scope. Activeeon is responsible for the changes on the ProActive Scheduler, and 7bulls will make the necessary modifications to the MELODIC Upperware. The usage of ProActive Scheduler with

MORPHEMIC will be possible based on its open-source license. The particular licenses for components will be described in D8.3 Final IPR and exploitation plan.
Replacing the Cloudiator Executionware is an invasive operation that jeopardizes the operability of the platform inherited from MELODIC, while its differences with the ProActive Scheduler requires significant integration work to devise. Therefore, the MORPHEMIC consortium has carefully reasoned on the properties and capabilities of the ProActive Scheduler and has weighed the benefits and the cost of this work.
Six main requirements have pleaded for the replacement of the Executionware:

1. Coping with distribution complexity through the consolidation of Executionware actions: The specification of reconfiguration processes as workflows permit to constrain the adaptation process solely through the interdependencies of ProActive tasks. This grants the scheduler additional options to consolidate complex reconfiguration actions. In the context of a multi-cloud and multi-tenant platform conducting multiple reconfiguration actions at once could have a positive impact on performance.
2. Operating HPC infrastructures: The support of High Performance Computing (HPC) platforms is missing in MELODIC Executionware. ProActive is an Open-Source scheduler supporting HPC infrastructures, bringing them at hands of the MORPHEMIC polymorphic adaptation. This increases MORPHEMIC customer base by supporting client application from the HPC sector.
3. Limiting the resource consumption in the operational environment: In accordance with the project's description of action, ProActive Scheduler will be used as the MORPHEMIC pre-processor runtime to limit the resource consumption of the deployment during adaptation determined by adaptive provisioning. Using ProActive for resource management permits to reinforce the vertical integration of MORPHEMIC platform. This permits us to avoid redundancy in the software stack of the MORPHEMIC platform, and to optimize the execution of adaptation actions by integrating them into scheduling. The number of resources required will lowered as the number of containers will be limited from 16 in Cloudiator to 2 used by ProActive Scheduler.
4. Enhancing self-healing application monitoring: The Resource Manager features the monitoring of infrastructures operated by ProActive nodes. This contributes to the self-healing feature by aiding the EMS in the identification of the cause of a failure. Additionally, the ProActive workflow format is flexible enough to enable the scheduler to interact with the EMS during application deployment. This tightens the integration of the Executionware with the EMS as it will provide a precise guidance to the EMS for the deployment of monitoring probes and monitoring infrastructure under ProActive's instruction.
5. Enabling further control of the reconfiguration process due to polymorphic adaptation: The polymorphic adaptation can leverage complex reconfiguration actions to fully exploit the capabilities of an infrastructure. For instance, the operation of FPGA-enabled infrastructure will require the installation and configuration of the Accelerator Resource Manager. The implementation of reconfiguration actions as ProActive Scheduler jobs and their segmentation as tasks permits finer control and reporting of the reconfiguration process.
6. Expanding support for exploitation activities: The ProActive Scheduler is developed as a core product of Activeeon. This includes new opportunities regarding the exploitation strategy of the MORPHEMIC project and its maintenance after the project's deadline. This specific benefit will be elaborated further on deliverable D8.2, the Initial IPR and Exploitation Plan.

3.1.2 Analysis of replacing MELODIC Executionware

Following the aforementioned benefits, the consortium has sought an integration strategy for the ProActive Scheduler component with the MORPHEMIC architecture. From a usage perspective, a typical user of the ProActive Scheduler defines the workload to be executed as a workflow in the graphical user interface, submits it to the scheduler for instantiation, and leaves the Resource Manager to handle infrastructure resource allocation based on the NodeSource policy configuration. The usage differs from Cloudiator as the management actions to undertake are determined by the Upperware, while Cloudiator is directly in charge of scheduling and enforcing those actions on infrastructure resources. The consortium has devised the following integration strategy in the ProActive Scheduler to enact its integrability with the existing MELODIC Upperware:

- The MELODIC Upperware still defines the management actions to be undertaken by the Executionware.
- The Scheduler will be in charge of enacting and following up on the progress of the determined management actions.
- The Resource Manager will provide the necessary tooling for enforcing management actions on cloud and edge resources while monitoring the resources in use.

The consortium has evaluated the technical differences between both Cloudiator and the ProActive Scheduler to identify the lines of work needed for adapting the latter to the specifications of the MORPHEMIC project. Table 1 summarizes the differences of features between Cloudiator and ProActive Scheduler.

Table 1: Technical comparison between Activeeon's ProActive Scheduler and Cloudiator

Supported Infrastructures	ProActive Scheduler	Cloudiator
Cloud	Azure, AWS EC2, Google Compute Engine, OpenStack Nova, VMWare	Azure, AWS EC2, Google Compute Engine, OpenStack Nova
On-premise	Yes, via SSHInfrastructure or OpenStack connector, or manual management of the agent.	BYON
Edge devices	Yes, via SSHInfrastructure, or manual management of the agent.	Yes
High Performance Computing (HPC) Infrastructure	Yes, natively via ProActive-agent package. Support for on-site HPC Scheduler (ex: PBS) in enterprise version	No
Deployed Resources/Platforms		
VM	Yes	Yes
FaaS	No	Yes
Containers	Yes	Yes
Monitoring probes	Yes	Yes
Execution/Scheduling Model		
Execution unit	Job task	Task
Execution worker/agent	ProActive Node	Process
Scheduling policy	Various scheduling policies (not limited to sequential execution)	Queues (only sequential execution)
Parallel execution	Yes	Yes (by deploying multiple nodes)
GUI	Yes	Yes
Security		
Secured support for credentials storage	Yes (third party credentials feature)	Yes
Key REST operations		

Supported Infrastructures	ProActive Scheduler	Cloudiator
Create nodes based on Node Candidates (i.e., deploy VMs)	Yes	Yes
Create Job (create workflow in Activeeon terminology)	Yes	Yes
Create schedule (submit Job)	Yes	Yes
Create process (deploy ProActive node)	Yes (actually automated during the creation of Cloud subscription, needs to be extended to allow on-demand deployment)	Yes
Interfacing with third-party components involved in polymorphic adaptation	Flexible, through workflow task design: support of several programming languages, fully integrated in deployment procedure	Yes, command execution in provisioned node
Integration with Runtime Scheduling	Native (ProActive engine is used for runtime scheduling)	External (a connector needs to be implemented)
Rich, easy to use UI to define workflow	Yes	No
Visual status of the deployment	Yes	No

The analysis has identified the following features to be incorporated in the Activeeon ProActive Scheduler to be compatible with MORPHEMIC components:

- Support for Function-as-a-Service platforms for application deployment.
- Creation of Cloud subscriptions without effective resource deployment.
- Retrieval of Node Candidates (i.e., specifications and pricing of resources available for self-service).
- Allocation of resources based on a Node Candidate specification.
- On-demand deployment of resources.
- Update of the resource type or system image.
- Resource migrations (e.g., updating of node infrastructure in Table 1).

The replacement of Cloudiator by ProActive Scheduler will allow to introduce flexible deployment of workflow applications, which is not currently supported by Cloudiator. This feature permits to better integrate components involved in polymorphic adaptation. Fulfilling this requirement will require significant reengineering of Cloudiator. Moreover, according to Task 5.4 of the Description of Action (DoA), the ProActive workflow engine will be exploited for the runtime optimisation: using ProActive Scheduler as the Executionware seamlessly integrates the resource management within the scope of optimisation. Finally, additional reason for replacing Cloudiator by ProActive Scheduler is dynamic development of ProActive Scheduler by Activeeon. The ProActive Scheduler is a flagship product of this company. The development of Cloudiator has been done by academic organizations and is no longer developed due to personal changes in the team. In addition, some of the features of Cloudiator will be adopted and reused in ProActive Scheduler.

The following subsection details the work that is planned for these features on the components of the ProActive Scheduler and the Upperware. The effective implementation will be elaborated in the deliverable D5.3 „Artifact Resource Manager".

3.1.2.1 Scope of the changes

Based on the analysis conducted in the previous subsection, the scope of the changes set in the Executionware components has been determined in order to ensure the integration of the ProActive Scheduler in the MORPHEMIC architecture. Three objectives have driven this work:

1. The support by ProActive Scheduler of infrastructure actions that were solely handled by Cloudiator.
2. The empowerment of application reconfiguration process by the capabilities brought by the specification of ProActive Workflows.
3. The modifications in Upperware components to comply with the new execution interfaces.

The following subsections address these points.

3.1.2.2 Scheduler

In accordance with the outcomes of the execution replacement analysis, the consortium has determined that no the scheduler is exempt of adaptation work: It is able to perform the same functionalities as those identified in Cloudiator.

3.1.2.3 Resource Manager and NodeSource connectors

The Resource Manager and the NodeSource-connectors components require modifications to cope with the MORPHEMIC requirements. The NodeSource-connector establishes the connection to IaaS services of cloud providers to be exploited by MORPHEMIC e.g., AWS Elastic Compute Cloud, Microsoft Azure Virtual Machines, OpenStack Nova and Google Cloud Engine. The "SSHInfrastructure" NodeSource-connector is not covered by the following modifications:

- The creation of cloud subscriptions without effective deployments implies the modification of the cloud connector to authorize the configuration of NodeSource featuring no nodes.
- The retrieval of Node Candidates involves supporting the pricing API of public cloud providers, and to devise a dedicated strategy for the others. It also requires the implementation of an API to query for Node Candidates and fetch results. The support of system images and instance types listing is also required to retrieve the Node Candidate specifications that are not negotiated with a pricing API.
- The allocation of nodes based on Node Candidate specifications implies that the cloud connectors must support the allocation of infrastructure resources based on their type and system image, along with new endpoints in Resource Manager to operate these new interfaces.
- The on-demand deployment of resources necessitates:
    - A dedicated NodeSource policy externalizing the resource allocation and deallocation to a third-party component.
    - A set of APIs on the Resource Manager to enact this third-party to invoke this feature.

To enforce the scheduling and third-party credential features in infrastructure management, the manipulation of NodeSource in the Resource Manager will be exclusively implemented through ProActive workflows.

3.1.2.4 Scheduling Abstraction Layer

The Upperware solely relies on asynchronous API calls and it does not implement a workflow-based interaction with the Executionware. On the contrary, the ProActive Scheduler and Resource Manager heavily rely on workflows that implement the defined reconfiguration actions. Therefore, it is necessary to elaborate an abstraction layer accommodating both approaches to ensure a proper integration.

The Scheduling Abstraction Layer is a component created specifically for the MORPHEMIC architecture addressing this objective: it implements the API to be used by the Upperware, it designs reconfiguration workflows accordingly, and submits the workflows to the scheduler for execution.

This component also establishes viable means for implementing the missing features from the Cloudiator Executionware to fulfil the requirements of MORPHEMIC project:

- The support for Function-as-a-Service platforms for application deployment will be achieved through a set of tasks interacting with FaaS platforms appended by the Scheduling Abstraction Layer.
- The creation of cloud subscriptions will consist in the registration of any parameter needed for the creation of NodeSource.
- The retrieval of Node Candidates is actively contributed by the Scheduling Abstraction Layer by aggregating the information obtained from the cloud API (e.g., pricing information, system images properties, instance type) into a consolidated list of Node Candidates to be exploited by the Upperware.
- The allocation of resources based on a Node Candidate specification is achieved by exposing interfaces for allocating resources based on their types and exploiting new interfaces from the Resource Manager dedicated to this purpose.

- The on-demand deployment of resources similarly requires the Scheduling Abstraction Layer to operate the new interfaces of the Resource Manager to arbitrarily request the allocation/deallocation of a resource.
- The update of resource type or system image will be actively enacted by the Scheduling Abstraction Layer by conducting the operation needed for allocating the new version of the infrastructure resource, the migration of application components, and the deallocation of the formerly used infrastructure resources.
- The resource migrations are to be implemented by the Scheduling Abstraction Layer: the former version and the new version of the infrastructure resource have, however, the same instance type and an analogue system image but are located on different cloud regions, possibly on other cloud providers.

3.1.2.5 MELODIC Upperware

There are changes that should be made in the MELODIC Upperware components, in order to replace the current Executionware and integrate with the ProActive Scheduler. These are the components which will be affected by these modifications:

- The CP Generator – getting Node Candidates in the CP Generator needs to be done by invoking a specific API offered by the Scheduling Abstraction Layer. The Client's API call (the contract) needs to be based on the currently used types (classes) and it needs to provide functionality identical to what is being carried out by Cloudiator. In essence the API call needs to obtain as input a specified set of Requirements and needs to provide a set of specific Node Candidates as output.
- The Adapter – needs to be changed in terms of creating the deployment plan and executing it by invoking a specific API offered by the ProActive Scheduler Client. The Client's API needs to be built upon the premise of a "Lazy Evaluation" approach, which means that the Adapter will be able to configure (describe) the whole deployment first, and when it's done, send it for execution. The API contract can be based on different types (classes), but it needs to conform to what data is currently available in the Adapter when using Cloudiator. What is more, there is a need to change the configuration file with a list of tools to be installed on every VM deployed using MELODIC.
- The GUI backend - deployment information retrieval needs to be modified to invoke a specific API offered by the ProActive Scheduler Client. All the required information displayed in the GUI needs to be provided by the Client.
- The integration with MuleESB - Credentials to cloud service providers should be passed to the ProActive Scheduler Client and securely kept. This is equivalent to creating a Cloud object in Cloudiator.

3.2 ProActive adaptation requirements and assumptions

The goal of the proactive adaptation feature is to provide the ability to optimize the application's deployment taking into account a future window of time. This means that the application will be optimized for the current execution context, as well as for future runtime conditions. This also allows the MORPHEMIC platform to be proactive, and to react before any SLO violation occurs instead of simply reacting to real-time monitoring data.
This goal is particularly important from a practical point of view. The reconfiguration of an application takes time, usually a couple of minutes. After that, the runtime requirements may change, which can lead to another reconfiguration. There are some critical applications for instance from the medical or networking fields that must run with minimal downtime. Thanks to this feature, the MORPHEMIC platform can proactively adapt application resources to prevent situations that are critical and ensure the expected performance of the application.

3.2.1 Requirements

The requirements are presented in a technical form and are based on the requirements provided in the MORPHEMIC project's Description of Action and the deliverable D6.1, Industrial requirements analysis.
Below is a list of technical requirements for proactive adaptation:

- The ability to predict future conditions and behaviour of applications - it is a key requirement for proactive adaptation and the purpose of that feature.
- The selection of the best deployment configuration in the given horizon in the future - it is also the key requirement for proactive adaptation and one of the purposes of the feature.
- The ability to verify the future performance of the selected deployment in conjunction with the traditional approach based on the actual values of metrics and variables.
- The ability to verify the performance of prediction models.
- Preparation of the input data and models for the MELODIC platform to make reasoning - the ability to prepare input CAMEL model to the MELODIC reasoning process.

- Minimal changes at most to the MELODIC reasoning process - input data should be prepared by the MORPHEMIC pre-processor in the format used by MELODIC input data, to minimize changes in the MELODIC platform.

3.2.2 Integration within MORPHEMIC

This subsection contains a description of the architecture of this feature and the proposed integration process within the platform.

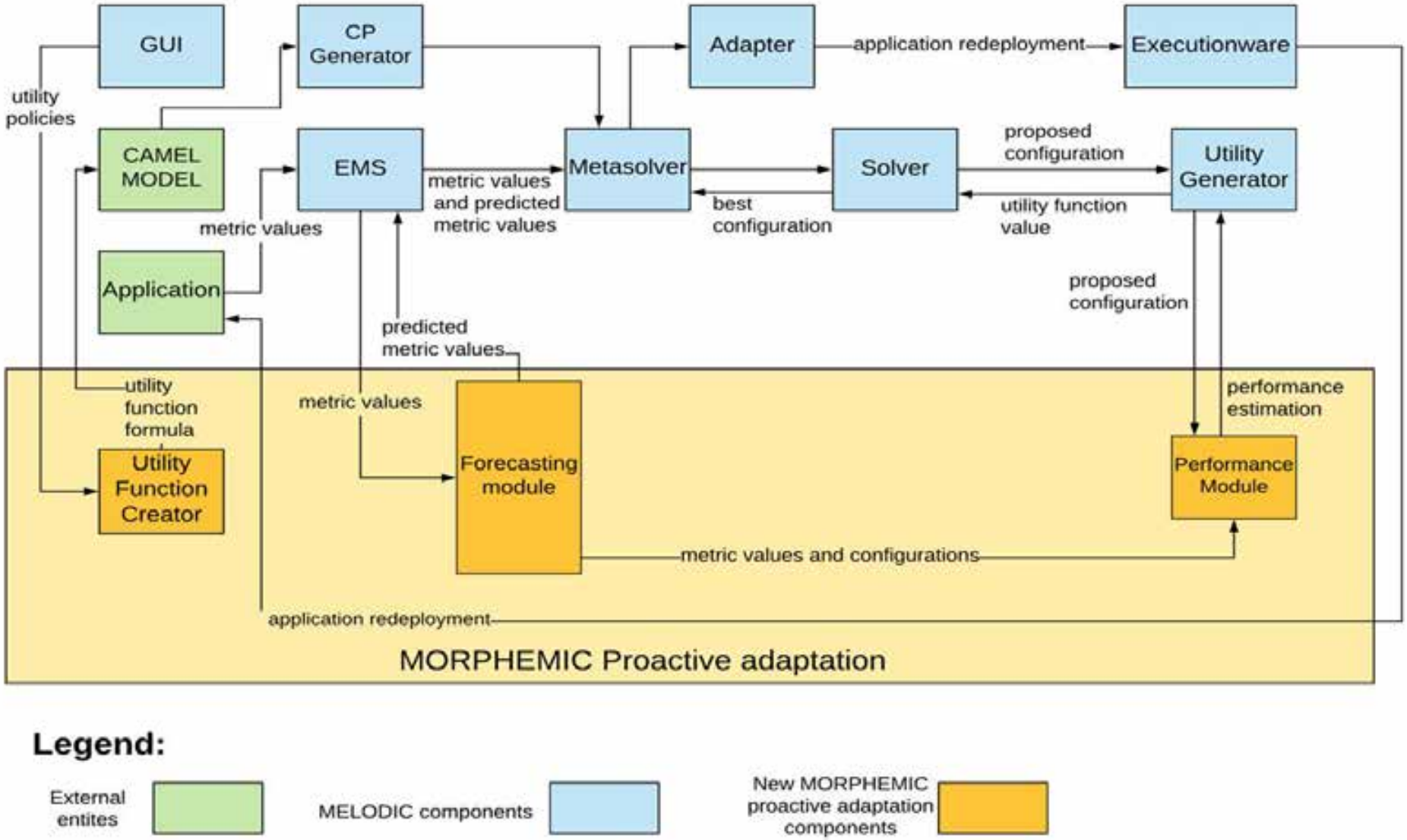


Figure 6: Preliminary architecture of the proactive adaptation feature in MORPHEMIC

Figure 6 presents the initial architecture of the proactive adaptation feature. To make the figure less distracting, only elements related to the proactive reconfiguration are illustrated. The complete architecture is presented in Section 4. The diagram, presented on Figure 6, depicts the logical architecture of the feature. The implementation will follow the fundamental MORPHEMIC architecture principles that each framework should be clearly separated by interfaces and in case of component reuse, the dedicated instances of the component will be created. New components are marked in yellow colour, the components that are already present in the MELODIC platform are marked in blue colour. Other artifacts such as the CAMEL Model and the deployed application have a red colour.
It is important to notice that the proactive adaptation can be seen as an extension of the current reasoning process. That is the reason why MELODIC Components are being extended to handle predicted metric values in addition to real-time metric values. Further, there are new components that handle the collection of metric data, forecasting, and performance estimation. Another new component that will be developed in the scope of this feature is called the Utility Function Creator and it aims to deliver the utility function formula from high-level utility policies defined by a user.

3.3 Polymorphic adaptation requirements and assumptions

The goal of the polymorphic adaptation is to provide the ability to optimise the (multi-cloud) application's deployment by repurposing the application architecture at runtime. In particular, as a runtime decision, the MORPHEMIC platform should decide which form or configuration of a component should be selected: as a virtual machine (direct deployment in a VM), as a container (deployment via container within a VM), as a serverless function (container-based deployment of functions), involving hardware accelerators, and the like. The selection will be based on the value of the utility function per each application architecture variant as well as the hard application requirements (e.g., SLOs). Utilizing such an optimization process will enable achieving the best possible adaptation of the application deployment, compared to the used architecture variant.

3.3.1 Requirements

The requirements are presented in a technical form and are based on the requirements provided in the MORPHEMIC project's Description of Action and the deliverable D6.1, Industrial requirements analysis. Below is the list of technical requirements for the polymorphic adaptation feature:

- Selection of the best deployment architecture for the current application context (including the current workload and achieved performance level) — this is the key requirement for the polymorphic adaptation and one of the main purposes of this feature.
- Analysis of the source code of applications fetched from open-source software repositories and matching it with that of application components — the goal is to recommend new application component configurations/forms based on the experience from the analysis and matching of different open-source applications and systems.
- Assessment and verification of the performance of polymorphic adaptation.
- Preparation of the right input data and models for the MELODIC platform to conduct reasoning based on its main reasoning process.
- None or minimal changes in the MELODIC reasoning process — input data should be prepared by the MORPHEMIC pre-processor in a similar way to the MELODIC input data, to minimize changes in the MELODIC platform.

3.3.2 Integration within MORPHEMIC

The polymorphic adaptation feature is completely realised within the MORPHEMIC pre-processor and relies on the functionality of two main composite components:
The Architecture Optimiser, a new component which supports the selection of the right application architecture variant based on the application requirements and context. Through this selection, it is then possible to apply the classification optimisation process in the MELODIC platform in order to select the right cloud services to support the deployment and execution of the application based on both application requirements and context. It is foreseen that this new selection/optimisation form will rely on the cooperation with the CP Generator to produce CP models of the right content (CP Generation) and potentially one or more new Solvers (or extensions of existing ones) which are able to solve/optimise those models. Furthermore, this Deployment Optimisation form, which we call Architecture Optimisation, will be applicable at both the application's initial deployment but also during runtime in case that the Architecture Optimiser senses the need to change the current application's architecture variant.

The Profiler it is the main vehicle which supports the complete management of the multi-cloud, polymorphic application profile. This latter component actually supports three groups of application profile management functionalities:
Application Profile Construction: the construction of the application profile covering both the functional and non-functional aspect. The functional aspect covers the specification of all application architecture variants while the non-functional aspect covers the specification of application requirements and the derivation of initial performance models for the different architecture variants (NonFunctional Profile Construction).
Functional Profile Enhancement: this is achieved by following a specific enhancement process which relies on crawling open-source software repositories, analysing the source-code of open-source software, classifying that software and matching it with the different application components. In result, each application component has a set of open-source software matching it, where it is also possible that such software might map to new configurations, not currently part of those that have been specified by the DevOps in the application's CAMEL model. As such, the set of different application component configurations could be enriched. However, the platform does not go beyond this point and the actual verification or development of the enrichment lies on the DevOps team. Nevertheless, it does enable to verify new configurations as it is indicated in the next functionality.
Application Profile Maintenance: the maintenance of the application profile with respect to both the functional and non-functional aspect. The Functional Profile Maintenance will rely on verifying new configurations by DevOps and thus enriching the different forms that application components can take in the CAMEL model. The NonFunctional Profile Maintenance will rely on collecting measurements for the application at hand at runtime and utilising them in order to update the non-functional performance models constructed. Furthermore, the functional and non-functional profile maintenance will also attempt to apply changes that concern the application's CAMEL model in terms of the application structure and/or its requirements.
The architecture of the polymorphic adaptation feature is depicted in Figure 7, where different colours have been utilised to denote different components (external – rose, MELODIC – blue, MORPHEMIC pre-processor – orange, model-based artifacts – green). Components with underlined text indicate MELODIC components that are extended with new functionality.

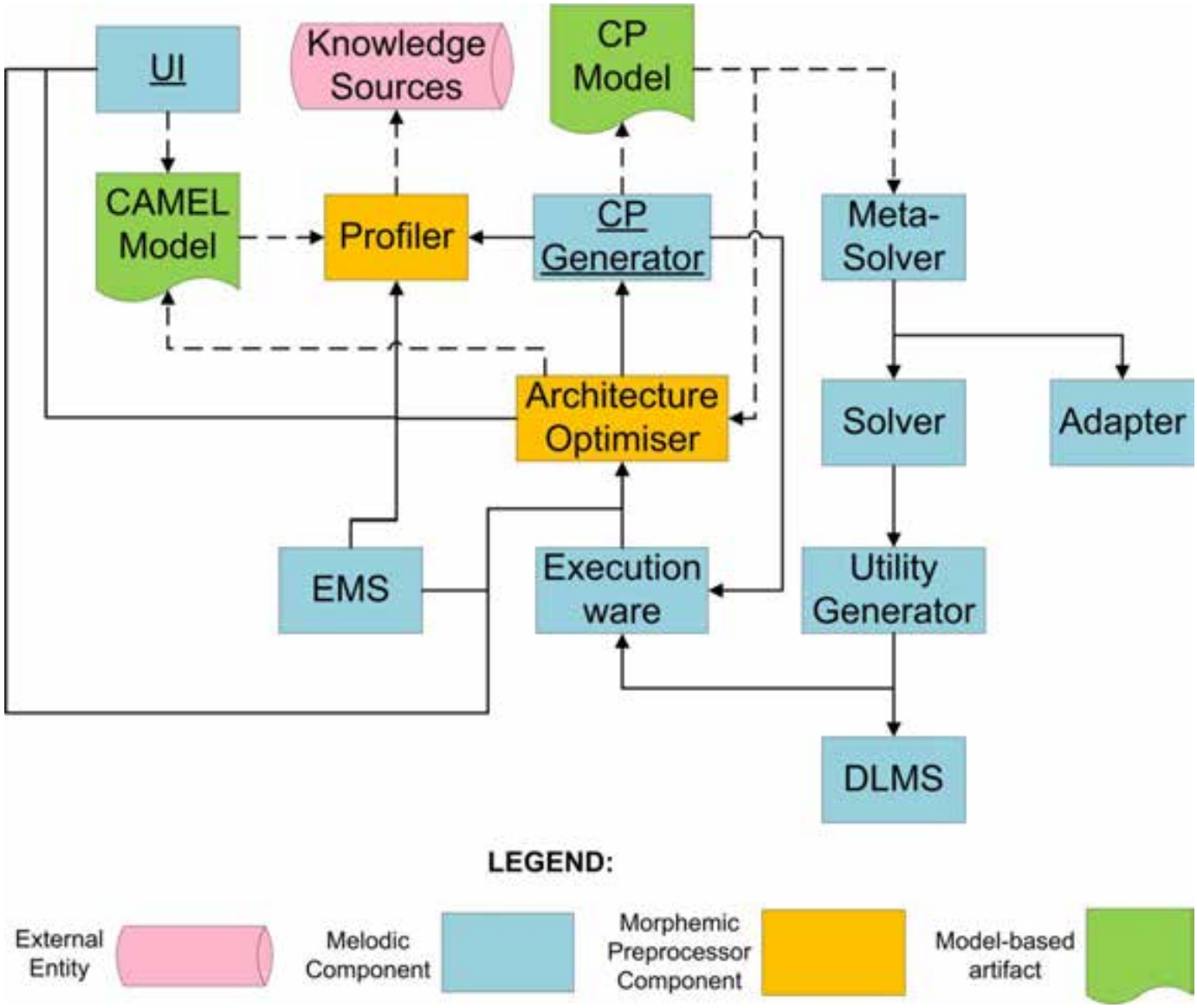


Figure 7: The architecture of the polymorphic adaptation feature in the context of the MORPHEMIC pre-processor

Concerning the model-based artifacts, the first one is the CAMEL model which will conform to CAMEL 3.0, the new version of CAMEL which now supports polymorphic application modelling and will incorporate the new version of the MDS (also catering for polymorphic application modelling) as one of its sub-models. The second is the CP model where in this case, it is foreseen that two variants of it will exist and be exploited by the enhanced MELODIC platform: (a) the classical one applicable to cloud service selection, i.e., the typical optimisation loop in this platform; (b) a new one that will be exploited for supporting a new optimisation loop dedicated to application architecture variant selection through the Architecture Optimiser. As such, this new CP model variant will require a respective extension of the CP Generator component.

A very interesting observation that can be derived from this figure is that the classical optimisation loop, as well as the subsequent deployment orchestration activity, are intact. This means that this feature does not require any extension to the respective MELODIC components supporting deployment reasoning and execution. This is quite logical due to the following reason: as soon as the application architecture variant is selected, the classical MELODIC platform version can handle it in order to support application deployment and adaptive provisioning.

It could be also argued that the changes/extensions introduced by this feature are minimal. However, this is not the actual case. As D3.1 deliverable indicates, the Profiler component is composite, incorporating a plethora of sub-components that are needed in order to support polymorphic application profile management. The same applies for the Architecture Optimiser, which currently comprises two new sub-components, the Decision Maker and the Solver. As such, the changes introduced by this feature are indeed substantial but cannot be actually seen from a high-level conceptual architecture that covers the MORPHEMIC pre-processor. It should be also noted that the Architecture Optimiser's architecture is rather in draft mode and might be expanded in the near future. Such an expansion will be reflected in D3.3.

The above architecture of the polymorphic adaptation feature can be considered as final, especially as it abstracts at the right level the internal details of the Profiler and Architecture Optimiser components. However, changes to the internal architecture of these components cannot be precluded and will be reported in respective deliverables (D3.1 for initial Profiler architecture and D3.2 for the final one plus D3.3 for the initial architecture of Architecture Optimiser and D3.4 for the final one). Further, we should explicate that the aforementioned interactions will guide the extensions to the MELODIC platform at the process level once the respective Profiler and Architecture Optimiser sub-components are realised. Such a realisation is expected to take place at the second project year and thus, the first complete version of these two composite components will be ready and integrated at the 2.0 release of the MORPHEMIC platform.

3.4 Hardware acceleration integration

The integration with hardware accelerators will provide the ability to run selected components of the application using hardware accelerator resources (i.e. FPGA-based accelerators). As a part of the polymorphic adaptation feature, MORPHEMIC will decide if the given components should be run using hardware acceleration. After the polymorphic adaptation and selection of the target architecture, the deployment of the component to selected accelerators will be conducted automatically and transparently.
InAccel has developed a vendor-agnostic orchestrator for FPGA-based accelerators that allows the easy deployment, scaling, and resource management of FPGA clusters.
Among others, the InAccel FPGA resource manager makes much easier the utilization of the FPGAs in software by hiding the complexity of the OpenCL utilization and replacing it with simple software functions. That way, software developers can deploy instantly and utilize a cluster of FPGAs in the same node. The resource manager supports both single-thread and multi-thread applications as the requests for the FPGA cluster are completely decoupled.
At the same time, InAccel has integrated its resource managers with Apache Spark that allows servers with multiple FPGAs to scale to hundreds of nodes through the Spark integration. The InAccel FPGA resource manager covers an essential missing part in the FPGA ecosystem as it allows software programmers to access FPGA accelerators seamlessly.
First of all, the InAccel FPGA orchestrator decouples the host code completely from the hardware kernel. A separate repository is used where the FPGA developers can upload all the functions that have been accelerated in the form of IP blocks. Software users do not need to change their code at all and can invoke the functions that can be accelerated using typical programming languages like C/C++, Python, and Java. Software users do no need to add to their code the name of the bitstreams or the buffer management that will be used for the communication with the FPGA. The InAccel framework automatically overloads the specific functions to be offloaded to the FPGA resources.
Furthermore, InAccel resource manager abstracts away the FPGA resources making feasible the utilization of the FPGA resources from multiple threads, processes, or applications. That way, users can utilize the efficiency of the FPGAs without worrying about conflicts and access to FPGA resources. In the same way the software code is serialized in the CPUs, InAccel manager serializes the requests for specific tasks to be accelerated based on the available FPGA resources.
Finally, InAccel resource manager allows the instant scaling of applications to multiple FPGAs. Software developers can invoke the function to be accelerated from the same thread or different thread or even different applications as many times as needed. Each function invocation is translated to a request for acceleration to the FPGA manager and the manager performs the load balancing and the dispatching of the requests to the available FPGAs in the cluster (multiple FPGAs in the same server). If a user wants to scale it to multiple servers, this is possible using the Kubernetes plugin of FPGA manager.
The FPGA orchestrator allows the deployment of FPGA-accelerated applications as a container or even at a serverless deployment.

3.4.1 Requirements

The general requirements for integrating hardware accelerators are listed below:

1. Support for FPGA hardware accelerators – it should be possible to deploy selected application components to GPU and FPGA hardware accelerators.
2. Ability to automate deployment to hardware accelerators – after the deployment reasoning process is over, the selected components should be automatically and transparently deployed to selected available hardware accelerators.
3. An open interface for integration with several types of hardware accelerators.

3.4.2 Integration within MORPHEMIC

The high-level API provided by InAccel will be used to integrate InAccel with Activeeon's ProActive Resource Manager that is used in the MORPHEMIC's Executionware.
To devise the first steps of the integration work, we exploit the ProActive Platform. It consists of four main components:

- Automation Dashboard, a cloud automation portal where you can manage any on-demand services with full life-cycle management.
- Workflow Studio, an interactive graphical interface that enables engineers to build, train, and deploy machine learning models at any scale.
- Scheduler, where you can orchestrate and automate multi-users, multi-application Jobs and watch the submitted pipelines.
- Resource manager, that manages and automates resource provisioning on public cloud, virtualization software, container system or any physical machine.

All of the components mentioned include REST APIs.
While there are available tasks in the Workflow Studio that enable expressing the GPU requirements and run accelerated pipelines if a NodeSource is set up with that type of resource, we consider adding a separate option for FPGA acceleration. This integration consists of three main steps:

1. Create and deploy a new NodeSource with FPGAs.
2. Configure the environment.
3. Use the InAccel Docker Image on every task we want to accelerate.

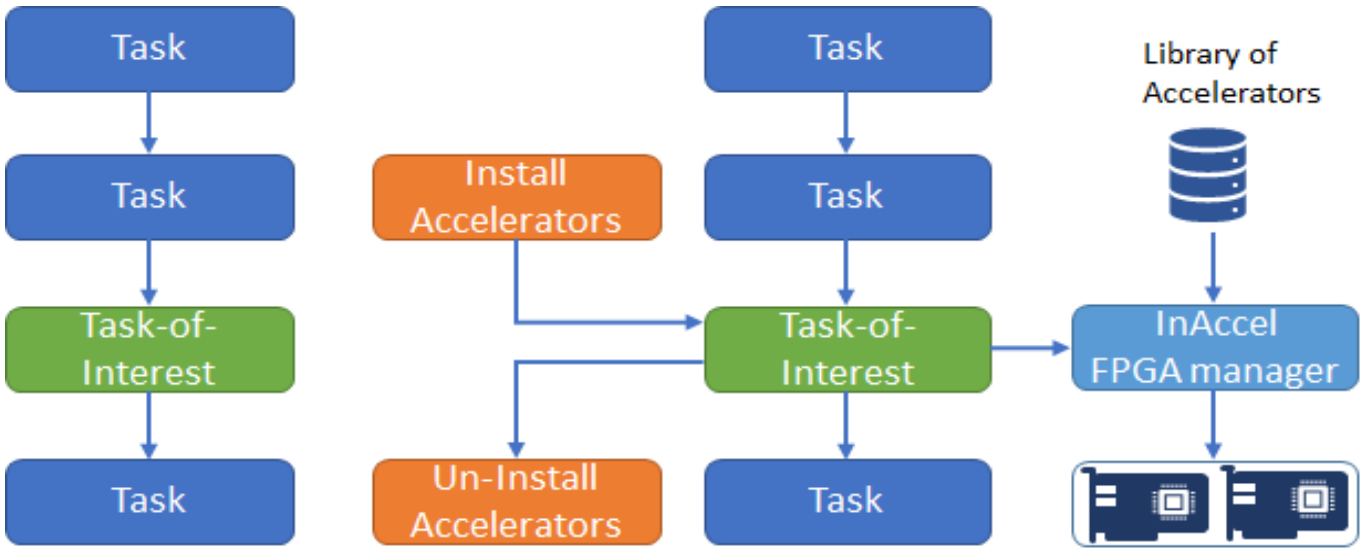


Figure 8: High-level overview of the accelerated tasks deployed on the FPGA-based cloud resources using the InAccel FPGA resource manager

Figure 8 initially shows a generic flow on the left side and a template of job acceleration on the right. The orange blocks show the configuration steps that need to be run once for the initialization of the FPGA resources. After the initialization, the tasks-of-interest can be offloaded to the FPGA resources for acceleration. Firstly, we must install the accelerators' files, then inside the specified task we must use InAccel's libraries (i.e., InAccel Keras) and then uninstall the accelerator on a separate task.
The Accelerated task specified in the Proactive Resource Manager is automatically offloaded (shown on Figure 9) by InAccel into the FPGA resources available. The task is dispatched on the FPGA resource manager. The resource manager based on the available resources will automatically program the FPGA and dispatch the tasks to the available FPGA resources. After the task has been finished, the results will return to the invoked function.

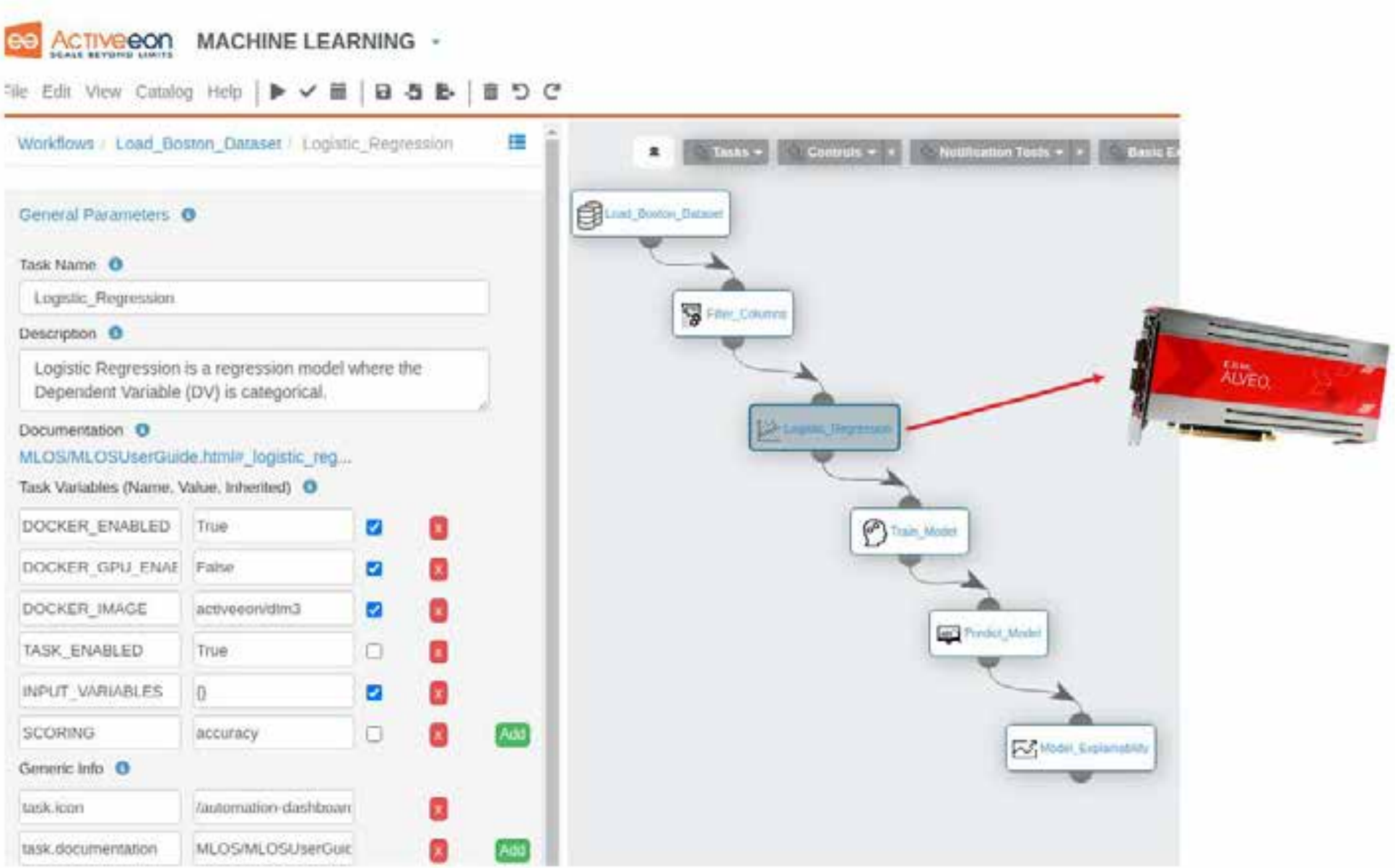


Figure 9: Offloading computationally intensive application on the FPGAs. A use case where a task is offloaded to the FPGAs through the InAccel resource manger

Once the design and the implementation of the accelerated pipeline is validated, it will be integrated into the MORPHEMIC architecture through the Scheduler Abstraction Layer component: the latter will feature the generation of reconfiguration workflows supporting the acceleration pipeline.

3.5 Security requirements and assumptions

Security in MORPHEMIC can be intended in two different meanings:

1. Platform security, related to the platform by itself, including access control mechanisms, privacy, and confidentiality for user data and capability to prevent unauthorized access to the services offered to a specific user.
2. Deployment security, i.e., the capability to select the most appropriate deployment environment and application form according to the security level required by the application: it has strong dependency on the associated infrastructures representing the offer that MORPHEMIC will take into account for the applications.

The MORPHEMIC platform is conceived to potentially interact with every deployment environment and every application. This means that the Deployment security will be provided if MORPHEMIC will provide these capabilities. More complex is the analysis of Platform security: the next subsections will be focused on the kinds of security defined by these two meanings.
The MORPHEMIC platform includes the functionalities of MELODIC and the new features described in this and other deliverables. At a high level, both platforms enable users to deploy applications on certain environments. However, MORPHEMIC will provide polymorphic and proactive adaptation, which will propose new challenges in terms of security. For this reason, MELODIC will be the starting point also concerning security requirements, which will be revised against the new requirements and new functionalities of MORPHEMIC. This section will analyse the preliminary information provided by the architecture and the industrial requirements detailed in the Deliverable 6.1. A complete analysis that will give a detailed description of the security feature of MORPHEMIC will be provided with the Deliverable 4.2.
The deployment security mainly depends on the security level offered by the associated infrastructures, but also on the capability of MORPHEMIC to derive an appropriate deployment model and include security related services in the deployment.

Platform security

The platform security is critical for the architecture of MORPHEMIC for the following reasons:

- Platform security directly impacts the security of the users of the MORPHEMIC and can have legal implications (e.g., GDPR).

The MORPHEMIC platform's architecture and security features can completely define the level of provided platform security. As mentioned above, the starting point of the MORPHEMIC platform is the MELODIC platform, from which most of the security features are inherited. Some of these security features will be reused. The preliminary analysis of the MELODIC platform has already provided some suggestions for improvement, which will impact the implementation of the MORPHEMIC platform: for this reason, some of the reused security features will be revised according. These suggestions include new policies concerning the interactivity with the public internet, limiting exposed networking ports and exploring the advantages of a reverse proxy to protect the platform from malicious activity. At time of writing this deliverable, this analysis is in progress and its results will be included in the Deliverable 4.2.
Concerning the MORPHEMIC architecture, a high security level will be guaranteed for both communications among internal components and interactions with the external network. Specifically, the most critical internal communications will be encrypted through TLS and secured through JWT-based authentication.
TLS encryption will also be used for the interaction between the user and the MORPHEMIC platform, along with username/password-based authentication, password complexity check, ABAC authorization and secure data storage.
At the time of writing of this deliverable, some aspects of the architecture remain to be fully defined, especially concerning the implementation details of some functionalities, especially concerning polymorphic and proactive adaptation. In many cases, the security aspects strongly depend on the technological choices, so it is possible that new security features will be added, or some legacy functionalities will be updated. The Deliverable 4.2, due in M18, will provide a complete analysis also defining all the aspects currently under discussion.

Platform security: related requirements

The complete list of industrial requirements is presented in the deliverable D6.1 (Industrial Requirements Analysis). Concerning platform security, the high-level requirement is the following:

- MEL-8: Secure and context-aware data access control mechanism.

This single, high level requirement defines the need for a complete access control mechanism to protect the data in the platform. This mechanism is present in MELODIC and will be probably improved in MORPHEMIC: in order to be fully satisfied the platform must provide secure access and secure communication mechanisms.

Deployment security

According to the requirements collected in the deliverable D6.1, it is possible to distinguish three categories of deployment security requirements, for which the impact on the MORPHEMIC platform is very different:

- Infrastructure-related features, related to the intrinsic security level of the infrastructure, e.g., on-premises vs cloud, private execution environments and support for traffic isolation.
- Application-related features, related to the capability to deploy applications supporting data privacy and security, such as secure databases, services supporting TLS and certificates or authentication mechanisms.
- The capability to deploy security management related services, such as intrusion detection systems, network monitors, etc.

It is easy to understand that even if the deployment security is much more related to the industrial requirements provided by the use cases, it has little impact on the architecture of the MORPHEMIC platform for two main reasons:

1. Infrastructure-related features, depends only on the associated infrastructures: once an on-premises infrastructure is associated with MORPHEMIC, it can be used for deployment providing its security features
2. Application-related features, and 3, security management services, ultimately concern services which the MORPHEMIC platform should be able to install on the associated infrastructures. However, MORPHEMIC should not express any limitation in terms of the deployable services, so these requirements are contained in the more general definition of the MORPHEMIC platform.

The deliverable D6.1 also includes a list of deployment environments that should be supported to obtain the required security level. These environments present different priority levels and are required by different use cases.

Deployment security related requirements

Most of the deployment security requirements are optional features that will make the selection of the environment on which the application will be deployed more fine-grained. Generally, the ability to deploy generic applications assures that also security related services (such as intrusion detection) are deployed by MORPHEMIC. Table 2, extracted from Deliverable 6.1, includes the relevant security requirements, their priority (MoSCoW model) and their assigned category.

Table 2: Security requirements

ID	Description	Priority	Category
UC-C-SEC.1	Support for traffic isolation	COULD	Infrastructure
UC-C-SEC.2	Support for secure communications	COULD	Application-related
UC-C-SEC.3	Support for security-related applications	COULD	Security-management-related services
UC-1.SE.8	Support for Security Management	COULD	Security-management-related services
UC-2.SE.2	Support for private execution environment	MUST	Infrastructure

3.6 Self-healing requirements and assumptions

The goal of the self-healing capabilities feature refers to the necessary methods and tools for propagating and processing application and infrastructure-specific monitoring information in order to detect current and future situations where application requirements might be violated. This involves the automatic deployment and exploitation of real-time performance monitoring data along with their efficient processing across dispersed multi-cloud and fog

environments. It refers to an advanced and distributed monitoring and complex event processing system that will be able to self-heal and cope with situations where monitoring and aggregation nodes become unreachable due to a failure on the hosting infrastructure or the network. This feature builds on and extends the event management system (EMS), designed and developed in the MELODIC project, by introducing appropriate health checks and automatic recovery functionalities with respect to the monitoring infrastructure. Based on these capabilities, application morphing and/or topology reconfigurations will be enacted for continuously guaranteeing the conformance to defined application specifications and QoS requirements.

3.6.1 Requirements

The requirements are collected based on the MORPHEMIC Description of Action and the deliverable D6.1 regarding the use case application requirements. This feature's requirements involve all the aspects concerning setting up the monitoring infrastructure and processing dispersed and heterogenous application performance information. The aim is to detect current or upcoming SLO violations, propagate them and assist in the continuous assessment of the current or future overall utility for a certain application by updating important application model information. We note that upcoming SLO violations can be detected by processing forecasted metrics produced by the proactive adaptation features of MORPHEMIC. In addition, it is required to achieve such capabilities in a bandwidth efficient way while guaranteeing uninterrupted operation of the monitoring infrastructure, even in cases where some nodes fail. Specifically, the following list encapsulates the main requirements for this feature:

- Ability to automatically deploy and configure the appropriate monitoring probes for sensing application and hosting infrastructure-specific performance metrics that will allow for continuous analysis of all the application components deployed in multi-cloud and fog environments.
- Ability to deploy a federated and resilient complex event processing system, able to detect service level objectives violations or any other dangerous situations, concerning the deployed application components, in order to enact reactive or proactive adaptations.
- Ability to persist monitoring information to be exploited by the forecasting capabilities of the MORPHEMIC platform (i.e., to be handed over to the forecasting mechanism for enhancing metrics prediction)
- Ability to assess situations according to the service level objectives (SLOs) defined in CAMEL, update critical aspects of the utility function used during the reasoning process and trigger (through Metasolver), when necessary, the reconfiguration process.

3.6.2 Integration within MORPHEMIC

This section refers to the high-level description of the main components that will implement the feature's capabilities based on the requirements mentioned in the previous section. Specifically, we refer to the following components depicted on Figure 10that correspond to core architectural elements of the MORPHEMIC platform:

- EMS
- Time-series Database
- Metasolver
- Adapter

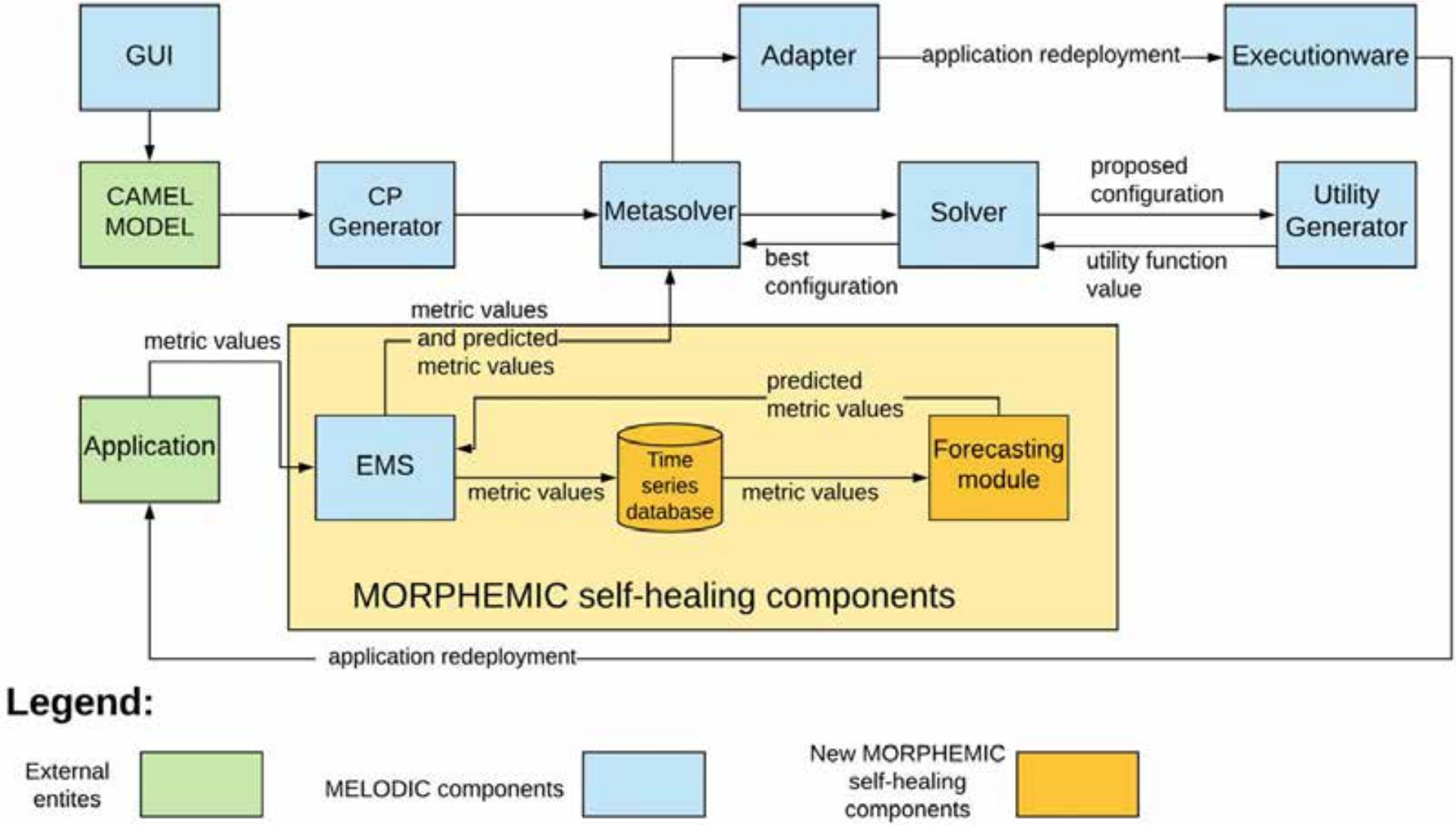


Figure 10: Core architectural elements of the MORPHEMIC self-healing capabilities feature

The Event Management System (EMS) is a distributed application monitoring system, which was developed and used as part of the MELODIC Upperware for monitoring the operation of the cross-cloud applications, in order to take appropriate actions when certain application-related performance objectives constraints are violated. The scope of the specific system will be significantly extended in the MORPHEMIC platform in order to be able to collect, process and deliver to interested parties, monitoring information pertaining to distributed applications deployed in multi-clouds and fog computing environments, according to CAMEL model specifications. Upon polymorphic application deployment the appropriate monitoring and analysis tools will be enacted in parallel to continuously observe and analyse the application operation and performance. The monitoring probes deployed will involve capabilities to aggregate any details of the hosting infrastructure structure or any application-level information deemed necessary by the DevOps (i.e., Average RAM usage, requests per second etc.). In addition, it will involve additional server and client-side components for health checking of the monitoring topology status and also extending topology management capabilities of EMS towards an even more decentralized approach that follows clustering techniques when electing the responsibilities of each monitoring node. Therefore, the EMS will be extended with self-healing capabilities. Last, the EMS should treat current and forecasted metrics transparently when trying to detect current or impending SLO violations that should trigger a new reconfiguration process.
A significant part of raw and processed monitoring metrics should be persisted in a dedicated time-series database. This database should be efficient enough to deal with a varying rate of incoming heterogeneous timeseries that represent a subset of all monitoring metrics acquired by the deployment topology. This will involve the appropriate subcomponent which, based on the instructions of the EMS, will be able to subscribe to events propagated in different layers of the monitoring infrastructure (i.e., instance level, cloud level, platform level), extract from the event payload the timestamp, metric name and value of each simple or composite metric and persist it in the database.
The Metasolver is another component developed as a microservice of the MELODIC platform, that undertakes the task of selecting an appropriate solver for a given CP problem, updating metric values in the utility function acquired from the EMS and subsequently verifying that the solution yielded by the solver is significantly better than the currently deployed one. A configurable threshold is used for defining how much better the new solution should be in order to trigger the implementation of an application adaptation. This component will also be extended in the MORPHEMIC platform in order to consider current and forecasted metrics when updating the utility function, thus enabling the reactive and proactive optimisation process. Essentially, this means that the processed monitoring data which may reveal changes in the application's behaviour will be propagated in the model morphing process through Metasolver.
The Adapter is a core MELODIC component, responsible for preparing a complete plan of application reconfiguration for a new deployment accepted by the Metasolver. The plan includes a series of tasks to be sent for execution to the Executionware, following a specific order. To fulfil this requirement, a graph structure is maintained internally by the Adapter to reflect the application structure along with the dependencies among tasks. This component will be also extended in the MORPHEMIC for coping with the advanced capabilities of the new platform's Executionware.

3.7 GUI for CAMEL modelling and UI extensions

The goal of this feature is to provide an ergonomic and user-friendly user interface for the preparation of the CAMEL model. Also, additional changes in the existing GUI for MELODIC should be implemented to support polymorphic and proactive adaptation.

3.7.1 Requirements

The key requirements for developing and extending the GUI, that have been collected together from all members of the consortium and from Use Case partners during the work on D5.1, are:

- Designing and implementing an ergonomic GUI for CAMEL modelling.
- Extending the MELODIC platform's GUI to support the new MORPHEMIC capabilities (i.e., polymorphic and proactive adaptation).
- Improving the existing GUI of MELODIC and ProActive Scheduler to increase operational efficiency.

Detailed and prioritized requirements are provided in the deliverable D5.1 "User Interface Specification" and D5.2 "User Interface Guidelines".

3.7.2 Integration within MORPHEMIC

The CAMEL modelling GUI (CAMEL Designer) will be integrated with the platform through data interchange (CAMEL models). It will be possible to fetch and upload the CAMEL model to the MORPHEMIC platform using the CAMEL Designer. The other GUI extensions will be developed following the existing UI architecture which includes an Angular-based web application and a backend REST-compatible API to provide the necessary input to the GUI.

3.8 Use-case application adaptation requirements

The use-case application adaptation activity is focused on maximising the benefits of the use cases applications deployments on multi-cloud environments using the MORPHEMIC platform. This activity is focused on the industrial requirements, suggested by the use case providers, which integrate and complete the preliminary proposal in the Description of Action. The set of requirements produced, along with the ones coming from the use cases, derives from the MELODIC Platform and the new concepts introduced in MORPHEMIC, namely Polymorphic and Proactive adaptation.
The use cases applications were chosen to cover a wide domain of digital cloud services: from software-defined networking, to industry, through healthcare. Each domain has its specific issues and requirements, all of which the MORPHEMIC platform will have to meet.
The next subsection describes the requirements whose details are available in [15].

3.8.1 Requirements

The requirements collected and analysed during this activity, have been categorized in three main groups and are detailed in deliverable 6.1:

- Requirements derived from MELODIC.
- Requirements specific to MORPHEMIC.
- Requirements defined by the use case providers.

Requirements derived from MELODIC
The requirements derived from the MELODIC Platform mainly concern multi-cloud and cross-cloud deployments and executions of data intensive applications and were in part provided by a set of use cases. The MORPHEMIC platform will be able to support different cloud infrastructures and different data sources by extending the original set supported by MELODIC.
Part of the requirements indicates the needed mechanisms of Access Controls and User Management: the details are presented, besides the mentioned list of the deliverable D6.1, in section 3.5, and in the deliverable D4.2, due by M18.
Finally, some non-functional requirements of the MORPHEMIC platform are derived from MELODIC, specifically extensibility, reusability, documentation, quality, fault tolerance and scalability.

Requirements specific to MORPHEMIC
The requirements specific to MORPHEMIC extend the multi-cloud support already provided in MELODIC, with the multi-environment support. Specifically, support for Edge, Fog, virtual machines, HPC, etc. will be included, with different priorities. Furthermore, MORPHEMIC will support different application forms, such as virtual machines, containers and serverless: finally, adaptation and self-healing are part of the requirements.
Polymorphic and proactive planning, modelling and adaptation are well defined by this category of requirements.

Requirements defined by the use-case providers

- The requirements derived from the use cases are very important to define the utility of the MORPHEMIC Platform in real life. The three considered use cases are Virtualized base station for 5G cloud-RAN, proposed by IS-Wireless, e-BrainScience, proposed by Centre Hospitalier Universitaire Vadois (CHUV) and Computational Fluid Dynamic Simulation, proposed by ICON.

They cover a wide spectrum of usage possibilities, providing a reliable indicator of what users demand and how a product like MORPHEMIC can be useful.
The deliverable D6.1 contains three tables with different lists of requirements and different priorities, including details and some comments. This section provides a high-level description of the main requirements associated to each use case. In particular:

- A Virtualized base station for 5G cloud-RAN is focused on software defined networks including specialized hardware components. This means that the capability to include hardware in the deployment model is required by this use case. Furthermore, a certain degree of management in terms of fault and performance is requested in SDNs: this means that, on one hand, the MORPHEMIC platform must be able to deploy specialized components for performing these controls, and, on the other hand, the overall deployment model should take into account the performance of the final applications. However, the latter aspect is considered a low-priority one.
- E-BrainScience is focused on neuroimaging, i.e., high-resolution medical graphical data. The most immediate requirements that are easily understandable in this context are the capability to manage data and security. Actually, since the scientists working on this topic are located in different places, geographical awareness is

also requested. For the specific activities that this use case should perform, security is intended as a capability to associate private environments on which private data can be safely uploaded. Finally, since part of e-BrainScience platform is ProActive Workflow, the specific requirement to efficiently deploy this product is mandatory.

- Computational Fluid Dynamic Simulation is focused on simulations requiring a variable number of resources. Specifically, Icon plans two categories of simulations: hi-fi and low-fi. The former requires high performance, especially concerning CPU, the latter requires less resources and could offer the possibility to optimise resource usage if dynamic re-deployment will be supported. Hardware components and hardware acceleration can be useful for hi-fi applications. Taking into account the price as a parameter of the Utility Function can be useful to optimise the resource usage for low-fi applications.

4 Preliminary MORPHEMIC architecture

4.1 General MORPHEMIC architecture

This section contains the description of the initial architecture of the MORPHEMIC platform. It has been designed based on baseline frameworks, requirements and the features presented in the previous section. Also, the decision of replacing Cloudiator by ProActive influenced the design of architecture, as described in section 3.1. The architecture is presented in the following subsections:

High level component architecture – This subsection contains the high-level component architecture of the MORPHEMIC platform and a description of the components.

High level sequence flow – This subsection contains a description of the two key processes within the platform: The initial deployment flow and the reconfiguration flow. These processes cover all the elements of the platform and show how components interact with each other.

The architecture and interaction diagrams are depicted with the use of the Unified Modelling Language (UML) [6].

The detailed architecture of the following components and features will be presented as follows:

1. Proactive adaptation – the details of the particular components and methods, their design and their implementation will be described in the following deliverables:
    a. D2.2 Implementation of a holistic application monitoring system with QoS prediction capabilities
    b. D2.3 Proactive utility: Framework and approach
    c. D2.4 Proactive utility: Algorithms and evaluation
    d. D3.3 Optimized planning and adaptation approach
    e. D3.4 Planning and adaptation results
2. Polymorphic adaptation – the details of the particular components and methods, their design and their implementation will be described in the following deliverables:
    a. D1.1 Data, Cloud Application & Resource Modelling
    b. D1.2 Component Specification Collection & Enrichment Mechanisms
    c. D1.3 Final Data, Cloud Application & Resource Modelling
    d. D1.4 Final Component Specification Collection & Enrichment Mechanisms
    e. D3.1 Software, tools, and repositories for code mining
    f. D3.2 Automatic source code identification of deployment modules
    g. D3.3 Optimized planning and adaptation approach,
    h. D3.4 Planning and adaptation results
    i. D5.5 Runtime optimisation mechanism
3. Hardware accelerators integration – the detailed architecture and integration process will be described in D5.4, Accelerator resource manager and accelerators
4. UI for CAMEL modelling and UI extension – the detailed requirements and user interface guidelines are described in:
    a. D5.1 User Interfaces Specification
    b. D5.2 User Interface Guidelines
5. Self-healing capabilities – the details of the particular components and methods, their design and their implementation will be described in the following deliverables:
    a. D2.1 Design of a self-healing federated event processing management system at the edge
    b. D2.2 Implementation of a holistic application monitoring system with QoS prediction capabilities
6. Security related details will be presented in D4.2, Security design and implementation.
7. The extension of ProActive Scheduler and Integration within MORPHEMIC platform will be described in the following deliverables:

   a. D5.3 Deployment artefact manager
   b. D5.5 Runtime optimization mechanism
8. Minor architecture changes will be described in D4.3 Selection, design and implementation of integration layer.
9. Use case application requirements are presented in the deliverable D6.1, Industrial requirements analysis.

4.2 High level component architecture

This section contains the list of all the top-level components of the MORPHEMIC platform. The MORPHEMIC platform follows the microservice architecture principles [1]. All the MORPHEMIC components are deployed using Docker containers [2], and the platform is built using the baseline frameworks MELODIC and the ProActive Scheduler. The MORPHEMIC pre-processor is mainly responsible for polymorphic and proactive adaptation, both of which are essentially done by performing the architecture selection and predicting future metrics values. As a next step, this information is provided to the MELODIC Upperware to execute the reasoning process to find the optimal deployment solution, which is then enacted upon and orchestrated. The deployment itself is performed by the ProActive Scheduler. A visualization of the relations between the MORPHEMIC pre-processor, the MELODIC Upperware and the ProActive Scheduler is depicted in Figure 5. The more detailed preliminary high-level architecture is presented in Figure 11.

The MORPHEMIC pre-processor is communicating with baseline frameworks through their public API. For synchronous communication, the REST API [3] with the JSON scheme [4] is used. For asynchronous communication, ActiveMQ 3 is used.

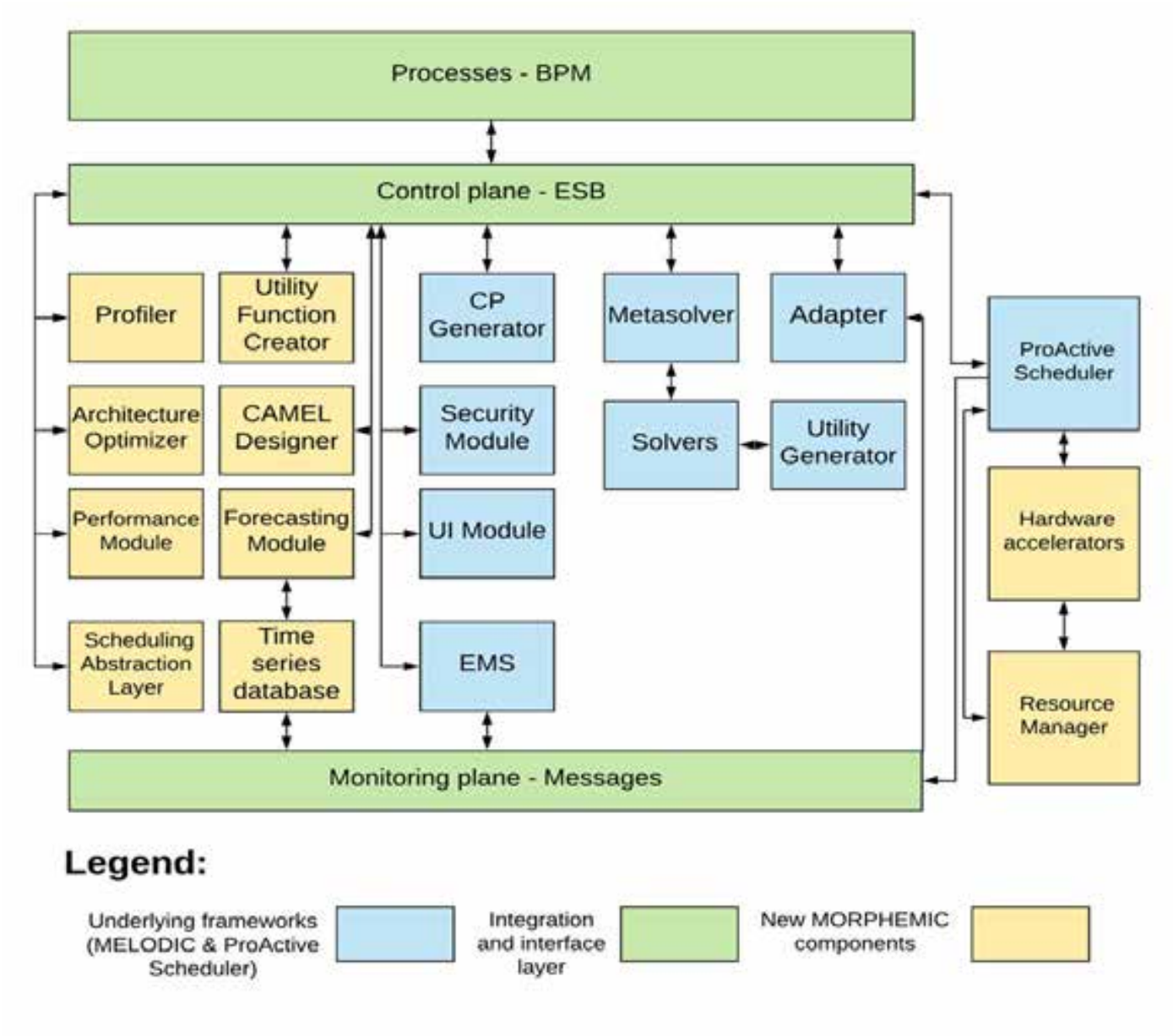


Figure 11: MORPHEMIC preliminary architecture

A list of the key components of the MORPHEMIC and baseline frameworks is presented below. For each component a short description of its role is provided, including information about its state (new/updated/not changed comparing to the baseline).

The components are depicted on the high-level architecture in Figure 12. The list of the components is presented in Table 3.

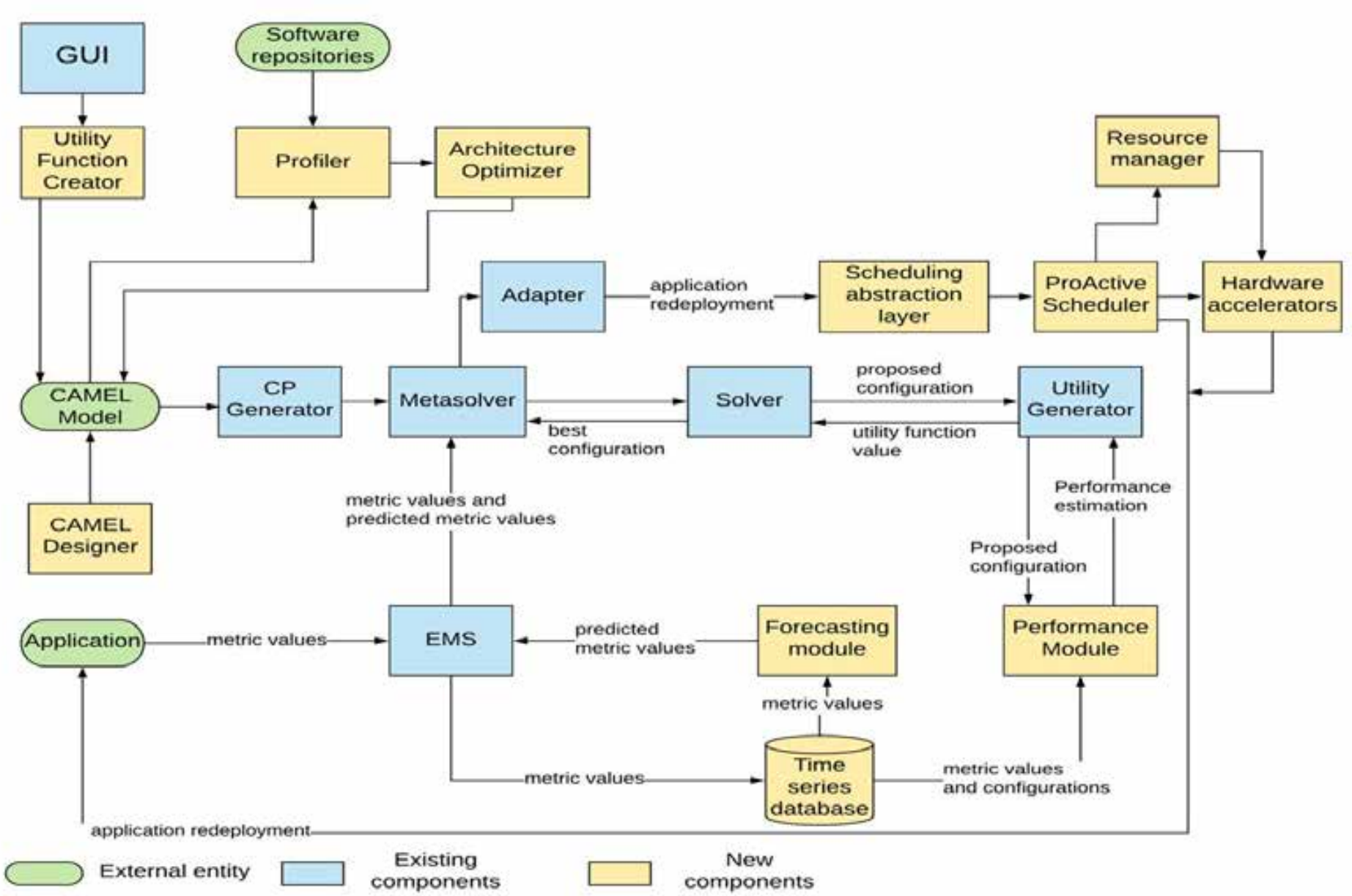


Figure 12: Key components of MORPHEMIC

Table 3: Key components of the MORPHEMIC pre-processor and proactive reconfiguration high-level architecture

Component name **(The name of the top-level component)**	Status **(Status of the component: New or Existing)**	Description **(The description and the role of the component)**	Planned development work **(Brief description of the planned development work for the existing components within the MORPHEMIC project)**
GUI	Existing	Graphical User Interface to configure and operate the platform.	The existing MELODIC GUI will be extended to cover the necessary MORPHEMIC configuration and operations.
Utility Function Creator	New	Creator of the utility function based on the user's preferences.	The component will be created.
Profiler	New	Responsible for constructing and maintaining the profile of an application.	The component will be created.
Architecture Optimizer	New	Responsible for optimizing the architecture of the application.	The component will be created.
CP Generator	Existing	Generator of the Constraint Programming model, which is solved by the Solver.	No or minimal changes to the CAMEL Parser needed to support updated versions of CAMEL.
Metasolver	Existing	Responsible for triggering the reconfiguration and selecting the Solver.	No changes foreseen.
Adapter	Existing	Orchestrating deployment of the application through the Executionware.	The component will be adapted and integrated to interact with ProActive Scheduler.
Solver	Existing	Set of sub-components responsible for solving the CP problem.	No changes foreseen.
Utility Generator	Existing	Calculation of the utility function value.	No or minimal changes for an integration with the Performance Module.

Scheduling Abstraction Layer	New	Abstraction layer for the ProActive Scheduler.	The component will be created.
ProActive Scheduler	Existing	Deployment and reconfiguration of the application to the cloud providers.	This component will replace the current Executionware of MELODIC and adoption and integration with MELODIC is necessary.
Resource Manager	Existing	Deployment and configuration of the infrastructure resources exploited by the MORPHEMIC applications.	The component will be extended to cope with MELODIC features and infrastructure scoped by polymorphic adaptation feature.
Hardware Accelerator	New	Deployment to hardware accelerated cloud computing resources.	The component will be created [16].
CAMEL Designer	New	GUI-based creation of the CAMEL model.	The component will be created based on the Modelio software.
EMS	Existing	Collection and processing of metric values	The component will be extended with self-healing capabilities and will support the propagation and persistence of predicted metric values.
Forecasting Module	New	Forecasting of metric values.	The component will be created.
Performance Module	New	Maintaining a performance model for application and cloud resources	The component will be created.
Time series Database	New	Storing real and forecasted metric values.	The component will be added to the platform.

4.3 High level sequence flow

This subsection contains the high-level sequence flow of the deployment and optimisation processes. Two individual diagrams and flow descriptions are presented:
The presented flows are the initial version of that flows and can be changed during the implementation to align with the final architecture.

4.3.1 Initial deployment flow

Initial deployment flow – depicted in Figure 13. It covers the actions and components needed to achieve the initial deployment of the application using the MORPHEMIC platform. The initial deployment is conducted once, as the first deployment of the application. The result of this flow is the deployed application using the best deployment solution found by the MORPHEMIC platform.

The MORPHEMIC initial deployment flow contains the following actions:

1. The user creates a CAMEL model using the GUI (Camel Designer) and saves it to the database.
2. The user defines his/her preferences related to the utility function in the GUI.
3. The GUI passes user preferences to Utility Function Creator.
4. Utility Function Creator creates a utility function, updates a CAMEL model, and stores it in the database.
5. The user using the GUI starts the reasoning process.
6. GUI invokes the Architecture Optimizer to start pre-reasoning.
7. The Architecture Optimizer fetches the CAMEL model.
8. The Architecture Optimizer invokes the Profiler to retrieve the application profile.
9. Based on the CAMEL model and the application profile from the Profiler, the optimal architecture of the application is chosen by the Architecture Optimizer.
10. The CAMEL model containing the specific architecture is created and passed to the CP Generator.
11. The CP Generator fetches possible offers from cloud providers using the ProActive Scheduler.
12. The CP Generator stores possible offers as node candidates to cache.
13. The CP Generator creates a CP Model which is passed to the Metasolver to start solving the problem.
14. The Metasolver selects a Solver and passes the enhanced CP Model with metrics and predicted metrics values to the selected Solver.
15. The Solver calculates a new and possible better candidate application configuration and then evaluates candidate application configuration by invoking the Utility Generator.
16. The Utility Generator fetches the Node Candidates from the Cache.
17. The Utility Generator invokes the Performance Module to estimate the performance of the candidate application configuration.
18. The Performance Module fetches the metric values to estimate performance and returns the estimated performance value to the Utility Generator.
19. The Utility Generator calculates the value of the utility function for a given candidate application configuration and returns it to the Solver.
20. Steps from 15 to 21 are repeated until the best solution is found.
21. The best solution is passed from the Solver to the Metasolver.
22. The Metasolver invokes the Adapter to orchestrate the deployment of the best solution.
23. The Adapter invokes the ProActive Scheduler via Scheduling Abstraction Layer to execute the deployment. If the application should be deployed using hardware accelerators, the ProActive Scheduler invokes the Hardware Accelerator to perform the hardware acceleration as described In section 3.4 and in [16].
24. After successful deployment, metrics from the application and the system are being sent to the EMS.
25. The EMS stores metrics in Time Series Database.
26. The Forecasting Module is forecasting future metric values.
27. The Forecasted metric values are sent to the Metasolver in the case of an SLO violation, to trigger the reconfiguration (see Reconfiguration flow).

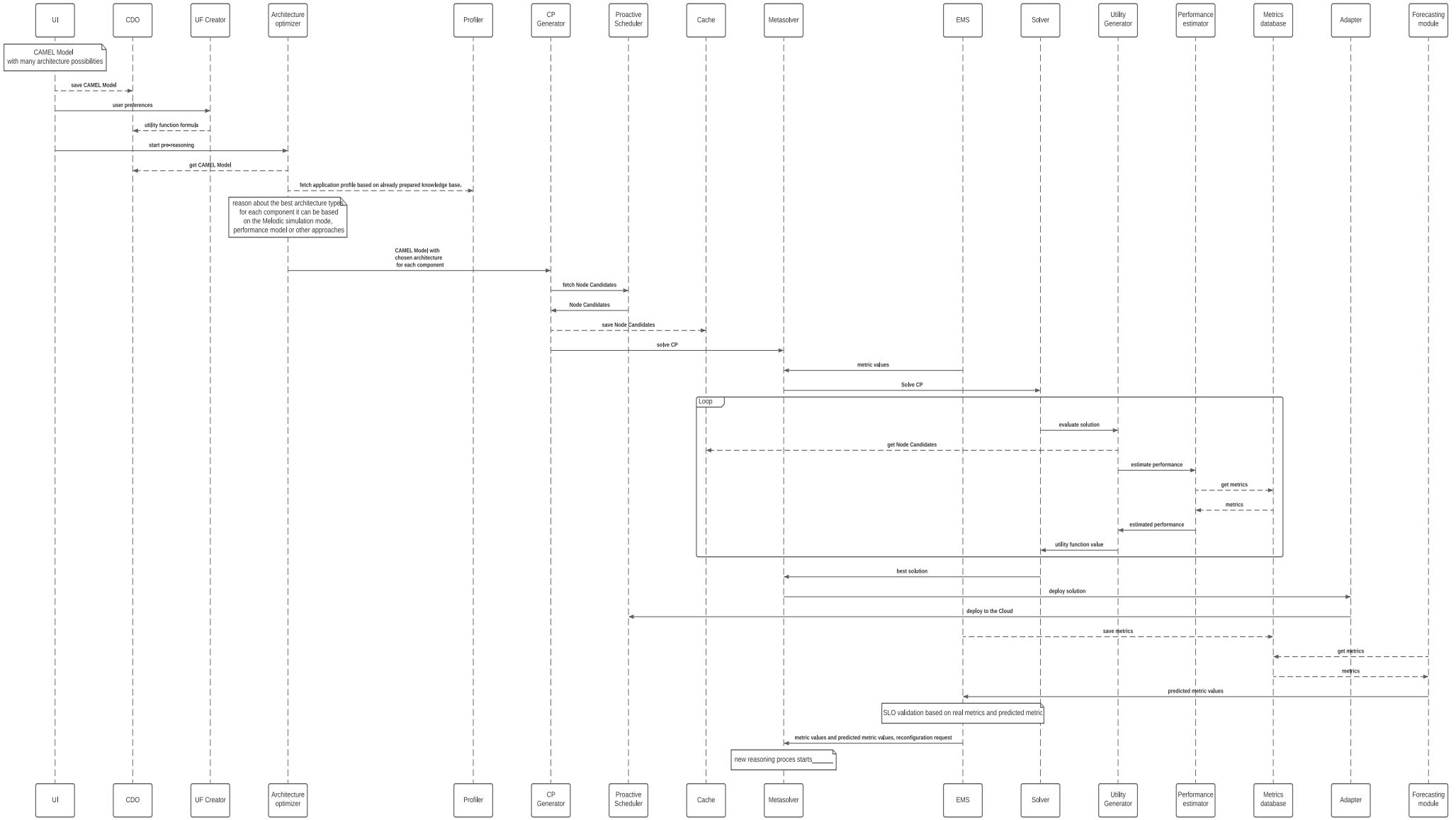


Figure 13: MORHPEMIC Architecture sequence - Initial deployment

4.3.2 Deployment reconfiguration flow

Reconfiguration deployment flow - depicted in Annex 1 or Figure 13 “MORPHEMIC Architecture sequence - Initial deployment”. It covers the actions and components needed to reconfigure applications and is based on the components of the MELODIC platform. The result of this flow is a reconfigured application adapted to the anticipated workload
The reconfiguration deployment flow (see Annexe 2 or Figure 14 “Architecture sequence – Reconfiguration”) contains the following actions:

1. The Architecture Optimizer is continuously doing optimization of the architecture based on actual values of metrics by executing the steps 2 to 5.
2. The Architecture Optimizer fetches the CAMEL model.
3. Based on the CAMEL model and the application profile fetched from the Profiler, the optimal architecture of the application is chosen by the Architecture Optimizer.
4. A CAMEL model with a current operational context is created and passed onto the CP Generator.
5. If the CAMEL model is different from the one currently deployed, then the CP Generator creates a CP Model after fetching available Node Candidates that fulfilling requirements from CAMEL Model which is passed onto the Metasolver to start solving the problem as described from step 10.
6. The EMS module continuously monitoring application and Forecasting Module continuously forecasts values of the metrics as described in steps 7 to 9.
7. The Forecasting Module fetches metric values from the Metric Database.
8. The Forecasting Module forecasts metric values and sends it to EMS.
9. The Metasolver fetches metrics and predicted metrics values from EMS.
10. In case of pushing new model by Architecture Optimizer or triggering SLO violation the reconfiguration process is started as described in steps 11 to 23.
11. The Metasolver selects a Solver and passes the CP Model, metrics, and predicted metrics values onto the selected Solver.
12. The Solver is looking for the best solution and evaluates candidate application configuration by invoking the Utility Generator.
13. The Utility Generator fetches the Node Candidates from Cache.
14. The Utility Generator invokes the Performance Module to estimate the performance of the candidate application configuration.
15. The Performance Module fetches the metric values to estimate performance and then returns the estimated performance to the Utility Generator.
16. The Utility Generator calculates the value of the utility function for a given candidate application configuration and returns it to the Solver. Solver compare the value of the utility function for given candidate application configuration and compare to the current best one. If the current candidate application configuration has better utility then it replaces previous best one. Steps from 12 to 16 are repeated until the best solution is found.
17. The best solution is passed from the Solver to the Metasolver.
18. The Metasolver invokes the Adapter to orchestrate the reconfiguration of the application deployment and deploy the best solution.
19. The Adapter invokes the ProActive Scheduler via Scheduling Abstraction Layer to execute the reconfiguration. If the application should be deployed using hardware accelerators, the ProActive Scheduler invokes the Hardware Accelerator to perform the hardware acceleration.
20. After successful deployment, metrics from the reconfigured application are being sent to the EMS.
21. The EMS stores metrics in a Metric Database.
22. The Forecasting Module is forecasting metric values.
23. The forecasted metric values are sent to the Metasolver in case of an SLO violation to trigger reconfiguration. The reconfiguration process will restart from step 11.

The presented sequence flows cover the key processes of the MORPHEMIC platform, including proactive and polymorphic adaptation, integration with hardware accelerators and self-healing monitoring system (EMS). Additional processes can be design and implemented during the implementation and described in the respective deliverables.

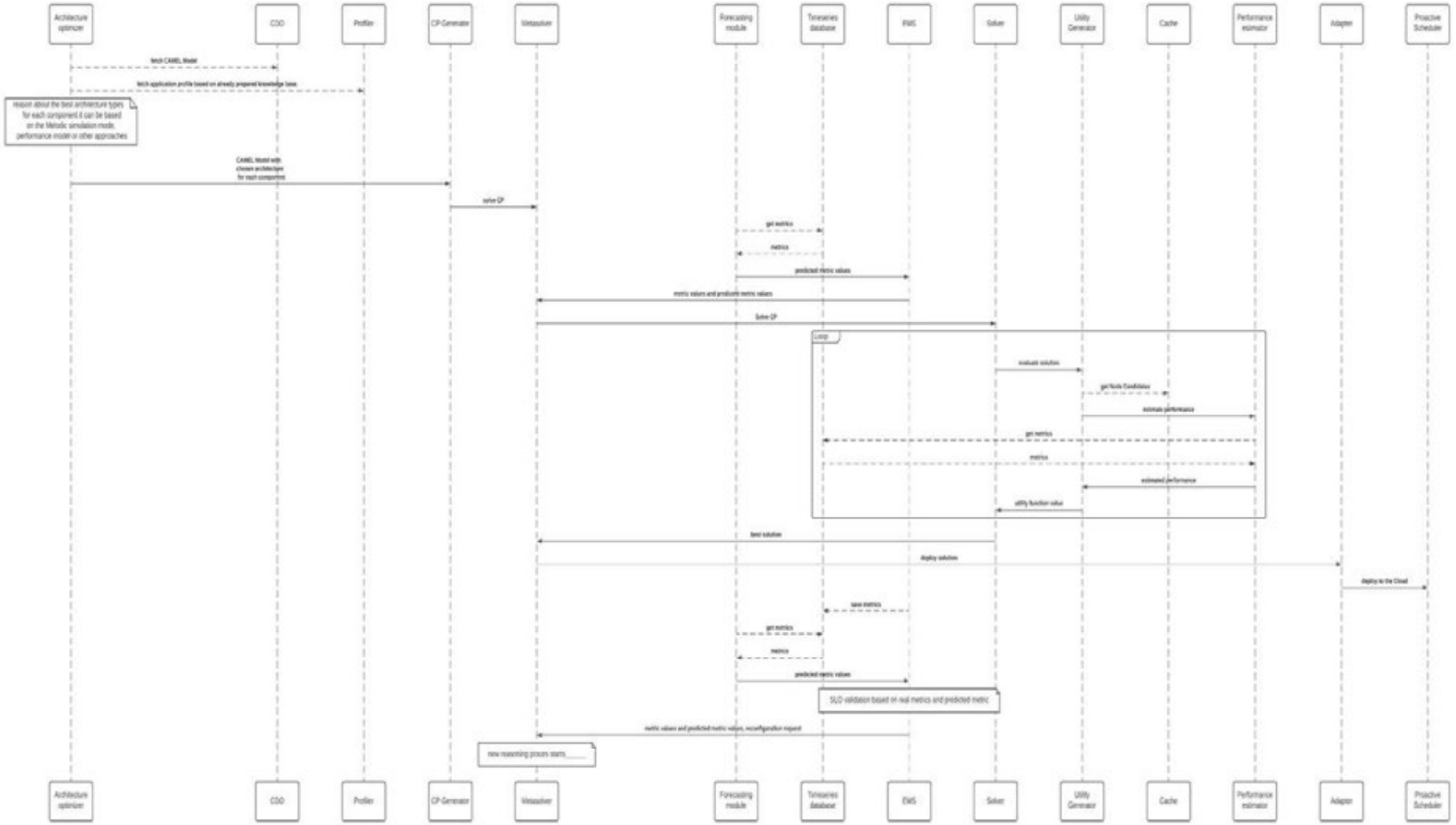


Figure 14: Architecture sequence - Reconfiguration

5 Summary

This deliverable was the outcome of the work conducted in all features of the MORPHEMIC platform and supplied an overview of its high-level component architecture. It presented three key areas of the MORPHEMIC project:

- The description and the presentation of the underlying frameworks, which are used by the MORPHEMIC pre-processor to formulate the integrated MORPHEMIC platform, together with the relation between the MORPHEMIC pre-processor and presented frameworks.
- The list of features of the MORPHEMIC project, with a brief description of each feature, purpose and a set of requirements that drove the design of the architecture of these features that should be respected by the integrated architecture of the Morphemic platform.
- The high-level architecture of the MORPHEMIC pre-processor. The architecture was presented using a UML component and sequence diagram. This approach allowed for the presentation of the complete high-level view of the architecture and two key processes in the system. In this way, both the initial deployment flow and reconfiguration flow have been presented. The architecture was presented at a high-level, using a top-down approach.

The upcoming work related to architecture will be the preparation of architecture and the detailed design of all new and updated components, as presented in section 4.2 and Table 3. Furthermore, the work on designing the integration layer and a detailed specification of the interface will be continued and the outcome will be presented in deliverable D4.3 “Selection, design and implementation of integration layer” - M18.

In general, this deliverable is served as a reference for the high-level software component architecture of the MORPHEMIC platform. It also provided the necessary guidelines for work that will be carried out by members of the consortium, as detailed in section 4.1.

6 References

[1] Nadareishvili, I., Mitra, R., McLarty, M., & Amundsen, M. (2016). *Microservice architecture: aligning principles, practices, and culture*. " O"Reilly Media, Inc.".
[2] Merkel, Dirk. "Docker: lightweight linux containers for consistent development and deployment." *Linux journal* 2014.239 (2014): 2.
[3] Masse, Mark. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. " O"Reilly Media, Inc.", 2011.
[4] Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., & Vrgoč, D. (2016, April). Foundations of JSON schema. In *Proceedings of the 25th International Conference on World Wide Web* (pp. 263-273).
[5] Li, Qing, and Yu-Liu Chen. "Entity-relationship diagram." *Modeling and Analysis of Enterprise and Information Systems*. Springer, Berlin, Heidelberg, 2009. 125-139.
[6] Pooley, Rob, and Peter King. "The unified modelling language and performance engineering." *IEE Proceedings-Software* 146.1 (1999): 2-10.
[7] Horn, G., Skrzypek, P., Prusiński, M., Materka, K., Stefanidis, V., & Verginadis, Y. (2019, October). MELODIC: selection and integration of open source to build an autonomic cross-cloud deployment platform. In *International Conference on Objects, Components, Models and Patterns* (pp. 364-377). Springer, Cham.
[8] Alessandro Rossini, Kiriakos Kritikos, Nikolay Nikolov, Jorg Domaschka, Frank Griesinger, Daniel ¨ Seybold, Daniel Romero, Michal Orzechowski, Georgia Kapitsaki, and Achilleas Achilleos, "The cloud application modelling and execution language (CAMEL)," OPEN Access Repositorium der Universitat¨ Ulm, p. 39, Mar. 2017. DOI: 10.18725/OPARU- 4339. (visited on 02/08/2018).
[9] Tobias Binz, Uwe Breitenbucher, Oliver Kopp, and ¨ Frank Leymann, "TOSCA: Portable automated deployment and management of cloud applications," in Advanced Web Services, Athman Bouguettaya, Quan Z. Sheng, and Florian Daniel, Eds., DOI: 10.1007/978- 1-4614-7535-4 22, Springer, New York, NY, 2014, pp. 527–549, ISBN: 978-1-4614-7534-7 978-1-4614- 7535-4. [Online]. Available: https://link.springer.com/ chapter/10.1007/978 - 1 - 4614 - 7535 - 4 22 (visited on 02/08/2018).
[10] Gordon Blair, Nelly Bencomo, and Robert B. France, "Models@run.time," Computer, vol. 42, no. 10, pp. 22–27, 2009, ISSN: 0018-9162. DOI: 10.1109/MC.2009. 326.
[11] D. E. Goldberg "Genetic Algorithms in Search, Optimization, and Machine Learning", 1989.
[12] David J. Earlab Michael W. Deem Parallel tempering: Theory, applications, and new perspectives, Physical Chemistry Chemical Physics, 2005.
[13] Cameron Browne, Edward J. Powley, Daniel Whitehouse et.al A Survey of Monte Carlo Tree Search Methods, IEEE Transactions on Computational Intelligence and AI in Games 4:1(1):1-43, 2012.
[14] Kritikos, Kyriakos, Paweł Skrzypek, and Marta Różańska. "Towards an Integration Methodology for Multi-Cloud Application Management Platforms." Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion. 2019.
[15] Ciro Formisano, Robert Gdowski, Adeliya Latypova, Ferath Kherif, Sebastian Geller, D6.1 "Industrial requirements analysis.
[16] D5.4 Accelerator resource manager and accelerators.