Project acronym:	**PrEstoCloud**
Project full name:	**Proactive Cloud Resources Management at the Edge for efficient Real-Time Big Data Processing**
Grant agreement number:	**732339**

D2.3 Conceptual Architecture

Deliverable Editor:	**Nenad Stojanovic, Nissatech**
Other contributors:	**ICCS, CNRS, Ubitech, ActiveEon, JSI, SAG**
Deliverable Reviewers:	**Ubitech, CVS**
Deliverable due date:	**30/09/2017**
Submission date:	**31/03/2018**
Distribution level:	**Public**
Version:	**1.4**

This document is part of a research project funded by the Horizon 2020 Framework Programme of the European Union

LIST OF CHANGES AS COMPARED TO 1ST SUBMISSION

Deliverable D2.3 is being resubmitted, in order to address the reviewers' recommendations following the outcome of the 1st interim review of the PrEstoCloud Project. In particular, the revisions aim at the rectification of the following recommendations:

- **Recommendation 1**

  "It is contradictory that the platform adopts a lambda architecture which uses a batch layer and real-time layer and the batch and real-time processing is not a requirement of the platform (FR-65). Please clarify"

- **Recommendation 2**

  "The components of the architecture are centralized and will run in one cloud, probably in the same region. The architecture should take into account the multicloud nature of the project and probably replicate most of the critical components both for availability and performance reasons."

- **Recommendation 3**

  "Please, revise the available assets"

- **Recommendation 4**

  "Mobile Context Analyzer available assets: "Scalable stream data storage and distributed stream processing."

- **Recommendation 5**

  "There is no storage for streams, please clarify. Later you mention ESPER, which is not scalable. On page 30 Storm is mentioned. Which CEP is going to be used?"

- **Recommendation 6**

  "Apache Calcite is not streaming engine, please update the deliverable."

- **Recommendation 7**

  "Page 31 "Open Source workflows scheduler (Java)", which one? Was it described in the SotA deliverable? The same kind of sentence is repeated several times along the document."

First, to address the 1st issue we would like to note that the mentioned requirement FR-65 is about the reconfiguration of the methods for data processing (real-time, batch), as described in the deliverable D2.2, page 38:

> *"The system should enable an easy reconfiguration of the data processing methods, usually done in the interaction with users (GUI)".*

It is defined as "Should Have" priority and it is not selected for the conceptual architecture, as depicted in Table 12. The requirement that is related to the batch and real-time processing is FR-45 in D2.2 (page 26):

> *"Ability to provide new methods for data processing (real-time, batch)".*

This requirement can be found in the mappings shown in Table 14, i.e. it is one of the main requirements for the Platform.

Our responses to the 2nd recommendation are included at the end of section 4.1 "Conceptual Architecture: high level view".

Regarding the 3rd recommendation, we assume that the exploitable assets are meant. As the exploitable assets are part of the exploitation planning, this issue has been already addressed in "WP8: Dissemination & Exploitation" in the deliverable „D8.7: Exploitation plan and reports - Iteration 1".

Regarding the 4th recommendation, we note here that the Mobile Context Analyser provides short-term stream data storage using Elastic search in order to perform context analysis. For details please refer to D3.5.1, "Mobile Context Analyser - Iteration 1", due March 2018.

As far as the 5th recommendation is concerned, several CEP engines will be reviewed and the one that fits best the PrEstoCloud requirements will be selected. This work will be reported in deliverable D5.1 "Situation Detection Mechanism - Iteration 1", due April 2018. Additionally, the Mobile Context Analyser provides short-term stream data storage.

Regarding the 6th recommendation, we removed the reference to Apache Calcite, because it is not relevant and we will not use it. Additionally, we replaced "Open source CEP engines (e.g. Esper)" with "Open source stream computing and complex event processing engines (e.g. Siddhi, Flink, Esper, Drools)".

Finally, our responses to the 7th recommendation are included at the end of section 5.2.1 "Autonomic Data Intensive Application Manager".

Change Log

Version	Date	Amended by	Changes
0.1	15/08/2017	Nissatech	1st version
0.2	20/08/2017	All technical partners	First comments
0.3	03/09/2017	ICCS, CNRS, ActiveEon, Ubitech, Nissatech, JSI	Initial inputs in all sections
0.5	11/09/2017	ICCS, ActiveEon, CNRS, Nissatech	Refinements, architecture
0.6	18/09/2017	ICCS, CNRS, ActiveEon, Ubitech, Nissatech	Refinements, details in each layers, interaction between components
0.8	29/09/2017	ICCS, CNRS, ActiveEon, Ubitech, JSI, Nissatech	Refinements, details about components
0.9	5/10/2017	ICCS, CNRS, ActiveEon, Ubitech, JSI, Nissatech	Mapping of the requirements
0.95	10/10/2017	Nissatech	Consolidated version
1.0	12/10/2017	Ubitech, CVS	Reviewed version
1.0	13/10/2017	SoftwareAG	Submission of the deliverable
1.1	07/02/2018	All technical partners	Analysis of the reviewers' recommendations following the outcome of the 1st Review
1.2	20/03/2018	Nissatech, ICCS, CNRS, ActiveEon	Review comments addressed
1.3	28/03/2018	Nissatech	Consolidated version ready for review
1.4	31/03/2018	Nissatech	Final version

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Acronym	Title
API	Application Programming Interface
BDVA	Big Data Value Association
CEP	Complex Event Processing
Dx	Deliverable (where x defines the deliverable identification number e.g. D1.1.1)
EDA	Event Driven Architecture
ETL	Extract Transform Load
GPS	Global Positioning System
GTP	General Trip Preferences
HTTP	HyperText Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
HPC	High-Performance Computing
I/O	Input / Output
ICT	Information and Communication Technologies
IoT	Internet of Things
JSON	JavaScript Object Notation
MOM	Message Oriented Middlelware
PubSub	Publish-Subscribe Mechanism
REST	Representational State Transfer
SQL	Structured Query Language
TB	Terabyte
WP	Work Package
XML	Extensible Markup Language

1. Executive Summary

This deliverable reports on the work done in WP2 in the context of developing a conceptual architecture that will satisfy very challenging requirements from the technical (deliverables D2.2 and D2.4) and use case (deliverable D7.1) point of view. The envisioned PrEstoCloud Platform sublimates ideas, based on the past work of partners from different technological areas, like Cloud computing, Big Data, Task scheduling, Adaptive systems and intends to create a novel computing and management infrastructure. It will enable the efficient deployment and realization of data-intensive applications, therefore the development of a proper architecture is a correspondingly challenging task.

However, through detailed and iterative analyses of the requirements and evaluation of the existing research results and relevant technologies, we did manage to create a conceptual architecture which is comprehensive enough to properly glue and harmonize different research ideas (around a common goal), but at the same time, technologically sound and based on the existing results of partners, in order to guarantee its efficient implementation and deployment in different infrastructures, starting from the selected use cases.

Indeed, by using a layer-based approach, we designed an architecture which enable complex processing within a particular layer and an efficient communication between layers in order to realize complex processing pipelines

In order to reflect the distributed nature of the system and the need to make the loose coupling between components, the data communication architecture postulates on the principles of event-driven architecture (EDA), having the communication and message broker as the central interaction hub. Moreover, EDA enables a proper scaling of the platform, by supporting an easy extension with new types of data sources (resources) and data processing elements. This is one of the fundamental properties for supporting the work with data-intensive applications (real-time big data applications) which are the focal point of the platform.

In parallel, through a powerful task scheduling and adaptation mechanism, the architecture supports an efficient detection of adaptation opportunities in the underlying computing infrastructure and an effective implementation of them, creating the basis for an adaptive management of complex applications. The adaptation mechanism covers the entire stack of processing resources, starting from edge devices till multi cloud infrastructure, making the Platform unique comparing to the state of the art.

Since PrEstoCloud aims to enable a continuous improvement (reconfiguration) of a complex computing infrastructure, the architecture is based on the well-known self-adaptivity pattern: Monitor – Analyze – Plan – Execute - Knowledge (MAPE-K model).

The deliverable explains the complex PrEstoCloud architecture through a comprehensive walkthrough, demonstrating a huge (innovative) application potential and the technological soundness. It contains details about all components and their interactions in order to realize complex scenarios, providing a foundation for the implementation phase. Last but not least, the deliverable shows in details how the functional requirements from the deliverable D2.2 are mapped to the presented architecture

We argue that the presented PrEstoCloud architecture is very innovative, enabling the realization of several advanced scenarios that are challenging for existing architectures in different domains, like:

- Fog computing - PrEstoCloud architecture enables dynamic service replacement along the entire fog computing infrastructure, from Edge to the Cloud and back
- Big data - the architecture enables continuous monitoring and improvement of the QoS in real-time data analytics applications
- HPC – the architecture enables self-adaptive reconfiguration of deployed computing infrastructure.

2. Introduction

2.1 Scope

The main goal of this deliverable is to provide details of the conceptual architecture that will serve as the basis for the technical development of the system. This work documented in this deliverable has been performed in the scope of WP2.

The deliverable is strongly influenced by several other deliverables:

- D2.1 that provides the analysis of the state of the art
- D2.2 that provides the list of requirements for the development of the system
- D2.4 that explains the data processing infrastructure
- D7.1 that describes use cases (and their requirements)

Since the envisioned Platform is very complex, this work also reflects the work of partners in different research and technology areas (like Cloud computing, Big Data, Task scheduling, Adaptive systems) and illustrates the connections to them, stating clearly our (unique) contributions.

This work will be used (and continued) in the work on the deliverable D6.1 Architecture of the PrEstoCloud platform.

2.2 Structure

The rest of the document is structured as follows:

- Section 3 describes the methodological approach that was adopted for the development of the conceptual architecture
- Section 4 present the conceptual architecture, including a walkthrough description of the main processing flows. Furthermore, the section elaborates on the advantages of the proposed architecture.
- Section 5 comprises detailed specifications for each layer. Each individual specification offers a description and explains the purpose of a component, describes the inputs and outputs and outlines its subcomponents.
- Section 6 elaborates on interfaces for each component. There are integrated components diagrams for each layer, as well as a set of tables that illustrate the interfaces and interconnections of the different elements of the platform.
- Section 7 validates the proposed conceptual architecture against the requirements defined in D2.2.
- Section 8 provides the conclusions and the summary of the document

3. Methodology

3.1 The base

Due to technological complexity of the envisioned system, as well as the very challenging requirements set by the use cases, the work on the conceptual architecture has been performed through an intensive collaboration of all partners (technical and use case providers).

The work method consists of an iterative refinement of the initial draft (cf. Figure 1) based on more elaborated use cases requirements (D7.1) and a deeper analysis of the data processing infrastructure (D2.4) driven by the list of requirements (D2.2).

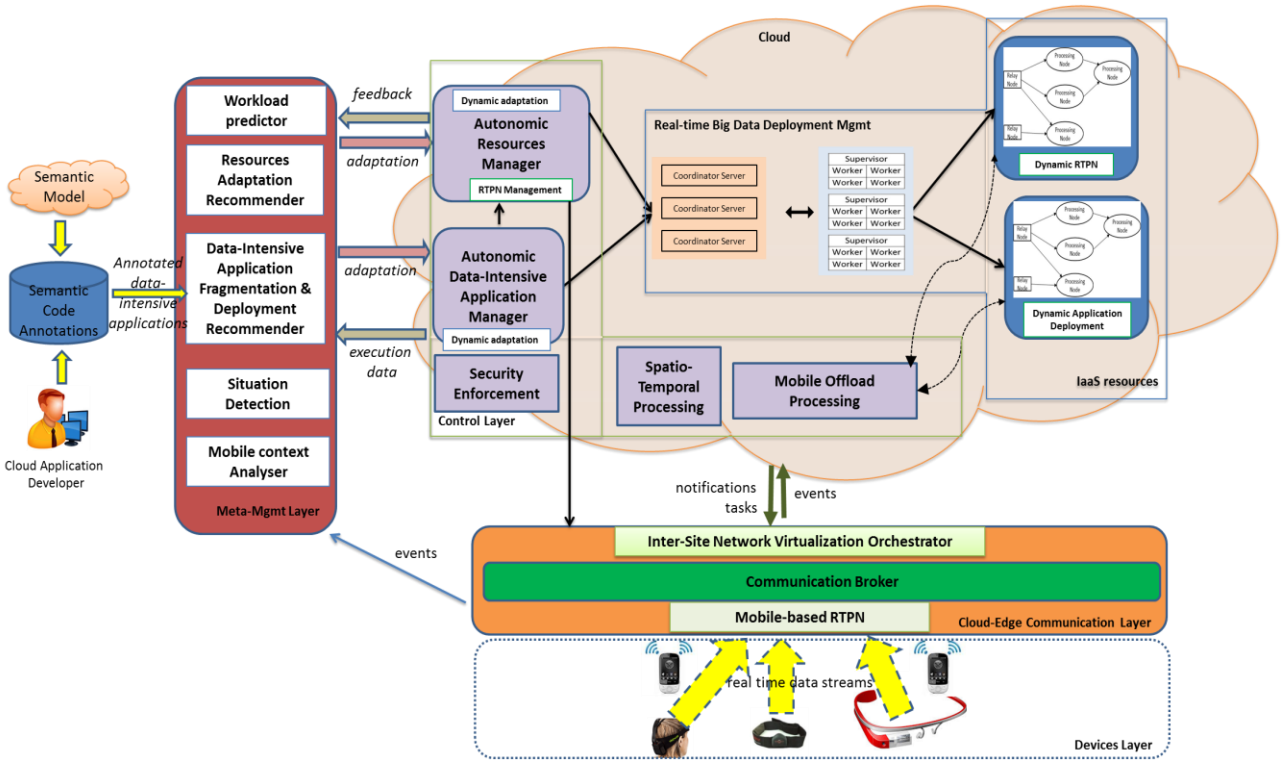


Figure 1: Initial conceptual architecture

There are three main directions for the refinement:

- clarifying the limitations of the existing results of partners that are used as the technological basis

- providing details of the functional specification of components based on advanced scenarios from use cases

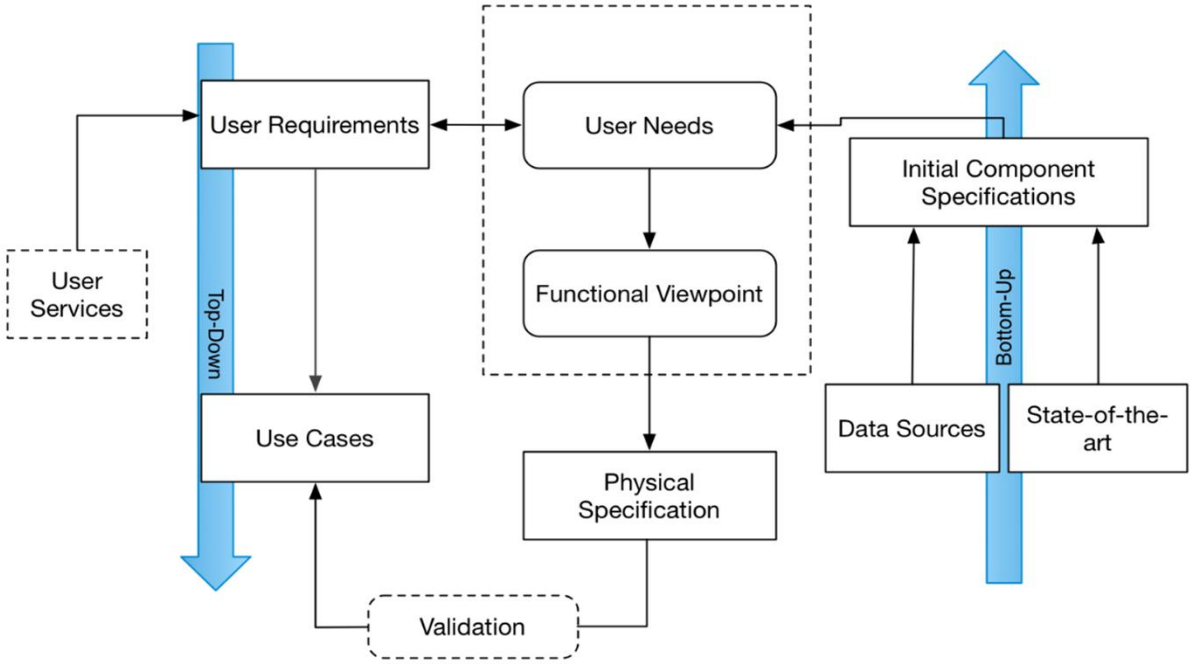
- getting better understanding how different layers can work together in a more efficient way in order to realize complex use case scenarios

3.2 The method

The overall methodology for the formalization of the architecture is an iterative combination of top-down and bottom-up approaches, as presented in Figure 2.

The bottom-up approach aims to the partial specification of the platform based on the intended functionality of the individual components and the identified innovations following the state-of-the-art analysis, as well as the availability of data sources at each pilot location.

On the other hand, the top-down approach enriches the specification using findings form the user



requirements analysis and use cases defined in D2.2 and D7.1 respectively.

Figure 2: Methodology for the formalization of the architecture

In this way, we have ensured that the work in WP2 is done in an efficient way and can exploit the results from other WPs properly.

One of the main benefits of using such a methodology is a very deep understanding of the processing pipelines (interactions between components, see Section 4.2 - walkthrough), which will make the technical realization of this complex system easier to be managed (less risks).

Moreover, as indicated in Figure 2, this methodology represents an efficient work practice for the further development and validation of the system.

Modeling Techniques

Currently, two approaches are in use for the architectural structure and processes — object-oriented analysis and design, and structured analysis and design. Both have strengths and weaknesses that make them suitable for different classes of problems; however, the object-oriented methodology is better for complex, interactive, and changing systems with many interfaces, which are the kinds of systems we aim to develop.

The object-oriented method takes a value-based approach to discover system capabilities. Use cases describe the behavior between the system and its environment. From the use case, the functionalities that the system must provide are derived. Those services are then realized by the internal structure of the system elements in iterative steps until system elements are simple enough to build. The resultant set of diagrams traces the composition of the system from its parts to the aggregated behavior captured within the set of use cases.

Object-oriented approaches focus on interaction from the beginning, which has the beneficial side-effect of defining the boundary between the system and its environment.

Sequence diagrams graphically illustrate the interactions the system must support. The "lifelines" of the diagram gather the behavioral responsibilities of each "object" participating in the use case. These responsibilities are the requirements to share data across the collection to produce the required result.

Component diagrams are used in order to describe the main PrEstoCloud components and depict their required and provided interfaces. This description reflects the way that these components are wired together for forming a more complex software system.

4. Conceptual Architecture

In this section we present the high level view of the architecture, describe the processing flow (walkthrough) and elaborate on the novelty of the architecture.

4.1 Conceptual Architecture: high level view

The overall physical architecture of the Platform consists of several components distributed across different layers. In the following text we describe these layers, while Figure 3 depicts the overall conceptual architecture of the platform.

The PrEstoCloud architecture has been structured across 5 different layers:

- The Meta-management layer mainly consists of decision logic capabilities required for enhancing the PrEstoCloud Control layer. Modules of this layer will use as input the situation details, the variation of the Big Data streams and the context of the mobile devices at the extreme edge of the network in order to recommend, at the appropriate time, the necessary adaptations of used resources in the real-time processing network.

- The Control layer manages resources of the Cloud infrastructure layer and contains modules which will monitor and manage cloud resources capabilities that can be extended to the edge of the network. Moreover, this layer is responsible for the scheduling of big data applications execution over the resources of the real-time processing network. The control layer detects available edge resources. The control layer selects target resources for deployment and plan application scheduling according to the recommendation of the meta management layer.

- The Cloud infrastructure layer will realize dynamic placement and scheduling capabilities allowing the utilization of the extreme edge of the network, deployed to private clouds, jointly with public clouds. The placement of microservices and application fragments will be handled based on placement constraints reflecting the requirements of the expected behaviour of the application. The infrastructure will realize constraints related to different properties like response time, security constraints or other provider wishes.

- The Cloud-Edge communication layer contains the inter-site network virtualization for coping with the need of connecting resources situated in multi-cloud environments and managing their orchestration and provisioning across different and heterogeneous providers. This also includes the control of the inter-sites network virtualization process in a secure way. This layer will also relay data streams on and off the PrEstoCloud platform providing standard publish/subscribe event brokering capabilities.

- The Devices layer consolidates any kind of device that can be used as a Big Data stream source or as a mobile computational node at the extreme edge of the network.

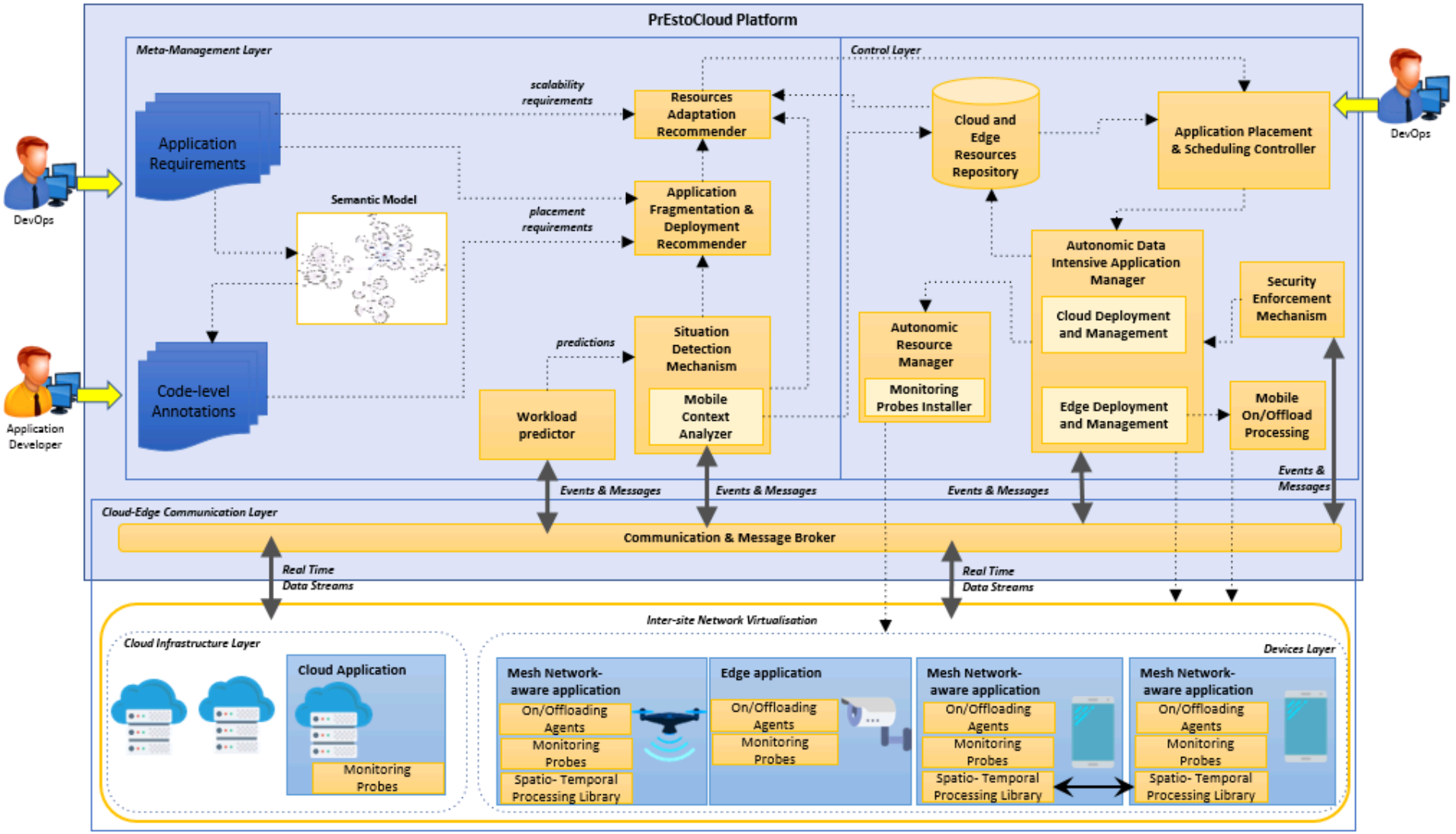


Figure 3: PrEstoCloud Conceptual Architecture

Below we include our responses regarding to the following reviewers' recommendation:

> ***Comment:*** *The components of the architecture are centralized and will run in one cloud, probably in the same region. The architecture should take into account the multicloud nature of the project and probably replicate most of the critical components both for availability and performance reasons.*

Out of the 15 components that are part of the PrEstoCloud architecture, 4 are distributed by design: On/Offloading Agents, Monitoring Probes, and Inter-Site Network Virtualization Gateways. These components are all included in the Cloud-Edge Communication Layer. The last component that is part of this particular layer, the Communication and Message Broker, will be implemented as a distributed broker, with one instance in each federated cloud.
As for the other components, they are part of the Meta-Management Layer and the Control Layer. They relate to job placement decision and actuation. Due to the nature of their work, they need to have a centralized view of the available resources, as well as a tight integration. For example, the Workload Predictor and the Mobile Context Analyzer infer context from the current situation of the application deployed by PrEstoCloud, either with respect to resource utilization (both, cloud and edge), or to job advancement. Based on the inferred situation, an adaptation of the deployment is recommended, e.g. by mapping certain jobs to certain kind of resources. These recommendations are then enacted by the Control Layer. These components work together so as to continuously optimize the resource usage and application status based on the evolution of the workload.
The deployment information (constraints, specification, location) are expressed and exchanged as a TOSCA file between each of these components, which is iteratively completed until the full deployment plan has been finalized. Each of these components will be implemented as a service that can be restarted, and whose state can be restored based on the last TOSCA specification, so as to avoid service failure due to an error within an individual component.

4.2 PrEstoCloud Architecture Walkthrough

This subsection delves into the details of the PrEstoCloud conceptual architecture (Figure 3) by providing a step-by-step usage walkthrough of the platform's main software components. In order to highlight the main functionalities of the PrEstoCloud components, we present their most important interactions clustered in three main control flow phases. These involve the:

1. design-time phase
2. initial application placement phase
3. application reconfiguration phase

During the walkthrough of each of these phases, we describe the main steps for creating, updating, placing, monitoring and reconfiguring a Big Data intensive application, using the PrEstoCloud platform. We note that the placement and reconfiguration of the Big Data intensive application, through PrEstoCloud, implies the necessary decisions and implementation actions for selecting and adapting the underlying cloud-based or edge-enabled infrastructures. Below, each of these phases are presented.

4.2.1 Design-time phase

The design-time phase involves all the preparatory actions that may be performed through PrEstoCloud for enabling initial placement decisions. During this phase two important actions can be performed with the assistance of PrEstoCloud components: i) the annotation of a cloud application and ii) the definition of its placement and scalability requirements.

Step 1a: Annotation of the Cloud Application

i. The Cloud Application Developer annotates each method of the application with certain metadata, coming from the PrEstoCloud semantic model, in order to indicate its processing complexity as well as any potential deployment constraints (e.g. can be deployed only cloud resources, edge resources could be used). This metadata will subsequently guide the placement and the reconfiguration of the Big Data intensive application.

Step 1b: Definition of Application Requirements

i. In parallel the DevOps formulates the application-specific placement and scalability requirements that are used as input for the Application Fragmentation and Deployment Recommender. These requirements correspond to SLA-based constraints, preferences and scaling requirements that should be considered during the initial placement or reconfiguration of the application.
ii. Once the Application requirements have been formulated the Application Fragmentation and Deployment Recommender introspects and stores them.

Step 1c: Definition of Initial Infrastructure

In addition, the DevOps provide the initial (physical and/or virtual) infrastructure on which the initial deployment will be carried out in the Cloud and Edge Resources Repository. For example, this includes the credentials needed for the Autonomic Resource Manager to connect and authenticate on public and private cloud platforms, as well as means to contact the edge devices.

4.2.2 Initial Placement phase

The initial placement phase involves the interaction of all the appropriate PrEstoCloud components for making initial placement decisions, implementing them and setting up monitoring mechanisms for following up on the deployment status over cloud and edge resources.

Step 1: Initial Placement Decisions

i. The Autonomic Resource Manager collects information about all available Cloud resources (i.e. flavours and instances) and Edge devices (i.e. types and specific devices), and submits it to the Cloud and Edge Resources Repository.
ii. The Mobile Context Analyzer aggregates and infers information about the health status of devices that could be considered for initial application fragment deployment.
iii. The Application Fragmentation & Deployment Recommender, based on the expressed application requirements, registers the possible application fragments (if applicable) that can be deployed on separate resources and issues a recommendation per each fragment (or the whole application if it is not possible to be fragmented) about the flavours or types of cloud and/or edge devices that can be used for deployment.

iv. Additionally, the Application Fragmentation & Deployment Recommender generates a set of dependencies and requirements that span the boundaries of cloud resources and reach the extreme edge of the network. Depending on the metadata of the application and its fragments and based on the context of the edge devices (inferred by the Mobile Context Analyzer) certain edge resources may be deemed as appropriate or inappropriate for hosting certain application fragments. In case that an acceptable placement recommendation cannot be issued given the described requirements and the available cloud and edge

resources the process is aborted and the application developer or the DevOps is informed about refining the described requirements.

v. The recommendations and requirements thereby generated by the Application Fragmentation & Deployment Recommender are sent to the Resources Adaptation Recommender in order to be expressed as an extended TOSCA specification. They make up for a partial specification that will be completed by the Application Placement and Scheduling Controller.

vi. The Application Placement and Scheduling Controller receives the partial TOSCA specification file, and contacts the Cloud and Edge Resources repository to identify the optimal configuration of the Big Data intensive application (i.e. commission VM instances on a cloud provider or select specific edge devices). Based on the available resources and existing constraints, the Application Placement and Scheduling Controller completes the extended TOSCA specification by specifying specific resources to be used. This information is send to the Autonomic Data-Intensive Application Manager.

Step 2: Initial placement implementation

i. Upon the reception of the completed extended TOSCA specification file, the Autonomic Data Intensive Application Manager parses it and extracts the requested resources. The Autonomic Data Intensive Application Manager contacts the Inter-site Network Virtualization to obtain the necessary network configuration to be applied to the resources. The Autonomic Data Intensive Application Manager creates a so-called "commissioning workflow".

ii. The Autonomic Resource Manager receives the commissioning workflow and creates the ad-hoc environments (e.g. VMs on clouds, containers on edge devices) inside which the Big-Data tasks will be inserted. The Cloud and Edge Resources Repository is updated (through the Autonomic Data Intensive Application Manager) to reflect the instantiation changes (e.g. adding new VMs).

iii. The Autonomic Data Intensive Application Manager implements a sequence of deployment actions. This corresponds to the enactment of a workflow consisting of processing tasks to be executed on every infrastructure node. These may include the acquisition of necessary libraries and executables, and the initialization of services. In case of edge resources used the appropriate On/Offloading agents are triggered for assigning to them processing application tasks.

iv. In parallel, the Security Enforcement Mechanism is contacted in order to ensure the security of all connections between the processing resources that will be part of the deployment topology.

Step 3: Monitoring setup

i. Once the core elements of the business logic of the application have been instantiated, the Autonomic Resource Manager instructs the installation of monitoring probes on each processing resource. These monitoring probes are essential for following up on the health status of each of the commissioned resources. The Cloud and Edge Resources Repository is updated (through the Autonomic Data Intensive Application Manager) to reflect any changes in the available resources (e.g. registering a new edge device, updating edge devices status).

ii. The Autonomic Resource Manager is configured to collect information about all Cloud Infrastructures used (i.e. flavours and instances) and Edge devices acquired

(i.e. types and specific devices), and submit it to the Cloud and Edge Resources Repository (through the Autonomic Data Intensive Application Manager).

iii. Upon the detection of a new edge device, the Autonomic Data Intensive Application Manager submits a related event to the Communication & Message broker, to be received by any interested Meta-Management layer component (e.g. Mobile Context Analyzer).

4.2.3 Reconfiguration Phase

The reconfiguration phase involves the whole control flow that engages appropriate PrEstoCloud components for proactively or reactively adapting a certain Big Data intensive application placement.

Step1: Feedback

i. The Monitoring Probes installed during the Initial Placement Phase, periodically send data concerning the health of their host devices. Additionally, Monitoring Probes may send data concerning the state of the Application which can also trigger a reconfiguration of the topology. The data is sent in the form of events and messages, to the Communication & Message Broker from which it can be distributed to any PrEstoCloud component.
    - As mentioned in the monitoring setup this also includes the detecting of recently added edge devices.

ii. In parallel any streaming data coming from the Communication & Message Broker is persisted in order to be used by the Workload Predictor for learning and deriving future predictions about expected workload fluctuations. Such workload fluctuations may lead to meaningful application reconfigurations for quality assurance.

Step 2: Reconfiguration Decision

i. A series of events transmitted by the processing topology, or certain messages coming from the Workload Predictor may trigger an application reconfiguration. Based on this the Situation Detection Mechanism notifies and triggers the Resources Adaptation Recommender. In the case of edge resources, the Mobile Context Analyzer is responsible for detecting the health status of devices already used, but also to track potential candidate edge resources to be used after a reconfiguration.

ii. Based on the detected situation, the Application Fragmentation & Deployment Recommender informs the Resources Adaptation Recommender about certain Application Fragments that can be offloaded from edge devices to cloud resources and vice versa (onloaded), along with their dependencies and requirements.

iii. The Resources Adaptation Recommender evaluates the Situation detected in conjunction with the application requirements and issues a recommendation about an update on the processing topology. This information is sent as a partial TOSCA specification to the Application Placement and Scheduling Controller for verifying the optimality of the new topology or enhancing the cloud resources instances to be used (e.g. instead of scaling out using two m1.medium VMs, use one m1.large).

iv. The Application Placement and Scheduling Controller contacts the Cloud and Edge Resource Repository. Based on all existing resources and the new constraints generated by the Resource Adaptation Recommender, it generates an optimal placement, as well as a list of transitional atomic actions to migrate the current (existing) infrastructure to the new optimal one.
v. A special kind of reconfiguration may take place in case the topology involves edge devices in proximity. In cases where connection to the cloud is broken or completely lost, these devices search for other devices by leveraging the Spatio-Temporal Processing Library and trying to create a Mesh Network.

Step 3: Reconfiguration Implementation

i. In the case that a new extended TOSCA specification document is sent by the Resources Adaptation Recommender, the Application Placement and Scheduling Controller contacts the Cloud and Edge Resources Repository, and the Autonomic Data Intensive Application Manager to deploy the changes in the architecture.
ii. In the case of poor edge connectivity the devices communicate in a local Mesh Network and exchange messages in order to fine-tune their processing activities. Once Internet connectivity is restored, any intermediate processing results are sent to the Mobile On/Offloading processing component which adds the Devices to the Cloud and Edge Resources Repository and the Mesh Network is dissolved.

4.3 Advantages

We argue that the presented architecture is not "only" suitable for the realization of complex scenarios (as given by selected use cases), but can also serve as an important contribution in the further development of the modern software architectures.

We argue that the presented architecture is an important contribution to the design of the modern data-intensive systems, especially those dealing with the dynamically changing real-time big data, which requires:

- capabilities for processing big data, in order to cope with huge real-time streams and
- ability for self/adaptivity, in order to cope with the dynamicity in the data

In the following two subsections we elaborate on these two characteristics.

4.3.1 Extending the Lambda architecture for Big data processing

Lambda architecture[1] is a data-processing architecture designed to handle massive quantities of data by taking advantage of both batch- and stream-processing methods. It is divided into three processing layers: the batch layer, serving layer, and speed layer, as shown in the Figure 4.

[1] https://en.wikipedia.org/wiki/Lambda_architecture

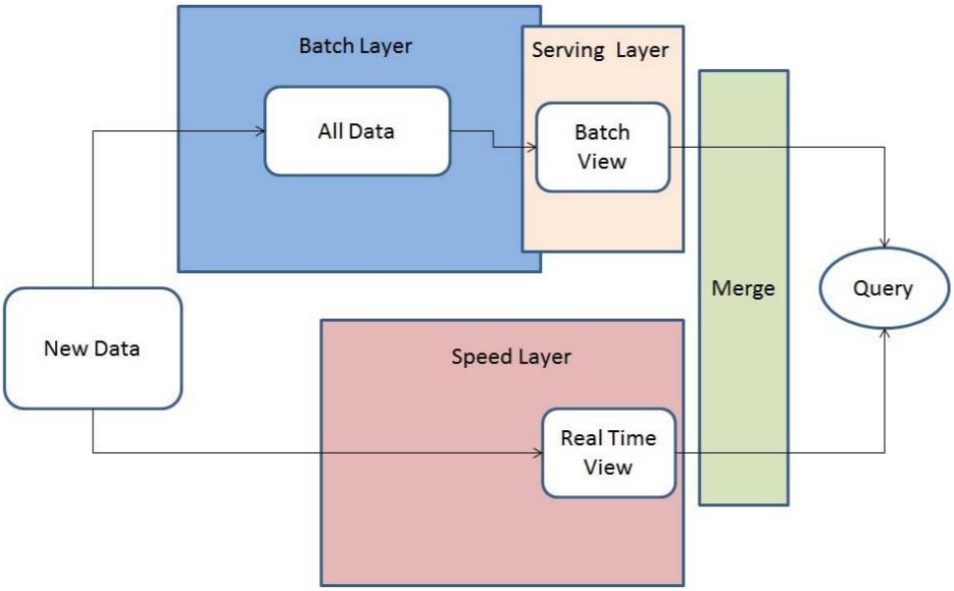


Figure 4: Lambda architecture

Figure 5 illustrates a part of the architecture that implements Lambda pattern.

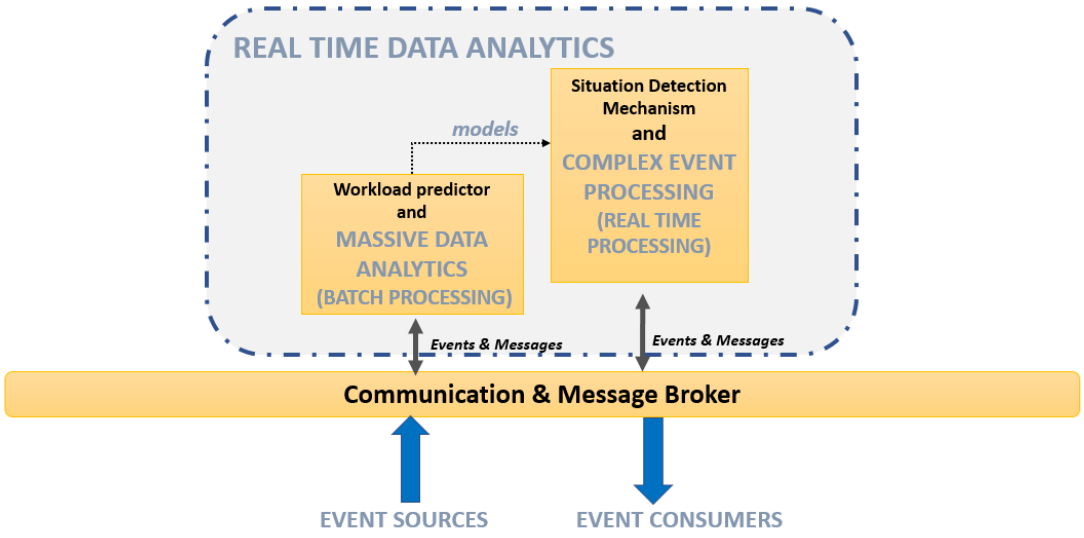


Figure 5: Lambda architecture design pattern adopted in the PrEstoCloud architecture

Moreover, if we put this pattern in a broader context as illustrated in Figure 6, the role of the entire architecture becomes clearer: it enables monitoring and management of the QoS (or SLA) for the big data system.

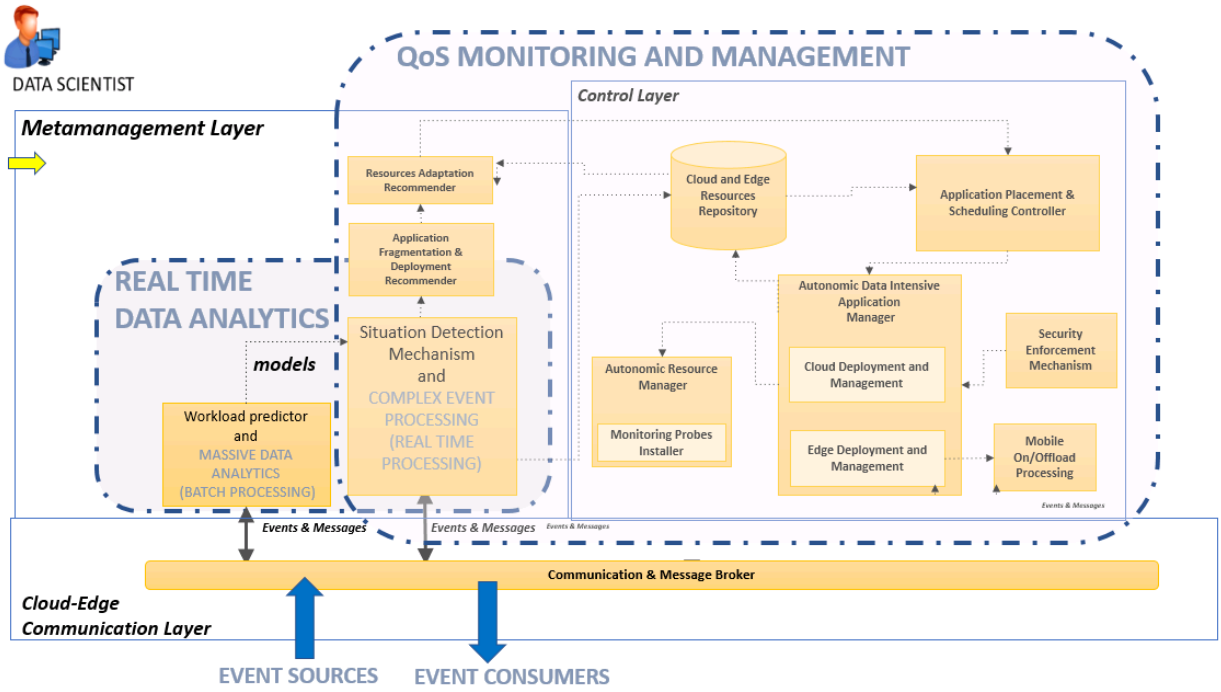


Figure 6: Enabling management of QoS of a big data system

4.3.2 Self-adaptive architecture: Realizing MAPE-K

Since PrEstoCloud aims to enable a continuous improvement (reconfiguration) of a complex computing infrastructure, the conceptual architecture follows, on a higher abstraction level, the self adaptivity pattern: **Monitor Analyze Plan Execute** Knowledge [IBM05] (MAPE-K model; so called "architectural blueprint for autonomic computing", introduced by IBM).

Figure 7 illustrates the mapping between these layers and MAPE-K model.

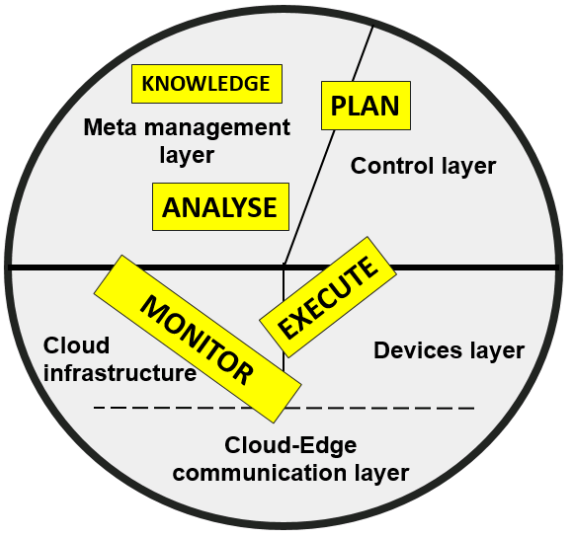


Figure 7: Mapping to MAPE-K model

Figure 8 provided below illustrates the mapping of MAPE-K to the presented conceptual architecture of PrEstoCloud.

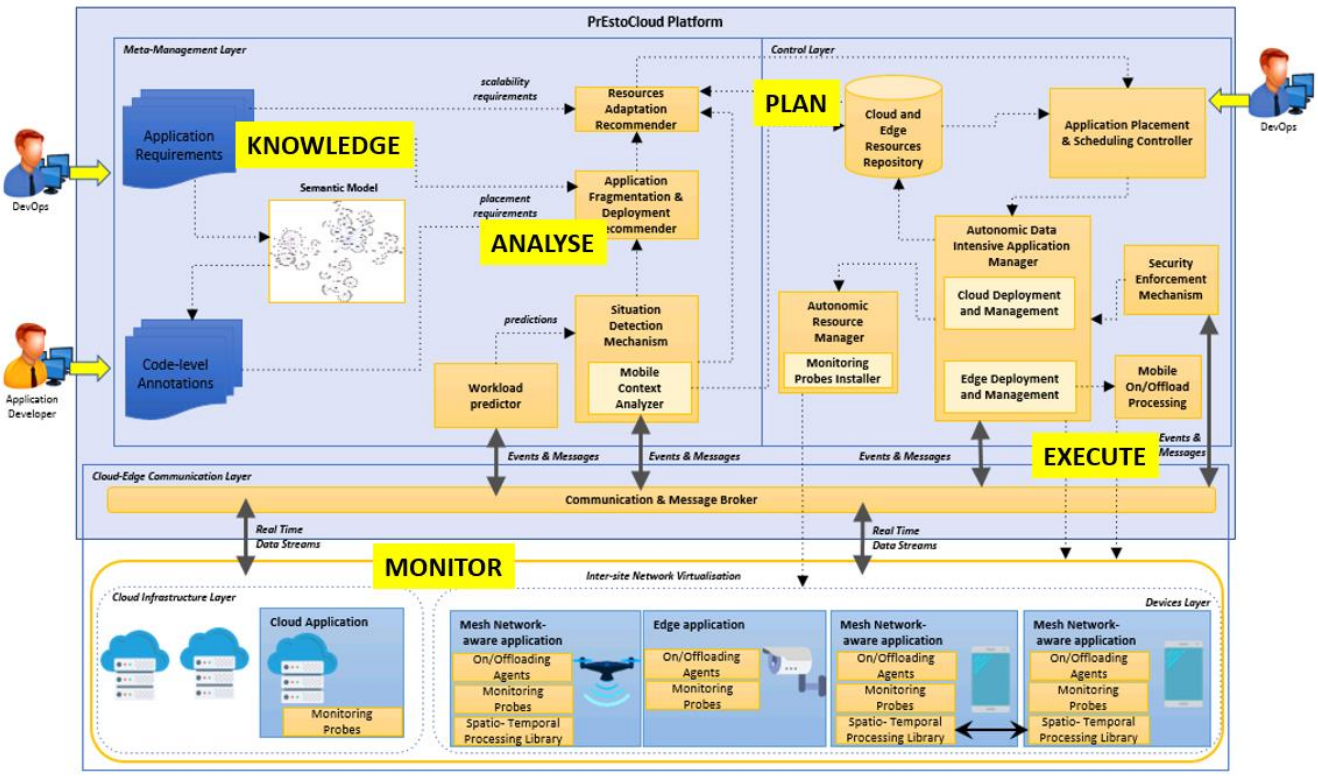


Figure 8: MAPE-K phases mapped to the architecture

4.3.3 Innovations

The architecture enables the realization of several advanced scenarios, which seem to be challenging for existing architectures in following domains.

High Performance Computing - HPC

PrEstoCloud architecture enables **self-adaptive reconfiguration** of deployed computing infrastructure:

- starting from detecting/predicting the need for changes,
- through analyzing all available factors (context) for recommending the most efficient reconfiguration and
- defining the optimal redeployment,
- till automatically assigning required resources to the running processing (tasks)

Fog computing

PrEstoCloud architecture enables **dynamic service replacement** along the entire fog computing infrastructure, from Edge to the Cloud and back, through:

- continuous monitoring of the status of the computing resources and the application context
- detecting/predicting the decrease in performance (increase in resources)
- deciding when and where to offload the service
- realizing the replacement in the most efficient way

Big data

PrEstoCloud architecture enables **continuous monitoring and improvement of the QoS** in real-time data analytics applications, through:

- continuous monitoring of the performance data required for QoS
- prediction of the situation where QoS will be violated
- real-time recommendations for the reconfiguration
- optimal realization of the reconfiguration (based on the global context)

Some of the work related to dissemination and communication will be in creating awareness about our work (esp. conceptual part) in relevant communities.

5. Components Specifications

This section details every component included in the PrEstoCloud architecture described in previous section. The description of every component relies on the following fields:

- **Function**: Describes the purpose and main role of the component within the PrEstoCloud architecture
- **Subcomponent**: Lists the components (if applicable) that are running inside the described component.
- **Sources**: Lists the components that provide data or any other input to the described component.
- **Consumer**: Lists the components that feed from the activities or data produced by the described component.
- **Responsible Partners:** The partner or partners that are responsible for the implementation of the specific artifact.
- **Available assets**: Describes the available assets (tools, methodologies, techniques, etc.) that are foreseen to be used for the development of the described component.

The following subsections describe the components grouped by their respective layers.

5.1 Meta-Management Layer

The Meta-Management layer mainly consists of decision logic capabilities required for enhancing the PrEstoCloud Control layer (e.g. Autonomic Resources Manager, Autonomic Data-Intensive Application Manager). This layer involves the following modules: Workload Predictor, Mobile Context Analyzer, Situation Detection Mechanism, Application Fragmentation & Deployment Recommender (DIAFDRecom), and Resources Adaptation Recommender (RARecom).

5.1.1 Workload Predictor

Good resource management is very important in the cloud and workload prediction is a crucial step towards achieving good resource management. While it is possible to predict the workloads of long-running tasks based on the seasonality in their historical workloads, it is difficult to do so for tasks which do not have such recurring workload patterns.

This component realizes a novel workload prediction approach that takes the statistical properties of a pool of tasks to help predict the workload patterns of new tasks. Our scheme integrates clustering and machine learning to maximize the effectiveness of learning and, hence, improves workload prediction accuracy. The main benefit is the possibility to process high-dimensional data and can be applied directly to memory, disk bandwidth, and network bandwidth demand predictions.

Our approach learns the models of the normal behaviour of different tasks. This is crucial step in enabling a proper understanding of the characteristics of a task and consequently to ensure better modelling of predictions. If using the whole dataset, the predictions are imprecise (too coarse grained).

One of the most important advantage is a strong clustering approach that works in high dimensional spaces. It generates the best possible separation of the problem space. For example, the clusters are compact and differ from each other.

Workload predictor	
Function	• Receive, pre-process and store the data relevant for workload prediction • Understand and model the types of the workloads based on the provided data (infrastructure, applications) • Predict the workload of the underlying cloud infrastructure • Enable refinement process (self-adaptation)
Subcomponents	Event listener, Pre-processor, Storage Clustering engine Prediction engine Real-time predictor
Input	• Monitoring sensors from cloud and edge resources • Application data (profile)
Output	• Real-time predictions
Sources	Communication & Message Broker (monitoring data)
Consumers	Situation Detection Mechanism
Beyond SOTA	Data-driven (unsupervised) modelling of the normal behaviour of an underlying system (computing infrastructure) and its application in real-time for diagnosis and predicting the behaviour
Responsible Partners	Nissatech
Available assets	• D2Lab framework • Open source CEP engines (e.g. Siddhi)

5.1.2 Mobile Context Analyzer

The Mobile Context Analyzer will perform an efficient analysis of the status of edge devices based on machine learning techniques and the PrEstoCloud semantic model. Some indicative analysis this component will perform is shown below:

- *When* battery level = 60% & job type to be onloaded = "intensive" *Then* battery status = critical
- *When* battery level = 60% & job type to be onloaded = "not intensive" *Then* battery status = normal
- *IF* Battery status = critical *Then Infer* that Edge device = 'not adequate for this processing job'
- *IF* Battery status = normal *Then Infer* that Edge device = 'adequate for this processing job'

The Mobile Context Analyzer will receive as inputs historical data coming from edge devices (types), edge related attributes, types of processing jobs running. Moreover, it should receive real-time data such as regularly pushed events from edge devices about QoS variations and pulled data from edge devices for certain parameters, as battery, location, network capacity etc. It will generate as outputs a correlation between edge related attributes, types of jobs running and "observed" QoS variations that will constitute the inferred context about the current status of edge devices, e.g., an estimate of the task execution time on edge device or an estimate of the remaining battery lifetime for edge device where it is applicable.

The Mobile Context Analyzer will go beyond the state of the art through a combination of stream processing-based machine learning with semantic inferencing.

Mobile Context Analyzer	
Function	• Collects monitoring data from edge devices and infers context • Can relate current context to the processing capacity of a device, e.g. expected execution time, battery level
Subcomponents	Event Database and Event Analyzing Service
Input	• Events
Output	• Per-device context inferred from monitoring data • Capability of edge device to process a given task
Sources	Communication & Message Broker (monitoring data)
Consumers	Situation Detection Mechanism, Autonomic Resource Manager
Beyond SOTA	Combination of stream processing-based machine learning with semantic inferencing
Responsible Partners	ICCS
Available assets	• Prototype android application in-development • Open source MQTT brokers • Scalable stream data storage and distributed stream processing

5.1.3 Situation Detection Mechanism

The Situation Detection Mechanism detects interesting situations that might lead to resources adaptation or data-intensive application reconfiguration or redeployment. The Situation Detection Mechanism receives as inputs the context of edge device, monitoring data from cloud & edge resources, the current workload (e.g. current throughput, volume) as well as workload predictions (e.g. predicted throughput, volume). It generates as outputs the detected situation and its associated context conditions.

The Situation Detection Mechanism will go beyond the state of the art by delivering a situations detection approach based on smart event subscriptions, workload predictions and inferred context in order to cope with the dynamicity of Big Data.

Situation Detection Mechanism	
Function	• Detects any interesting situations (that might reveal adaptation opportunities) during the usage of the Cloud Application under certain context conditions. • Triggers adaptation of the processing topology • Guides the deployment of application fragments
Subcomponents	Event listener, complex event processing engine
Input	• Monitoring sensors from cloud and edge resources, inferred context from edge resources, application workload predictions
Output	• Interesting situations (that might reveal adaptation opportunities)
Sources	Mobile Context Analyzer, Communication & Message Broker (monitoring data), Workload Predictor
Consumers	Application Fragmentation & Deployment Recommender, Resource Adaptation Recommender

Beyond SOTA	Delivery of a situations detection approach based on smart event subscriptions, workload predictions and inferred context, coping with the dynamicity of Big Data
Responsible Partners	ICCS
Available assets	• Situation Action Network (SAN) Engine • Open source stream computing and complex event processing engines (e.g. Siddhi, Flink, Esper, Drools)

5.1.4 Application Fragmentation & Deployment Recommender

The Application Fragmentation & Deployment Recommender will recommend the appropriate fragmentation of cloud applications into smaller parts that can be efficiently deployed over cloud / edge resources. Moreover, it will associate applications and application fragments with placement constraints and optimization preferences. Examples of information that this mechanism may dispatch to the rest PrEstoCloud components are:

- Fragment 1 *must run on* a VM with RAM > 4 GB
- Fragment 2 *may run on any* edge device
- All fragments *should be placed under* the same availability zone.

The Application Fragmentation & Deployment Recommender will receive as input the available VM flavours & edge devices as well as the qualitative, quantitative preferences of the DevOp, Application developer in order to formulate the optimization function. It will generate as output a recommended fragmentation along with a recommended deployment serialized in a TOSCA-based specification (without specific VM and edge instances). The recommendation will take into consideration constraints such as security constraints or other quantitative or qualitative constraints, e.g. cost, response time, data sanitization support etc. The recommendation will be forwarded to the Resource Adaptation Recommender, which will then use them to reactively or proactively (in case the situation detected involves workload predictions) invoke the Application Placement & Scheduling Controller to find an optimal solution by resolving a constraint programming problem. Based on this, it will complete the TOSCA specification and dispatch it to the Autonomic Data Intensive Application Manager to deploy it.

The Application Fragmentation & Deployment Recommender will go beyond the state of the art by extending existing approaches which focus primarily on either cloud resource management or off-loading between cloud and edge resources. Moreover, it will combine constraint programming and AI or rule-based approaches to seamlessly deploy application fragments on cloud or edge resources.

Application Fragmentation & Deployment Recommender	
Function	• Fragments a Cloud Application into smaller processing parts, in order to facilitate deployment on cloud and edge resources • Deploys the Cloud Application on cloud and edge resources
Subcomponents	Annotation Processing Service, Fragments Generator, Abstract Deployment Generator
Input	• Interesting situations detected from the Situation Detection Mechanism, the Application placement requirements and annotations
Output	• Recommendation on the Deployment of application fragments
Sources	Annotations, Placement Requirements, Situation Detection Mechanism, Cloud & Edge Resources Repository
Consumers	Resources Adaptation Recommender

Beyond SOTA	Extension of the existing approaches which focus primarily on either cloud resource management or off-loading between cloud and edge resources. There will be a combination of constraint programming and AI or rule-based approaches. Additionally application fragments will be seamlessly deployed on cloud or edge resources
Responsible Partners	ICCS
Available assets	• Java Parallel Processing Framework (JPPF) • React Native

5.1.5 Resources Adaptation Recommender

The Resources Adaptation Recommender will provide context-aware, edge-cloud adaptation recommendations that include changes to the used cloud resources, the used edge devices and the applications or application fragments placement. It will receive as input the current processing topology and placement, the detected situations along with the respective context of the used and the available edge devices. It will generate as output the adaptation recommendation to reconfigure the processing topology, e.g., to introduce new processing nodes, replicate nodes for failover purposes, remove redundant or underused processing nodes. Moreover, it will recommend how to alter the processing topology on cloud resources, e.g., to reconfigure additional VMs that host existing processing nodes, span VMs to new physical machines to deal with failover, start new containers or deploy on additional hosts in a cluster. Finally, it will shift processing effort to/from resources at the extreme edge of the network.

The Resources Adaptation Recommender will go beyond the state of the art by implementing context-aware edge-cloud adaptation recommendation algorithms that estimate additional factors that may influence adaptation recommendation decisions (task/VM migration time, cost etc.). Moreover, it will find feasible adaptation recommendations seamlessly among edge devices and cloud resources based on generic scalability requirements.

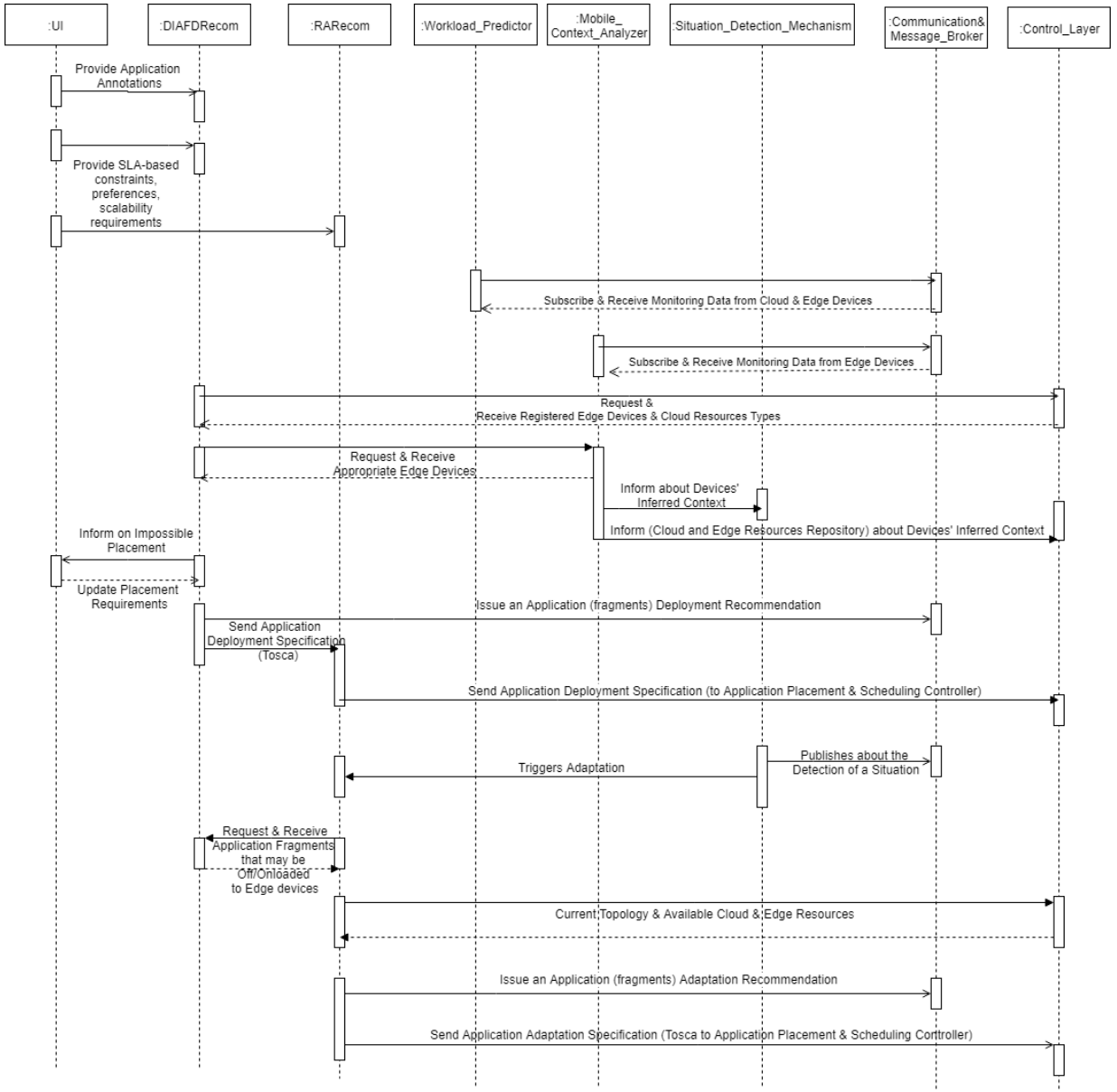
Resources Adaptation Recommender	
Function	• Recommends adaptations on the application deployment topology, in order to meet the application requirements and deployment constraints.
Subcomponents	Resources Adaptation Engine
Input	• The current situation from the Situation Detection Mechanism and the current application deployment by the Application Fragmentation and Deployment Recommender
Output	• An extended TOSCA specification document, describing the new application deployment topology
Sources	Application placement requirements, Situation Detection Mechanism, Cloud & Edge Resources Repository
Consumers	Application Placement & Scheduling Controller
Beyond SOTA	Context-aware edge-cloud adaptation recommendation, estimating additional factors that may influence adaptation recommendation decisions (e.g task/VM migration time, cost). Feasible adaptation recommendations among edge devices and cloud resources will be found seamlessly, based only on generic scalability requirements
Responsible Partners	ICCS

Available assets	• Service Adaptation Recommender (SAR)

5.1.6 Meta-Management layer: an integrated view

In the Figure 9, we present a coarse-grained sequence diagram that depicts the most important interactions, as explained above, between the main PrEstoCloud components that constitute the Meta-Management Layer. We note that for completion and diagram readability reasons, we also consider interactions that span the boundaries of the Meta-Management layer (i.e. Communication & Message Broker, Control Layer) but we depict them without providing any details on their corresponding components (as these will be provided below in this section).

Figure 9: PrEstoCloud Meta-Management Layer – Sequence Diagram



5.2 Control Layer

The Control layer mainly consists of managing PrEstoCloud resources and applications. Based on recommendations from the Meta-Management Layer, the Control Layer is in charge of computing initial and new resources placements, deploying and monitoring Cloud and Edge resources, and managing Data Intensive applications.

This layer involves the following modules: Application Placement & Scheduling Controller, Mobile On/Offloading Processing, Autonomic Data Intensive Application Manager, Autonomic Resource Manager, Cloud & Edge Repository, and Security & Enforcement Mechanism.

5.2.1 Autonomic Data Intensive Application Manager

The Autonomic Data Intensive Application Manager (ActiveEon) is a workflow scheduler that rely on the Autonomic Resource Manager to execute commissioning and deployment tasks on PrEstoCloud resources.

The Autonomic Data Intensive Application Manager will receive as input an extended TOSCA specification file from the Application placement & Scheduling controller. Based on the TOSCA file received, the Autonomic Data Intensive Application Manager extracts the specifications and the desired placement of the resources to their respected cloud infrastructure. Then, the Autonomic Data Intensive Application Manager queries the Inter-Site Network Virtualization component to retrieve all relevant network configurations of the resources to deploy or reconfigure.

When actions on resources are required (e.g. new resources must be acquired, existing resources must be released) then a commissioning workflow is generated. The workflow mainly consists to query the Autonomic Resource Manager to deploy and acquire new resources from public or private clouds.

Note: To 'acquire' a resource, the Autonomic Resource Manager installs a 'Java agent' (ActiveEon technology) on the resource itself (using clouds APIs) to be able to inject and execute tasks remotely, and also to monitor the resource itself.

Finally, once all resources actions are performed, a deployment workflow is generated and executed. This workflow consists to orchestrate application specific tasks on remote resources (task examples: "Add a resource to a specific VPN", "Create a new Storm topology", "Add a 'bolt' to an existing Storm topology", etc.) by communicating with the resources' agents previously installed.

The Autonomic Data Intensive Application Manager will go beyond the state of the art by controlling and mutating remote Event Processing Network (EPN) topologies from heterogeneous deployment workflows (e.g. configuring resources to be part of a new Storm topology, updating network configurations, etc.). Furthermore, its own internal Monitoring system will go beyond the state of the art by extending the existing set of metrics to edge devices (battery, network antenna, etc.).

Autonomic Data Intensive Application Manager	
Function	• Generate and schedule commissioning workflows to manage resources (deploy, acquire and release resources) • Generate and schedule deployment workflows to deploy and manage data intensive applications • Monitor resources by subscribing to monitoring events on Communication & Message Broker
Subcomponents	Cloud Deployment and Management, Edge Deployment and Management
Input	• Complete TOSCA specification and reconfiguration actions, Security policies/requirements
Output	• Heterogeneous workflows scheduling (commissioning, deployment, and

	monitoring)
Sources	Application Placement & Scheduling Controller, Autonomic Resource Manager
Consumers	Autonomic Resource Manager, Cloud & Edge Resources Repository, Mobile On/Offloading Processing
Beyond SOTA	The Autonomic Data Intensive Application Manager will go beyond the state of the art by controlling and mutating remote Event Processing Network (EPN) topologies from heterogeneous deployment workflows (eg. configuring resources to be part of a new Storm topology, updating network configurations, etc.). Furthermore, its own internal Monitoring system will go beyond the state of the art by extending the existing set of metrics to edge devices (battery, network antenna, etc.).
Responsible Partners	ActiveEon
Available assets	• Open Source workflows scheduler (Java)

Below we include our responses regarding to the following reviewers' recommendation:

> ***Comment**: Page 31 "Open Source workflows scheduler (Java)", which one? Was it described in the SotA deliverable? The same kind of sentence is repeated several times along the document*

The "Open Source workflows scheduler (Java)", mentioned in the table above, refers to the "ProActive Workflows & Scheduling" platform edited by Activeeon. The "Open Source Multi-Cloud resource manager (Java)", mentioned in the table in next section, refers to the "ProActive Cloud Automation" platform edited by Activeeon. These two platforms were not covered by the SotA deliverable, but they were stated as the target platforms for implementation of the mentioned functionalities in the description of work (see Section 1.4.3.2 Beyond the state of the art, WP4 description, 2.2.3 Exploitation activities). These platforms are released in open source under AGPL (Affero General Public License). The revised version of SotA deliverable now describe the ProActive Workflows & Scheduling platform.

5.2.2 Autonomic Resource Manager

The Autonomic Resource Manager (ActiveEon) is in charge of deploying, acquiring and releasing cloud or edge resources.

To manage cloud resources, the Autonomic Resource Manager directly interacts with the private or public IaaS cloud APIs to deploy and release Virtual Machines (VMs). In order to 'acquire' a resource, a script is executed on each VM (through corresponding IaaS APIs) to start a specific agent that will directly communicate with the Autonomic Data Intensive Application Manager to receive tasks execution orders and monitoring rules.

In the case of edge resources, the Autonomic Resource Manager acquires (resp. releases) a resource by starting (resp. stopping) a specific Docker container into Edge devices. The container itself includes a similar agent than the one installed on cloud resources and provides the exact same features.

The Autonomic Resource Manager receives as input requests from the Autonomic Data Intensive Application Manager to respectively deploy, acquire and release resources. Requests are executed from a commissioning workflow scheduled by the Autonomic Data Intensive Application Manager. In case of complex cloud deployments (private networks, NAT rules, security groups, etc.), the resources deployments are executed in order and VMs are deployed in parallel to minimize the whole deployment time.

The Autonomic Resource Manager provides as output all the information collected on the new deployed resources to the Autonomic Data Intensive Application Manager. This includes cloud related information

(VM type, data disk type and size, ID, public IP address, etc.), as well as application information (e.g. Storm topology configuration), and PrEstoCloud specific information (VPN configuration, private IP address, local domain name, etc.). All this information is then further enriched by the Autonomic Data Intensive Application Manager (if necessary) and transmitted to the Cloud & Edge Resource Repository which updates its database accordingly.

Thanks to a tight cooperation with the Autonomic Data Intensive Application Manager, the Autonomic Resource Manager will go beyond the state of the art by deploying, managing and interconnecting resources (VMs and/or Containers) on private/public IaaS and edge devices seamlessly.

Autonomic Data Intensive Application Manager	
Function	• Deploying and releasing cloud and edge resources. • Acquiring resources by installing ActiveEon agent and Monitoring probes.
Subcomponents	Monitoring probes installer
Input	• (De)commissioning tasks.
Output	• Information on deployed cloud and edge resources.
Sources	Autonomic Data Intensive Application Manager
Consumers	Autonomic Data Intensive Application Manager, Cloud and Edge resources
Beyond SOTA	Thanks to a tight cooperation with the Autonomic Data Intensive Application Manager, the Autonomic Resource Manager will go beyond the state of the art by deploying, managing and interconnecting resources (VMs and/or Containers) on private/public IaaS and edge devices seamlessly.
Responsible Partners	ActiveEon
Available assets	• Open Source Multi-Cloud resource manager (Java)

5.2.3 Application Placement & Scheduling Controller

This component ensures the initial placement of the application jobs and their reconfiguration from an observation of runtime events. It will receive as inputs TOSCA documents from the resources adaptation recommender and the application fragmentation & deployment recommender. Those documents will define the application expectations in terms of resource allocation, and placement constraints (e.g. hardware or site affinity, fault-tolerance). Aside, it will also use from the Cloud & Edge resource repository the view of the current infrastructure and the current application status. The scheduler will then check if the current deployment satisfies all the application constraints. If some of the constraints are violated, it will then contact the Autonomic Data Intensive Application Manager to propose the actions to perform over the infrastructure to ensure the constraint satisfaction. These actions will be issued using TOSCA documents to express explicitly the mapping of the jobs on resources or the reconfiguration actions to execute using a TOSCA extension to define.

In public or private cloud infrastructure, the placement constraints are limited to VM to node or VM to VM (anti-) affinities expectations and resource allocation. In the context of PrEstoCloud, we will propose a new set of constraints to reflect the particularities of edge computing such as heterogeneous runtimes (virtual machines, containers) or platforms (edge layer, private or public clouds). We will also propose a function, to be minimized, to reflect the hosting cost while satisfying all the placement constraints.

Application Placement and Scheduling Controller	
Function	• Scheduler for placing jobs under constraints • Initial constraints are set for the initial deployment • Constraints are refined or added depending on the context inferred by the meta-management layer
Subcomponents	Tosca to/from BtrPlace model compilers, BtrPlace implementation of PrEstoCloud specific constraints, Constraint solver (provided)
Input	• Description of jobs to be placed (e.g. CPU/RAM requirements) • Set of constraints to be applied (e.g. colocation of tasks on hardware) • One objective function (e.g. maximizing resource provider revenue) • Description of available physical resources (edge/private cloud), and of available public clouds.
Output	• Action plan for initial deployment of the infrastructure • Action plan for migration of current infrastructure to new optimal
Sources	Resources Adaptation Recommender
Consumers	Autonomic Data Intensive Application Manager
Beyond SOTA	• Characterization and placement of tasks that are not VMs • Extension of code-base to enable modeling of public clouds • Monetary cost functions for joined edge, private clouds, and public cloud infrastructure.
Responsible Partners	CNRS
Available assets	• BtrPlace

5.2.4 Security Enforcement Mechanism

Security Enforcement Mechanism is based on a module that is responsible for ensuring protection with different granularity levels and to enforce protection in different operational layers, all the way from the networking infrastructure up to the cloud application itself.

The security reinforcement at the network layer is based on the SDN technology in order formulate rules on existing virtual firewalls and Intrusion Detection Systems(IDS). Furthermore, the Security Enforcement Mechanism is responsible to improve the network security by instructing the deployment of VNFs (such as firewalls and IDS virtual devices) through virtual machines that can be instantiated on demand, following the NFV paradigm.

The security aspects of resources provisioning through PrEstoCloud can be further enhanced with context-aware access control that can be enforced for the access or configuration of the network and edge resources or the processing outcomes of the applications. For this reason, the PrEstoCloud Semantic Model and the PaaSword Context Aware Security Model 0 (based on the XACML standard) will be used as the core of the context-aware authorization mechanism that PrEstoCloud envisions. The actual configuration and enforcement of these policies requires the creation of a lightweight intervention mechanism, probably by adapting and extending PaaSword or by using Drools (https://www.drools.org/) as a reasoning engine (regarding the contextual information).

All the functionalities of the Security Enforcement Mechanism will be available as programmable security resources that can be used by PrEstoCloud platform users, and will utilize the available cloud resources.

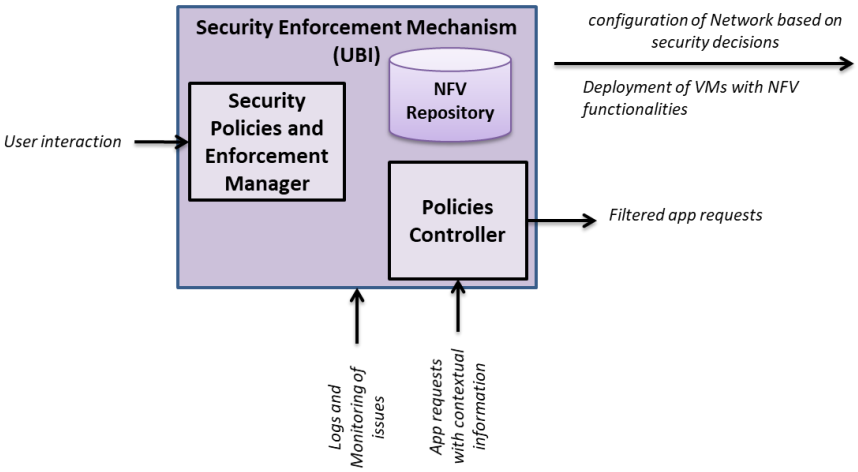


Figure 10: Security Enforcement Mechanism Interactions

The following table describes the Security Enforcement Mechanism.

Security Enforcement Mechanism	
Function	• Enforces protection in different operational layers • Leverages SDN technology and follows the Network Function Virtualization (NFV) model • Provides access and usage control mechanism
Subcomponents	NFV Repository, Security Policies and Enforcement Manager (UI)
Input	• Monitoring Logs and events
Output	• Changes on the configuration of the network sent to Autonomic Data Intensive Application Manager component • Enablement of SDN rules • Deployment of VMs with NFV functions (load balancer, firewall or IDS) capabilities in the cloud • Configuration mechanisms
Sources	DevOps interactions (policies definition and configuration), Communication Broker (monitoring data), Inter-Site Network Virtualization
Consumers	Autonomic Data Intensive Application Manager (ADIAM), Inter-Site Network Virtualization
Beyond SOTA	Adaptation of context-Aware control mechanism from PaaSword, support for SDN and NFV capabilities on a dynamic and multi-cloud environment
Responsible Partners	Ubitech
Available assets	• PaaSword (policy model and policies enforcement mechanism) • Open source VNFs like IPS/IDS (probably Snort[2]) and Firewall, • Drools

[2] https://www.snort.org/

Due to the nature of the component the layers and components of the architecture of PrEstoCloud platform have to be defined before taking final decisions of the security enforcement in dynamic resources provisioning.

5.2.5 Mobile On/Offload Processing

The mobile offload component is the cloud side of the cloud-edge interaction. It provides the interface for registration and monitoring of heterogeneous edge devices (e.g. microcontroller based, Android based, lightweight Linux based computers, etc.) by communicating with a management agent installed on every device.

It maintains a database of the current state of all of registered edge devices, their connectivity, availability, and a list of device capabilities and associated resources (e.g. collocated data sources that constrain the offloading to a specific device instance). The database (Edge Resources Database) exposes a JMX interface to the Autonomic Data Intensive Application Manager, providing an abstraction of edge resources that can be targeted by the system.

The component receives deployment and reconfiguration instructions from the Autonomic Data Intensive Application Manager, and translates them into platform dependent deployment instructions, which are issued to the Edge Resource Agent via the Message Broker. Any required task binaries or containers that are not cached on the edge are encapsulated in the management messages and sent to the edge device along with deployment instructions.

Following table describes the Mobile On/Offload Processing component using the common list characteristics of the components identified in the start of this section.

Mobile On/Offload Processing	
Function	• Abstracts the available heterogeneous edge resources for the use by Autonomic Data Intensive Application manager • Maintains a database of available edge resources • Communicates with offloading agents on edge resources to start, stop and migrate tasks to and from the edge
Subcomponents	Edge Resources Repository; Registration, Monitoring and Management component
Input	• Deployment and reconfiguration instructions; • Device registration and status updates from edge resources
Output	• Device dependent deployment instructions to the edge device agent • Edge Resources Database API interface
Sources	Autonomic Data Intensive Application Manager, On/Offloading agents via Message Broker
Consumers	Autonomic Data Intensive Application Manager, On/Offloading agents via Message Broker
Beyond SOTA	Ability to deploy the same task to different platforms, depending on the current situation
Responsible Partners	JSI
Available assets	• Java Parallel Processing Framework (JPPF) • Docker, Linux Containers (LXC)

5.2.6 Control layer: an integrated view

Figure 11 depicts the sequence of interactions between components in this layer.

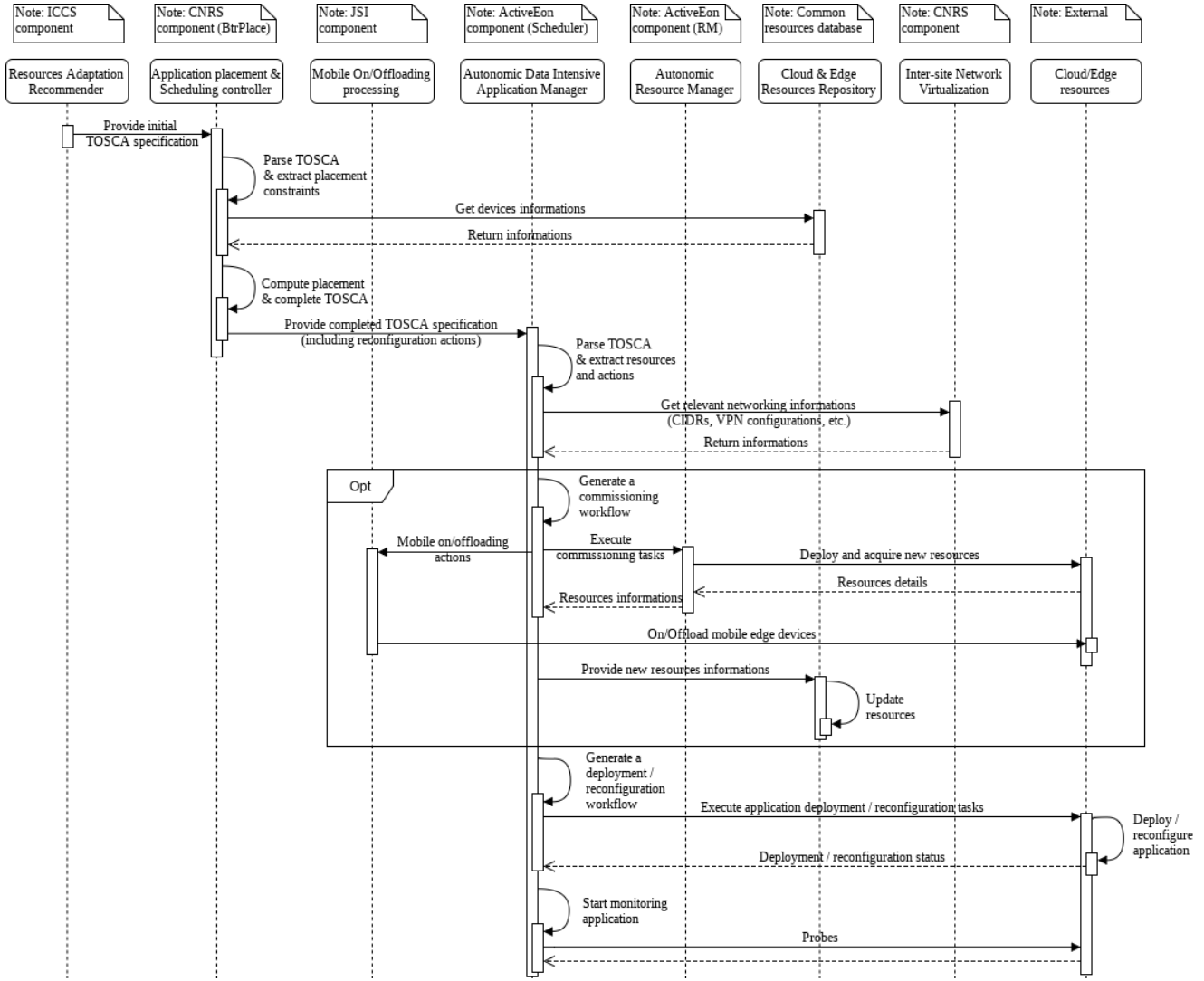


Figure 11: Control layer sequence diagram

5.3 Cloud-Edge Communication Layer

This layer represents the core communication mechanism, which realizes event-driven architecture in order to enable loosely coupling between components, esp. between data sources and data consumers.

5.3.1 Communication and Message Broker

Broker is a central part of a distributed system. This broker realizes Publish/Subscribe message queuing form. It has a sender and one or more receivers for a single message. In this form, the sender is called a publisher, because the sender will send the message by publishing a message to the topic. The receiver(s) subscribed to the topic will then receive the message, which will be present in the topic until all subscribers receive the message or until the message expires. For each type of message in the Publish/Subscribe form of MOM, a "publisher" is chosen which sends out messages, and one or more "subscribers" are chosen which "subscribe" to the messages. Once a subscriber has been registered with the middleware component, any new messages sent by the publisher are automatically delivered to that subscriber in addition to sending the same messages to any listeners, which are already registered, usually through an event or callback mechanism [SAN14].

Communication and Message Broker	
Function	• Routes messages to the specific consumer/recipient • Supports asynchronous communication • Decouples data providers and data consumers • Supports scaling of a distributed system
Subcomponents	Broker, Protocol, Topics
Input	Monitoring sensors from cloud and edge resources Application data (profile)
Output	• Monitoring sensors from cloud and edge resources • Application data (profile)
Sources	Cloud and edge resources (monitoring) Cloud and edge application
Consumers	Workload Predictor Situation Detection Mechanism Mobile Context Analyzer
Beyond SOTA	Public access to the broker in order to a very broad type of edge applications
Responsible Partners	Nissatech
Available assets	• Open source solutions (e.g. RabbitMQ)

5.4 Cloud-infrastructure and Device layers

These layers are dealing with the resources (infrastructure and the devices) in order to enable a harmonized access and management of monitoring activities.

5.4.1 Spatio-Temporal Processing Library

The Spatio-Temporal Processing Library can be used in the edge resources in order to create an ad-hoc overlay network that can be used in order to allow the nodes to communicate directly and exchange data. For the creation of this ad-hoc overlay network several state of the art technologies have to be employed.

More specifically, SDN principles will be adopted along with innovative protocols such as LISP[3] and/or DHT[4] in order to guarantee ad-hoc path identification, routability and information exchange.

Locator/ID Separation Protocol (LISP) is routing architecture that provides new semantics for IP addressing. The current IP routing and addressing architecture uses a single numbering space, the IP address, to express two pieces of information a) device identity and b) the way the device attaches to the network. The LISP routing architecture design separates the device identity, or endpoint identifier (EID), from its location, or routing locator (RLOC), into two different numbering spaces. Splitting EID and RLOC functions yields several advantages such as scalability and routing simplicity.

On the other hand, a distributed hash table (DHT) is a class of a decentralized distributed system that provides a lookup service similar to a hash table. To this extend, (key, value) pairs are stored in a DHT, and any participating node can efficiently retrieve the value associated with a given key.

This distributed system will allow to correlate devices locations and have better usage of the resources at the edge of the network. Thus, processing can be optimized by taking both space and time information of the resources and allowing the deployed services to efficiently manipulate data in poor network coverage situations.

As already mentioned, Overlay networking and DHTs are fundamental technologies for the realization of the Spatio-Temporal processing library. The ability to register the computational profile of a resource and use it without the restriction of network reachability is extremely valuable in the frame of the project. An overlay network is established based on resources provided by (all) the nodes in the ad-hoc network. Application specific information (required for the purposes of the processing) is stored in the overlay network using Distributed Hash Tables (DHT) techniques.

Ubitech already maintains a reference implementation of the LIST protocol for overlay management and a reference implementation of a DHT named Ubi:chord. Ubi:chord is a java library that can be used in order to maintain a consistent hash map (i.e. key value store) among network nodes that are connected using a mesh network topology (i.e. non centralized). Such a key/value store can be used as a medium of communication for high level applications that do not want to rely to static synchronous established channels in a mobile network. In that sense the library creates an overlay that can be used during the exchange of information that occurs during processing.

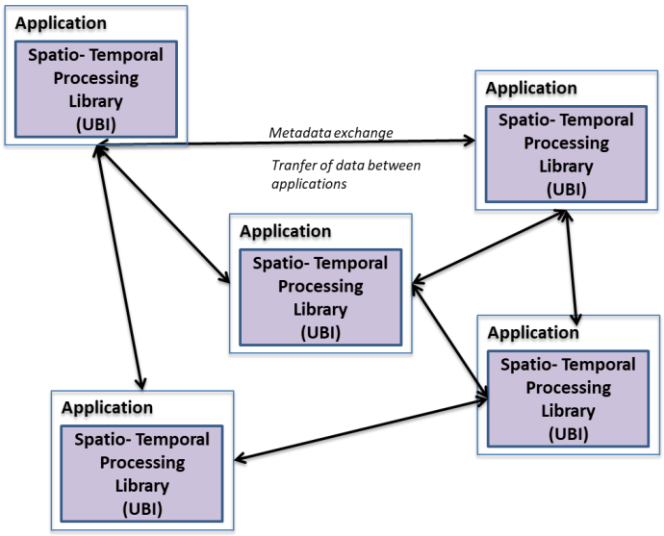


Figure 12: Spatio-Temporal Processing Library Interactions

[3] https://tools.ietf.org/html/rfc6830

[4] https://en.wikipedia.org/wiki/Distributed_hash_table

Spatio-Temporal Processing Library	
Function	A library that can be used as agent or be integrated to applications in order to create an ad-hoc overlay network that allow direct communication and data exchange on the edge nodes.
Subcomponents	N/A
Input	N/A
Output	N/A
Sources	Applications with Spatio-Temporal Processing Library
Consumers	Applications with Spatio-Temporal Processing Library
Beyond SOTA	The goal to create ad-hoc network on the edge that is self-configurable using mobile resources can provide solution in many scenarios with poor network coverage situations. Mobile Ad-hoc Networks (MANET)[5] is an area with high research interest and PrEstoCloud has the goal to make usage of such functionalities in real life scenarios (by extending ubi:chord) and specifically on edge resources.
Responsible Partners	Ubitech
Available assets	• ubi:chord[6]

5.4.2 Inter-Site Network Virtualization

The Inter-Site Network Virtualization component is responsible for creating an overlay network between the sites selected to run computation tasks, and the site where the PrEstoCloud platform is deployed, if it resides at a different location. The goal of this component is to enable full duplex communication between components (either platform components, or customer applications, etc.), through the public Internet, but within the safe compounds of a dedicated tunnel. Consequently, the applications are able to talk to each other, as if they were located on the same site. This enables multi-cloud deployment of standard application, e.g., Storm.

A common referrer for this kind of overlay network is a *Virtual Private Network* (VPN). By relying standardized VPN technologies, such as IPsec and OpenVPN, the Inter-Site Network Virtualization component is able to overcome the limitations of interoperability due to the implementation of proprietary protocols. For example, public cloud providers, such as Amazon AWS and Microsoft Azure), offer dedicated VPN solutions, so that their customers can easily use the public cloud as an extension of their own private infrastructure. But, these offerings are proprietary and not compatible among themselves (a possible vendor lock-in). Another advantage of relying on standardized technologies is the ability for the PrEstoCloud platform to orchestrate the deployment of the overlay, so as to optimize its topology (thereby avoiding network bottlenecks), and the ability to adapt the overlay network to the infrastructure requirements, as formulated by the Meta-Management Layer (e.g. addition or removal of a site).

The features and specifications of the Inter-Site Network Virtualization component are summarized in the following table. It is internally structured in 5 components that we now briefly describe.

[5] https://datatracker.ietf.org/wg/manet/about/

[6] https://www.ubitech.eu/technology/autonomicity/

The *IP Numbering* module is responsible for attributing unique IP addresses within the overlay network. For virtual machines, this corresponds to the private IP address that will be assigned to the VM at boot-up, alongside the network-level configuration (i.e. gateway, resolver, etc.). For edge devices, this corresponds to the IP address to be used by the device with its tunnel interface.

The *Gateway Initialization* module is responsible for the initial setup of a location inside the overlay. It reserves an IP space to be used for machines in that location; generates the required cryptographic keys so as to provide mutual site authentication, and tunnel encryption; and it provides a template virtual machine to be used as the site's gateway to the outside world.

The *Gateway Routing* module provides the ability to disseminate the routing information among the site's gateways so that they know how to establish tunnels to new sites, and send data through these tunnels.

The *Tunnelling* Module is responsible for establishing the tunnels between the different sites. It can be run from within a site's gateway (in the case of a cloud), or on its own (in the case of an edge device).

The *Measurement* Module is responsible for collecting network-level information about a site, and returning them to the PrEstoCloud platform. A number of metrics can be collected passively, e.g. current transfer rate, total transfer, etc. Another set of metrics need to be actively measured, i.e. by injecting measurement traffic into the network, e.g. estimation of the available bandwidth. Of course, in pay-as-you-go services, such as public IaaSes, carrying out these experiments means paying for more data transfer. For this reason, the Measurement module can be remotely called by the PrEstoCloud platform only when the need for a specific and precise measure arises.

Inter-Site Network Virtualization	
Function	• Creates an overlay network between the sites where tasks are deployed • Optimized routing between sites • Site-to-site security
Subcomponents	IP Numbering Module Gateway Initialization Module Gateway Routing Module Tunnelling Module Measurement Module
Input	1. Site/location where tasks are going to be deployed 2. RPCs for proceeding to network measurements
Output	1. IP address and network configuration to be applied to VMs 2. Network-level information about site
Sources	Autonomic Data Intensive Application Manager Communication and Message Broker
Consumers	Autonomic Data Intensive Application Manager Resources Adaptation Recommender Communication and Message Broker
Beyond SOTA	• Provider-independent technology • Dynamic, scalable, and on-demand deployment
Responsible Partners	CNRS
Available assets	• OpenVPN,

	• StrongSwan

5.4.3 On/Offloading Agents

Mobile offloading agents are the edge component of the cloud-edge interface. They are installed on every edge resource, and respond to the Mobile On/Offloading Processing component's requests for task offloading, and periodically send the resource status updates to the Edge Resources Database.

Both a lightweight Linux agent (for deployment on Raspberry Pi and similar devices) and an Android agent (for deployment on Android based user interface devices in trucks) will be developed, with the architecture sufficiently generic to allow for other kinds of devices (e.g. microcontrollers) to be supported by the same architecture. Depending on the evolution of use-cases, either task-based or container based approach will ultimately be used on pure Linux platforms, and a JPPF task-based approach on Android platforms.

The agent is responsible for downloading of task binaries or containers from the Mobile On/Offload Processing component, executing them with appropriate permissions, and cleaning up after the execution.

Mobile On/Offload Agent	
Function	• Registers the edge device with the On/Offloading Processing component's Edge Resource Database • Responds to management requests for task migration, starting and stopping
Subcomponents	-
Input	• Edge device sensors, • Task install/start/stop instructions from the Mobile On/Offload Management component
Output	• Device and task status
Sources	Mobile On/Offload Processing component via Communication Broker
Consumers	Mobile On/Offload Processing component via Communication Broker
Beyond SOTA	Extension of current technology, possible extension to microcontroller based systems if needed
Responsible Partners	JSI
Available assets	• Java Parallel Processing Framework (JPPF), • Docker, Linux Containers (LXC)

5.4.3 Monitoring Probes

Monitoring Probes are present on every resource managed by the architecture, as well cloud than edge resources. These monitoring probes are responsible of communicating information about the resources. In the case of cloud or edge devices with virtualization capability, they are installed by the Autonomic Resource Manager when deploying virtual machines. In case of edge resources such as mobile phones, they are part of the corresponding mobile application, and if the operating system supports it (eg. Odroid ARM board) they are containerized using Docker. They publish into the broker both computing health information - such as CPU load, memory usage, network latency, battery load, etc. - , status of the

application agent - idle, busy, paused, error - and flexible big data application metrics - such as progress, finish, etc. These informations are sent in form of event messages to be consumed by both control and meta-management layers.

Monitoring Probes	
Function	Informs about the status of the managed resources and applications
Subcomponents	none
Input	• Inner resource information (resource health, agent status, application metrics)
Output	• Events about changes in resources status
Sources	none
Consumers	Communication Message Broker
Beyond SOTA	A complete monitoring probes to measure whole information, both low level (hardware) and high level (applicative)
Responsible Partners	ActiveEON
Available assets	Existing monitoring component of the proactive platform

5.4.5 Cloud-Edge Communication Layer: an integrated view

Figure 13 presented below depicts a sample sequence of interactions between the Communication Broker and other components of the platform.

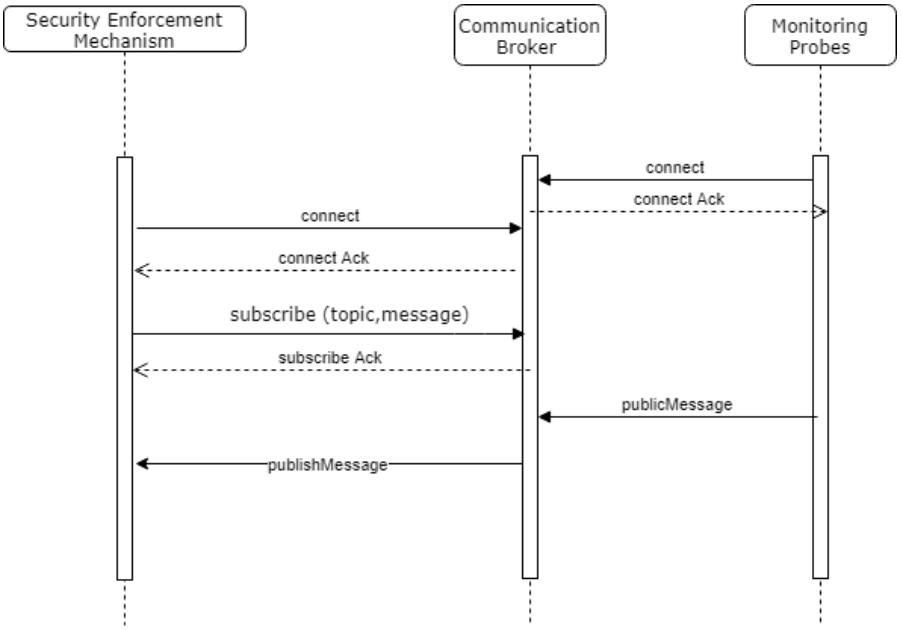


Figure 13: Cloud-Edge Communication Layer Sample interaction

6. Interfaces Documentation

This section describes the main interfaces within the PrEstoCloud architecture, as these identified with the definition and the integration needs of the components specified in section 5. This section also gathers information about the interfaces that will help all technical partners on the implementation of the integrated solution of PrEstoCloud by defining the communication between the components that will be created in the scope of Work packages 3-4-5.

6.1 Interfaces of the Meta-management Layer Components

In this section, we focus on the main interfaces provided and requested by the individual components that formulate the PrEstoCloud Meta-management Layer. Specifically, Figure 14 presents the main components involved and depicts either internal or external interfaces (to Communication & Message Broker and Control Layer). We note that placement requirements and application annotations will be injected to the Meta-management Layer through an appropriate user interface that will be detailed as part of the WP5 work.

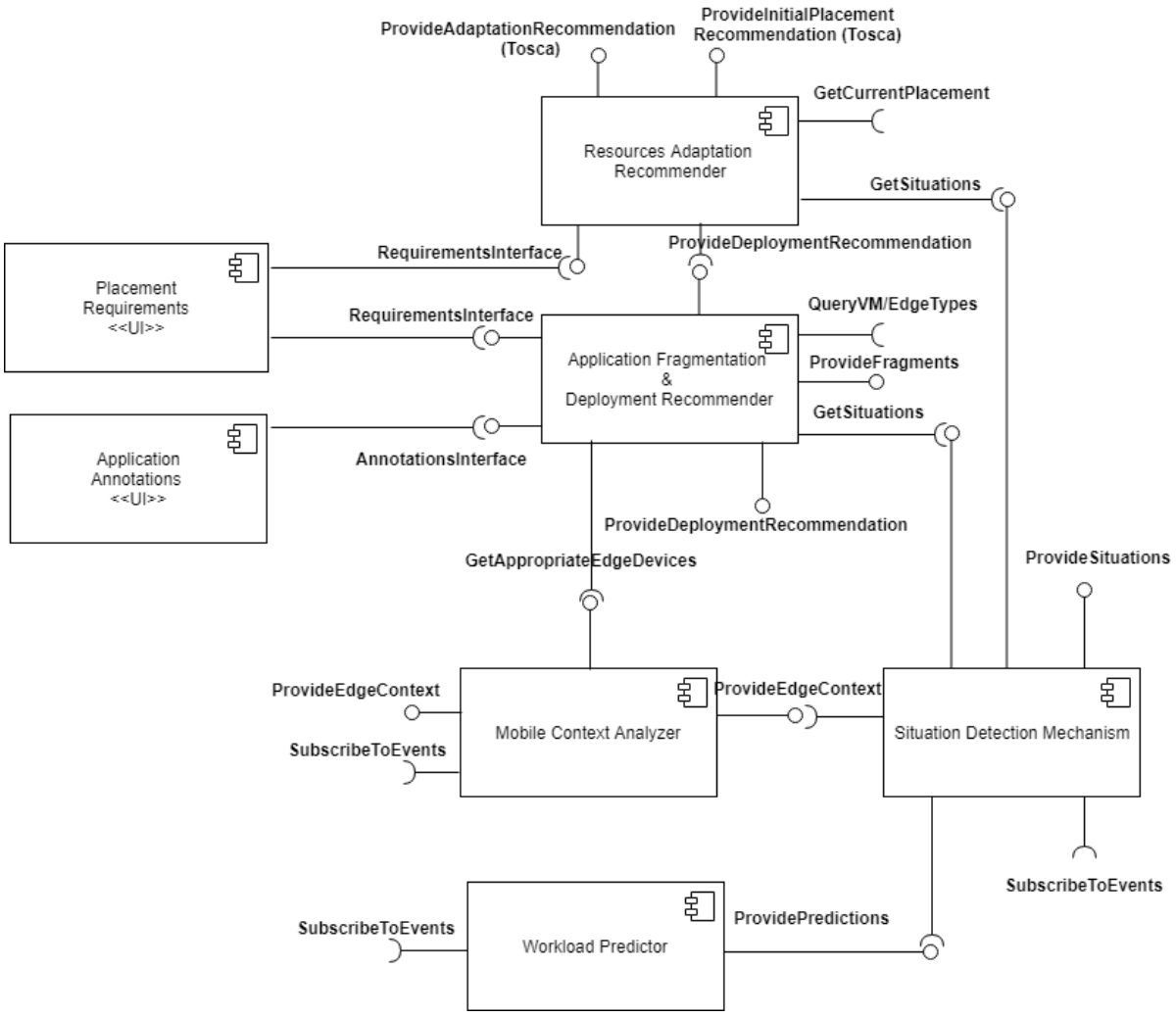


Figure 14 : Meta-management Layer Component Diagram

The following tables refer to the analysis of Figure 14, since they carry information about the communication envisioned between the PrEstoCloud components, the connection type and message format that may be used along with the expected constraints that may characterize this communication. Specifically, Table 1 refers to the main interfaces of the Mobile Context Analyzer, Table 2 describes the interfaces of the Situation Detection mechanism, Table 3 the interfaces of the Application Fragmentation &

Deployment Recommender (DIAFDRecom) and Table 4 the interfaces of the Application Fragmentation & Resources Adaptation Recommender (RARecom).

Table 1: Interfaces/Connection Types – Mobile Context Analyzer

Component1	Component2	Connection type	Message format	Constraints (throughput/size)	Description
Mobile Context Analyzer	Communication & Message Broker	Pub/sub (e.g. MQTT)	e.g.1 JSON Topic *resource.metric* Payload *timestamp* *metric value* *e.g.2* JSON Topic *request.context* Payload *timestamp* *aspect_1* *aspect_2* *aspect_n*	- Per s/min/h - or upon request - KBs/event	Receive regularly pushed events & events on request from edge devices for certain parameters e.g. battery, location, network capacity etc.
Mobile Context Analyzer	Situation Detection Mechanism	Replies (e.g. Restful)	e.g. JSON Post *edgedevice.id* Payload Edge.contextProperty1 Edge.contextProperty2	- Upon request - KBs/request	Provide context of edge devices
Mobile Context Analyzer	Autonomic Resource Manager	Replies (e.g. Restful)	e.g. JSON Topic *edgedevice.id* Payload Edge.contextProperty1 Edge.contextProperty2	- Upon request - KBs/request	Relay contextual information for appropriate edge devices to handle a certain processing job
Mobile Context Analyzer	DIAFDRecom	Replies (e.g. Restful)	e.g. JSON Topic *edgedevice.id* Payload Job.complexity1 Job.complexity2	- Upon request - KBs/request	Relay information on appropriate edge devices for undertaking a certain processing job

Table 2: Interfaces/Connection Types - Situation Detection Mechanism

Component1	Component2	Connection type	Message format	Constraints (throughput/ size)	Description
Situation Detection Mechanism	Communication & Message Broker	Pub/sub (e.g. MQTT)	e.g. JSON Topic *resource.metric* Payload *timestamp* *metric value*	- Per ms or s - KBs/event	Receive monitoring events from cloud & edge resources
Situation Detection Mechanism	Mobile Context Analyzer	Requests (e.g. Restful)	e.g.1 JSON Get *edgedevice.context.All*	Upon request - KBs/request	Retrieve context of edge devices
Situation Detection Mechanism	Workload predictor	Subscribes/Requests (e.g. MQTT, Restful)	e.g. JSON Topic *Stream.id/name* Payload *timestamp* *predicted volume.value* *predicted volume.confidence* *predicted velocity.value* *predicted velocity.confidence*	- Every time there is a new prediction event - KBs/event	Receive workload predictions (e.g. predicted throughput, volume)
Situation Detection Mechanism	Resources Adaptation Recommender/Communication & Message Broker/ Application Fragmentation & Deployment	Publishes (e.g. MQTT)	e.g. JSON Topic *situation.id/name* Payload *timestamp* *situation.parameter1* *situation.parameter2*	- Per ms or s - KBs/event	Provide detected situations under a certain context conditions

	Recommender/Autonomic Resource Manager				

Table 3: Interfaces/Connection Types - Application Fragmentation & Deployment Recommender

Component1	Component2	Connection type	Message format	Constraints (throughput/size)	Description
DIAFDRecom	Cloud and Edge Resources Repository	Request (e.g. Restful)	e.g. JSON Get VM_flavours Get Edge_types	- Upon request -Once per application initial placement	Request and receive available VM flavours and edge devices types for solutions filtering
DIAFDRecom	UI requirements/ Annotations	Request (e.g. Restful)	e.g. JSON Get *Annotations* Get *App.preferences*	-Upon request -Once per application initial placement - KBs/request	Receive qualitative/quantitative preferences from DevOps and application developers along with annotations that can define fragments and their requirements
DIAFDRecom	Mobile Context Analyzer	Requests (e.g. Restful)	e.g. JSON Get *edge.jobcomplexity.All*	- Upon request - KBs/request	Receive information on appropriate edge devices per certain processing job complexity (e.g. CPU intensiveness)
DIAFDRecom	Situation Detection Mechanism	Requests (e.g. MQTT)	e.g. JSON Topic *situations.id/name*	- Per ms or s - KBs/event	Receive detected situations under a certain context conditions
DIAFDRecom	RARecom/Control layer components	Provide (e.g. Restful)	e.g. JSON Post *application_fragment.id* Payload *timestamp* *quantitive_constraints* *qualitative_contraints*	- Upon request - KBs/event	Provides meaningful fragments per application along with their requirements (e.g. certain fragment cannot be deployed on edge devices)

Table 4: Interfaces/Connection Types - Application Fragmentation & Resources Adaptation Recommender

Component1	Component2	Connection type	Message format	Constraints (throughput/size)	Description
RARecom	UI requirements	Request (e.g. Restful)	e.g. JSON Get *App.preferences*	Upon request - KBs/request	Receive qualitative/quantitative preferences from DevOps and application developers
RARecom	Cloud & Edge Resources Repository	Request (e.g. Restful)	e.g. JSON Get *Topology* Get *ApplicationPlacement*	Upon request/ situation that triggers adaptation - KBs/request	Request current processing topology and placement
RARecom	Application Placement and Scheduling Controller	Provides (e.g. Restful)	Extended TOSCA specification document Post *new_deployment*_file	Immediately following a situation that triggers adaptation - KBs/event	Send reconfigured topology and application placement

RARecom	Situation Detection Mechanism	Pub/Sub (e.g. MQTT)	e.g. JSON Topic *Situation.id/name* Payload *timestamp* *situation.parameter1* *situation.parameter2*	- Per ms or s - KBs/event	Receive detected situation & context of the used and/or available edge devices
RARecom	DIAFDRecom	Request (e.g. Restful)	e.g. JSON Get *application_fragment.All*	Upon request - KBs/request	Requests fragments per application along with their requirements (e.g. certain fragment cannot be deployed on edge devices)

6.2 Interfaces of the Control Layer Components

In this section, we focus on the main interfaces provided and requested by the components that formulate the PrEstoCloud Control Layer. Specifically, Figure 15 shows the main components involved in the Control Layer and depicts internal and external interfaces and the interaction between components in the Control layer.

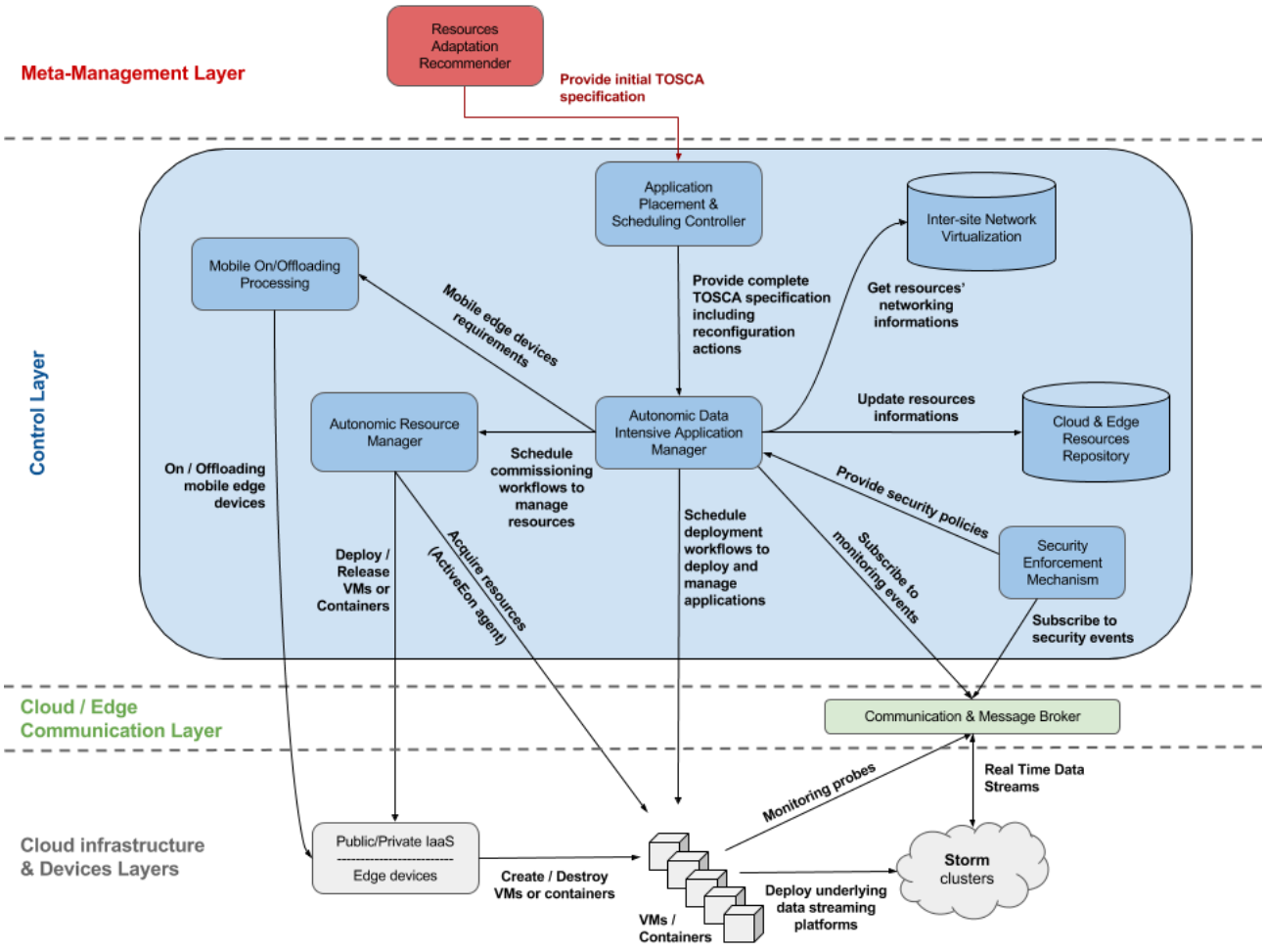


Figure 15: Control layer - interaction between components

Figure 16 depicts the UML component diagram.

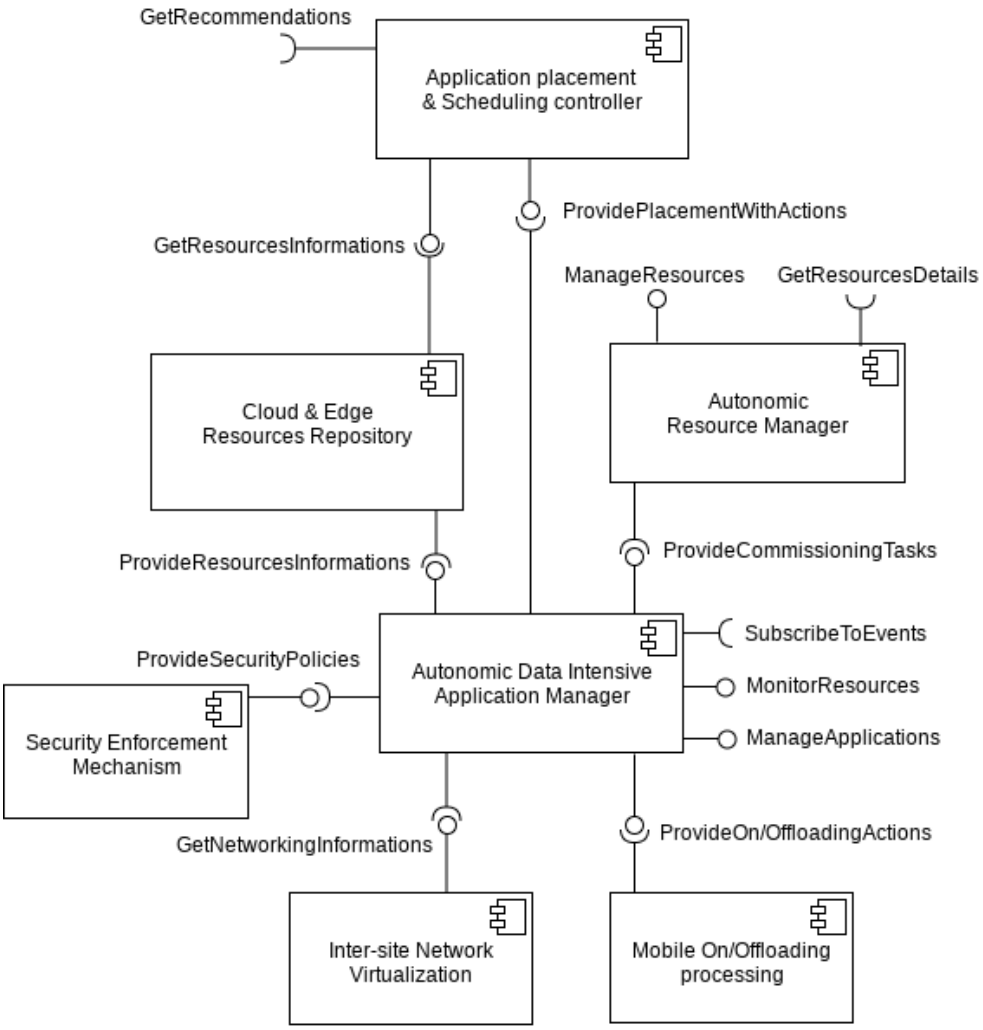


Figure 16: Control layer – component diagram

Similarly to the Meta-Management Layer, the following tables refer to the analysis of the communications illustrated in Figure 15 and Figure 16. Specifically, Table 5 refers to the main interfaces of the Autonomic Data Intensive Application Manager and Table 6 describes the interfaces of the Autonomic Resource Manager. Table 7 correspond to Application Placement & Scheduling Controller. Table 8 describes the interfaces used by the Security Enforcement Mechanism.

Table 5: Interfaces/Connection Types – Autonomic Data Intensive Application Manager

Component1	Component 2	Connection type	Message format	Constraints (throughput/size)	Description
Autonomic Data Intensive Application Manager	Autonomic Resource Manager	REST API (JSON). Provides commissioning and deployments workflows through REST API.	JSON. Eg. { tag", "image", "number", "hardware":{"type", "cpu", "mem"}, "credentials":{ "username", "password" }}	One request for each resource type to deploy (a single request can deploy multiple resources of the same type (eg. cloud, region, and flavour)).	Executing commissioning script to deploy, acquire, and release resources.
Autonomic Data Intensive Application Manager	Inter-site Network Virtualizatio n	REST API (JSON). Retrieves networking information about resources involved in an initial deployment or a reconfiguration. scenario	- JSON/YAML for resource information - RAW data for network configuration files	One request per resource to configure, and one request for the configurations of the whole application network.	Retrieving the desired network configuration for the new/redesigned application to configure.

Autonomic Data Intensive Application Manager	Cloud & Edge Resources Repository	Send resources description files (JSON/XML). Publish up to date information about one or more resources.	JSON: a complete description of the new resource. (unique identifier. IP address, CPU, memory, disk, location, etc.)	Every time a new resource is deployed or released .	Updating the shared cloud/edge resources catalog to reflect last deployed/released resources.
Autonomic Data Intensive Application Manager	Monitoring	REST API (JSON). Provides complete monitoring rules to probe resources and react on events	JSON. Eg: Each request /rule to submit must contains: - the metric to probe and the probing interval - the condition(s) to be fulfilled in order to trigger an action. - the action to trigger (Java based code. Eg. trigger a workflow with specific parameters)..	One request per rule to add on the CEP engine.	Defining and enacting specific monitoring rules when acquiring new resources (Cloud/Edge).
Autonomic Data Intensive Application Manager	Cloud/Edge resources	Periodical probes to Java Management Extensions (JMX) endpoints (hosted on resources) through: - ActiveEon's own communication protocol. - Java Remote Method Invocation (RMI)	e.g. JSON Get VM_flavours	Periodical requests per resource to monitor. The frequency depends on the configured monitoring rule.	Monitoring a resource (cloud or edge) by querying a remote JMX endpoint. The endpoint is automatically provided by the Autonomic Resource Manager when acquiring a new resource. The remote endpoint is made accessible to the Monitoring component through ProActive's own communication protocol.
Autonomic Data Intensive Application Manager	Mobile On/Offloading processing	REST API. Sends mobile on/off loading actions	eg. JSON description of the mobile resource (edge), and the action to perform.	Every time a new mobile resource (edge)is acquired or released .	Enacting mobile device on/offloading actions to start/stop processing data on the edge.

Table 6: Interfaces/Connection Types - Autonomic Resource Manager

Component1	Component2	Connection type	Message format	Constraints (throughput/size)	Description
Autonomic Resource Manager	Cloud/Edge resources	Public/private IaaS cloud APIs. SSH to edge devices.	e.g. JSON CreateInstance() DestroyInstance() AssignPublicIP() AssignSecGroup()	Every time there is a new resource to deploy or an existing resource to release. Docker containers must be used to deploy ActiveEon agents on edge	Deploy and acquire Cloud/Edge resources.

			...	devices.	

Table 7: Interfaces/Connection Types - Application Placement & Scheduling Controller

Component1	Component2	Connection type	Message format	Constraints (throughput /size)	Description
Cloud and Edge Resources Repository	Application Placement & Scheduling Controller	Provides up-to-date list of available resources	JSON	None within scope of project	BtrPlace needs to have a description of available resources on physical machines (e.g. uCPU/uRAM) to be able to place jobs on them.
Resource Adaptation Recommender	Application Placement & Scheduling Controller	Provide application requirements	TOSCA file	Asap. For the initial deployement requests. Every 15 minutes for reconfiguration requests.	Completes the TOSCA++ file provided by the Resource Adaptation Recommender, pins all jobs to a specific device.
Application Placement & Scheduling Controller	Autonomic Data Intensive Application Manager	Provides the jobs to resource mapping and the reconfiguration actions.	TOSCA file	None within scope of project	

Table 8: Interfaces/Connection Types – Security Enforcement Mechanism

Component1	Component 2	Connection type	Message format	Constraints (throughput/size)	Description
Security Enforcement Mechanism	Communication Broker	Any Pub/sub topic (e.g. MQTT)	e.g. JSON Topic *resource.metric* Payload *timestamp* *metric value*	- Per ms or s - Kbs/event	Monitoring data from cloud & edge resources
Security Enforcement Mechanism	Autonomic Data Intensive Application Manager	Requests (e.g. Restful)	e.g. JSON Payload *timestamp* *deploy.vnf.param1* *config.vnf.param1*	- Limited frequency; during deployment or management of a security related cloud resource - Kbs/ request	Deploys and configures VNF resources
Security Enforcement Mechanism	Inter-site Network Virtualisation Orchestrator	Requests (e.g. Restful)	e.g. JSON Payload *timestamp* *config.sdn.param1* *config.sdn.param2*	- Limited frequency; during the configuration of a security related cloud resource - Kbs/ request	Configures capabilities over the existing SDN network

Table 9 lists the interfaces that are exposed by the Mobile On/Offload Processing component. On one hand, the component talks to the Autonomic Data Intensive Application Manager and provides it with the information on which edge devices are currently available for task offloading, abstracted in such a way that the rest of the system does not need to treat them in a special way. In the other direction, it receives instructions for task on/offloading and translates them to a platform dependent instruction set.

Table 9: Interfaces/Connection types – Mobile On/Offload Processing

Component1	Component2	Connection type	Message format	Constraints (throughput /size)	Description
Autonomic Data Intensive Application Manager	Mobile On/Offloading processing	REST API Receives mobile on/off loading actions	JSON description of the edge resource, and the action to perform.	Every time a new mobile resource (edge)is acquired or released	Receiving actions to be performed on the edge devices (starting/stopping of on/offloading)
Mobile On/Offloading processing	Autonomic Data Intensive Application Manager	Java JMX API	API	Periodical requests/responses, depends on the configuration	Provides the current information about edge resource availability and state to the management layer
Mobile On/Offloading processing	Mobile On/Offloading Agent	Message broker	JSON description of the actions required, with embedded task binaries	Every time a resource is acquired or released	Implements the communication between mobile offload management and agent

6.3 Interfaces to the Cloud-Edge Communication Layer

Inter-Site Network Virtualization

This Section presents the interfaces of the Inter-Site Network Virtualization component available to the other components. They can be divided into two groups: (i) request/response function calls with the Autonomic Data Intensive Application Manager, which shall occur on initial deployment of the infrastructure, or upon refactoring; and (ii) asynchronous communication of meta-data between the PrEstoCloud platform and the Inter-Site Network Virtualization gateways. Table 10 provides a summary of these interfaces. We describe them in more details below.

Table 10: Interfaces/Connection Types – Inter-Site Network Virtualization

Component1	Component2	Connection type	Message format	Constraints (throughput /size)	Description
Inter-Site Network Virtualization	Autonomic Data Intensive Application Manager	Response (to e.g. RESTful request)	e.g. JSON parameter: - site identifier	N/A	Provides the IP address and network configuration to apply to a VM before its instantiation.
Inter-Site Network Virtualization	Autonomic Data Intensive Application Manager	Response (to e.g. RESTful request)	e.g. JSON containing ANSIBLE commands	N/A	Provides a template VM to be instantiated as the network gateway for the site
Inter-Site Network Virtualization	Communication & Message Broker	Pub/Sub (e.g. MQTT)	e.g. JSON topic network.id	Max. a couple of times per minutes, per site.	Publishes a set of passive measurements related to the network-level information of the site (e.g. current transfer rates, traffic matrices, etc).
Communication & Message Broker	Inter-Site Network Virtualization	Pub/Sub (asynchronous RPC call through the broker)	e.g. JSON topic network.active.id parameters identifiers for site and experiment	Bandwidth-hungry and time consuming. Recommended for use when precise information needed.	Asks to a specific site to carry out a specific active measurement experiment on the network.

Inter-Site Network Virtualization	Communication & Message Broker	Pub/Sub (asynchronous result of a RPC call through the broker)	e.g. JSON topic network.active.id	Bandwidth-hungry and time consuming. Recommended for use when precise information needed.	This completes the asynchronous transaction with the RPC call given through the Broker.

The interfaces related to deployment and redeployment are request and responses made to the Inter-Site Network Virtualization by the Autonomic Data Intensive Application Manager. Their goal is two-fold. First, to provide a standard template for a gateway VM to be deployed in each site. This VM will be the entry and exit point of all traffic leaving the local zone. All of these gateways will be configured to connect to each other, in order to establish secure tunnels that allow the end applications to exchange data among each other. Second, to provide the individual site-specific network configuration to be applied to VMs running end-applications, in order to provide them with the right routes, resolvers, and also to avoid any possible duplicate IP address within the overlay.

The interfaces related to the asynchronous communication of meta-data aim at providing a network-level view of each site to the Meta-Management Layer, and to the PrEstoCloud platform users (e.g. DevOps). The collected data will be of two types: passive measurements, and active measurements. Passive measurements are gathered through the monitoring of network activity through the gateways. Metrics, such as the current traffic activity, cumulative traffic, average transfer rates, etc. will be periodically published on an ad-hoc topic at the Broker. On the other hand, some metrics, e.g. the tunnel capacity, in other words, the available bandwidth for applications to exchange data within the tunnel, can only be reliably measured by injecting measurement traffic. In situations where data transfers are costly, it is undesirable to proceed to these measurements periodically. For this reason, measurements can be asynchronously requested from the Inter-Site Network Virtualization component. Upon the receipt of a measurement request, the specified site will proceed to the specified measurement on the specified link. Upon completion of the experiment (which, depending on the situation, can take several minutes), the measured value will be published on a dedicated feed of the Broker.

On/Offloading Agents

Table 11 describes the interface between the Mobile On/Offload Processing Management component and the Agent running on every device. The abstract interface is common to all the device types, but the specific instructions and payloads sent to the device depend on the target platform.

Table 11: Interfaces/Connection types – Mobile On/Offloading Agents

Component1	Component2	Connection type	Message format	Constraints (throughput /size)	Description
Mobile On/Offload Processing	Mobile On/Offloading Agent	Message broker Pub/Sub	JSON description of the edge resource state	Periodic and every time a device registers to the system, and when connectivity is regained	Notifies the Edge Resources database through the Autonomic Data Intensive Application Manager about the availability and state of edge devices

7. Conceptual Architecture Validation

This section validates the proposed conceptual architecture against the requirements defined, collected and categorized in D2.2. To achieve this validation, each component implementer identified the relevant requirements that have to be addressed and also each requirement was mapped to the platform's components in order to ensure that the functionality described can be realized by the platform. Multiple iterations of architecture have been shaped and this process helped on concluding on the final version of the conceptual architecture.

The functional and non-functional requirements have been defined accordingly, after an iterative formalization process that included also the identification of "responsibilities" for each layer included in the initial concept envisioned during the inception of PrEstoCloud. Therefore, both the requirements and elicited in D2.2 and the initial concept depicted in Figure 1 have been used to support the architecture design process, as it is described in the following subsections.

7.1 Functional Requirements Mapping to Components

Functional requirements describe the list of functionalities required from the system by the stakeholders, thus describing *what* the system should be able to do. Conceptual architecture and the defined components describe *how* to implement this functionality, and therefore by creating a mapping between the functional requirements and the components will reveal the functional completeness of the designed platform and will lead to an implemented platform with high degree of functional suitability[7].

The requirements defined in D2.2 include requirements regarding the general capabilities of the platform but also more specific requirements regarding configuration, monitoring, runtime and regulatory characteristics. All functional requirements have been categorized using the notion of the priority of the requirements in order to make the development as feasible as possible. The MoSCoW[8] prioritization technique has been used, that suggests the following priorities:

- **Must have**
- **Should have**
- **Could have**
- **Won't have (this time)**

During this process only the "Must have" and "Should have", have been used for the definition and validation of the platform. In more details, "Must have" were used to give the main direction of the architecture, while both of "Must have" and "Should have" have been used for the task of validating the conceptual architecture.

For the ease of referencing, the following table collects the important requirements defined in D2.2, listed by priority and also provides an aggregated view regarding if these requirements are mapped to components. It has to be stated that some of the "Should have" functionalities were not possible to be mapped to components at the current conceptual architecture, as it was unclear which and if the components could provide the required capabilities. This will be again checked in the actual architecture of platform the platform (that will be documented in D6.1). However as these are not "Must have" requirement the effect on the platform completeness would be minimal.

[7] http://iso25000.com/index.php/en/iso-25000-standards/iso-25010/58-functional-suitability

[8] https://en.wikipedia.org/wiki/MoSCoW_method

Table 12: Summary of most important functional requirement priorities and their coverage on the conceptual architecture

ID	Description	Priority	Mapped
FR-1	Ability to register cloud infrastructure by providing appropriate credentials and properties	Must have	Yes
FR-3	Ability to monitor resources	Must have	Yes
FR-5	Anomaly detection and alerting	Must have	Yes
FR-6	Platform offers a unified view of the sites available	Must have	Yes
FR-7	Platform offers a unified view of the resources available	Must have	Yes
FR-8	Platform reports global resource utilization	Must have	Yes
FR-9	Ability to use cloud and edge compute resources	Must have	Yes
FR-14	Ability to apply network function virtualization	Must have	Yes
FR-16	Ability to implement custom scalability policies	Must have	Yes
FR-17	Ability to monitor the queue of workflows	Must have	Yes
FR-18	Ability to manually update the amount of resources	Must have	Yes
FR-19	Platform allows to express the network configuration for all workloads	Must have	Yes
FR-20	Platform reports current and recent workload resource consumption	Must have	Yes
FR-22	Platform supports workload migration	Must have	Yes
FR-23	Platform reports network usage per machine, and per site	Must have	Yes
FR-24	Ability to define application deployment constraints	Must have	Yes
FR-25	Ability to define data placement/storing constraints	Must have	Yes
FR-26	Ability to enact (e.g. call an API) the execution of the initial deployment	Must have	Yes
FR-27	Ability to receive recommendations on initial application placement	Must have	Yes
FR-31	Ability to express runtime scalability and qualitative constraints at the level of individual Data-Intensive Applications(DIAs)	Must have	Yes
FR-33	Ability to receive recommendations on application placement reconfiguration	Must have	Yes
FR-35	Ability to enact (e.g. call an API) the implementation of the application placement reconfiguration	Must have	Yes
FR-37	Ability to use an editor for defining a situation triggering model	Must have	Yes
FR-39	Ability to detect interesting/critical situations that may lead to application reconfigurations or data	Must have	Yes
FR-40	Ability to extract high-level context for edge resources based on lower level monitoring data	Must have	Yes
FR-41	Ability to send / retrieve events to / from the Communications Broker (e.g. through an API)	Must have	Yes
FR-42	Ability to register data sources (real-time) to the system	Must have	Yes
FR-44	Enable an efficient and scalable access to past data	Must have	Yes
FR-45	Ability to provide new methods for data processing (real-time, batch)	Must have	Yes
FR-51	Ability to write/design custom selection scripts and apply it to desired tasks	Must have	Yes
FR-53	Ability to build complex workflows from a WEB interface	Must have	Yes
FR-55	Ability to write custom tasks based on common script language	Must have	Yes
FR-56	Platform understands the notion of locality	Must have	Yes
FR-57	Platform estimates inter-site network cost	Must have	Yes
FR-58	Platform estimates inter-site delay	Must have	Yes
FR-59	Platform accepts placement constraints	Must have	Yes
FR-61	Ability to guide fragmentation of DIAs using annotations	Must have	Yes
FR-62	Ability to guide the deployment of DIA fragments over cloud and edge resources using annotations	Must have	Yes
FR-63	Ability to accept recommendations about DIAs fragmentation and deployment	Must have	Yes
FR-69	Ability to containerize a Data Intensive Application and provide both the configuration layer and the scalability profile	Must have	Yes
FR-70	Ability to wrap/upload and use a data intensive application in the PrestoCloud platform	Must have	Yes
FR-2	Ability to set and attach metadata to resources based on a common description model	Should have	Yes
FR-4	Ability to centralize the monitoring in a common view or place	Should have	Yes
FR-10	Uniform resource interaction (harmonized API for different cloud providers)	Should have	Yes
FR-12	Platform enables the establishment of secure inter-site channels	Should have	Yes
FR-13	The ability to comply with the new GDPR (General Data Protection Regulation) regarding the protection of personal data and personal sensitive data.	Should have	No
FR-15	Ability to modify network rules to maintain regulatory compliance	Should have	Yes
FR-28	Ability to receive recommendations on initial data placement	Should have	Yes
FR-30	Ability to enact (e.g. call an API) the implementation of the initial data placement	Should have	Yes
FR-32	Ability to express runtime data migration constraints	Should have	Yes
FR-34	Ability to receive recommendations on data migration	Should have	Yes
FR-36	Ability to enact (e.g. call an API) the implementation of data migration	Should have	Yes
FR-48	Ability to express runtime constraints at the level of a) individual Data Intensive Applications, b) entire Data Intensive Service Graph	Should have	No
FR-52	Retrieve desired set of metadata from common PrEstoCloud model	Should have	No
FR-54	Ability to manage (export/import) custom workflows	Should have	Yes
FR-64	Ability to adjust fragmentation and deployment	Should have	Yes
FR-65	Ability to configure methods for data processing (real-time, batch)	Should have	No
FR-68	Ability to express and compose a directed acyclic graph (hereinafter data intensive service graph - DISG) that consists of data-intensive-apps (hereinafter DIA) that collaborate each other	Should have	No

The following Figure 17 visually depicts all the "Must have" and "Should have" requirements on the architecture schema in order create in an easier to understand visualization of the role of each component and layer. Also, the coloring of the requirements represents the different categories of the requirements.

With gray color boxes the general requirements are depicted, green the configuration requirements, orange the monitoring requirements, yellow the regulatory requirements and with blue the runtime requirements.

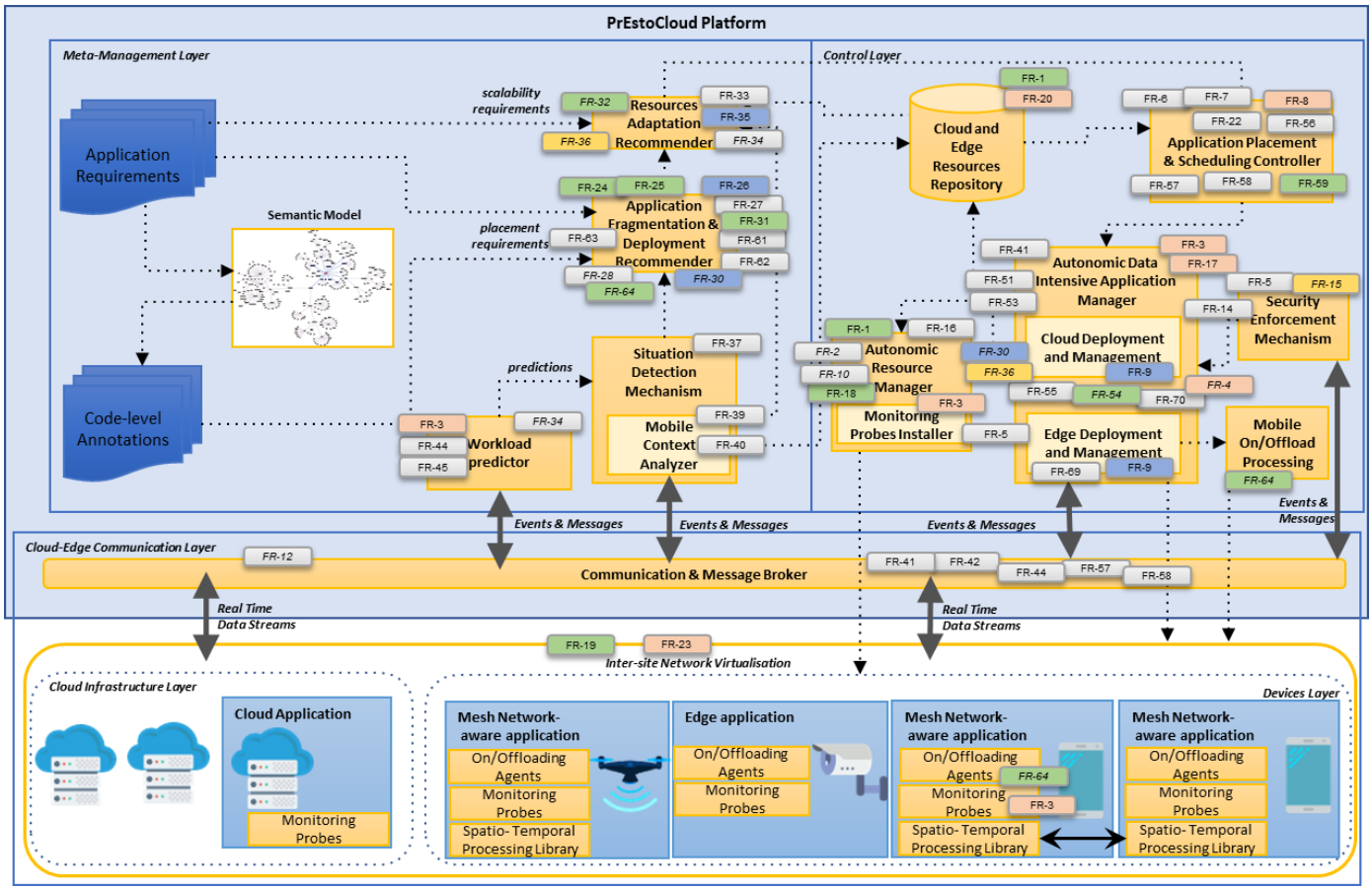


Figure 17: Illustration of requirements mapping

What is made clear from this figure is the importance of some specific components for the successful creation of the platform. More specifically, the Autonomic Data Intensive Application Manager, the Application Fragmentation & Deployment Recommender, the Autonomic Resource Manager and the Application Placement & Scheduling Controller are composing the most important part of the platform to be developed.

In the following subsections the mapping of requirement to components and layers is presented based on the defined priorities, in order to validate the design and also to ease the development of the components and the integrated platform of PrEstoCloud, as each developed component has to assure the compliancy with the constraints or the functionality that each requirement suggests.

7.2 Must Have Requirements Coverage

The following table presents the mapping of the 41 functional requirements identified as most important, on the components and the layers of the architecture.

Table 13: "Must have" functional requirements mapped to components

	Meta-Management Layer					Control Layer									Cloud-Edge Comm. Layer		Cloud Infastructure / Devices Layer		
	Resources Adaptation Recommender	Application Fragment. & Deployment Recomm.	Situation Detection Mechanism	Mobile Context Analyser	Workload Predictor	Cloud & Edge Resources Repository	Application Placement & Scheduling Controller	Autonomic Data Int. Application Manager	Cloud Deployment and Manag.	Edge Deployment and Manag.	Autonomic Resource Manager	Monitoring Probes Installer	Security Enforcement Mechanism	Mobile On/Offload Processing	Communication & Message Broker	Inter-site Network Virtualisation	Monitoring Probes	On/Offloading Agents	Spatio-Temporal Processing Library
FR-1						x					x								
FR-3					x			x				x					x		
FR-5								x			x		x						
FR-6							x												
FR-7							x												
FR-8							x												
FR-9									x	x									
FR-14								x					x						
FR-16											x								
FR-17								x											
FR-18											x								
FR-19																x			
FR-20						x													
FR-22							x												
FR-23																x			
FR-24		x																	
FR-25		x																	
FR-26		x																	
FR-27		x																	
FR-31		x																	
FR-33	x																		
FR-35	x																		
FR-37			x																
FR-39			x	x															
FR-40				x											x				
FR-41								x							x				
FR-42															x				
FR-44					x														
FR-45					x														
FR-51								x											
FR-53								x											
FR-55								x											
FR-56							x												
FR-57							x								x				
FR-58							x								x				
FR-59							x												
FR-61		x																	
FR-62		x																	
FR-63		x																	
FR-69										x									
FR-70								x											

7.3 Should Have Requirements Coverage

The following table presents the mapping of the 17 functional requirements identified as most important, on the components and the layers of the architecture.

Table 14: "Should have" functional requirements mapped to components

	Meta-Management Layer					Control Layer									Cloud-Edge Comm. Layer		Cloud Infastructure / Devices Layer		
	Resources Adaptation Recommender	Application Fragment. & Deployment Recomm.	Situation Detection Mechanism	Mobile Context Analyser	Workload Predictor	Cloud & Edge Resources Repository	Application Placement & Scheduling Controller	Autonomic Data Int. Application Manager	Cloud Deployment and Manag.	Edge Deployment and Manag.	Autonomic Resource Manager	Monitoring Probes Installer	Security Enforcement Mechanism	Mobile On/Offload Processing	Communication & Message Broker	Inter-site Network Virtualisation	Monitoring Probes	On/Offloading Agents	Spatio-Temporal Processing Library
FR-2											x								
FR-4								x											
FR-10											x								
FR-12																x			
FR-13																			
FR-15													x						
FR-28		x																	
FR-30		x					x	x											
FR-32	x																		
FR-34	x				x														
FR-36	x						x	x											
FR-48																			
FR-52																			
FR-54								x											
FR-64		x																x	
FR-65																			
FR-68																			

7.4 Mapping of Requirements to Phases

In order to help on the better understanding of the what the platform will do, the following table, depicts the mapping of the requirements covered by the architecture to the phases identified and explained in the section 4.2.

Table 15: Mapping of the requirements

ID	Description	Phase 1: Design Time	Phase 2: Initial Application Placement	Phase 3: Application Reconfiguration
FR-1	Ability to register cloud infrastructure by providing appropriate credentials and properties	x		
FR-3	Ability to monitor resources		x	x
FR-5	Anomaly detection and alerting			x
FR-6	Platform offers a unified view of the sites available	x		
FR-7	Platform offers a unified view of the resources available		x	
FR-8	Platform reports global resource utilization			x
FR-9	Ability to use cloud and edge compute resources		x	
FR-14	Ability to apply network function virtualization		x	
FR-16	Ability to implement custom scalability policies	x		
FR-17	Ability to monitor the queue of workflows			x
FR-18	Ability to manually update the amount of resources		x	
FR-19	Platform allows to express the network configuration for all workloads		x	
FR-20	Platform reports current and recent workload resource consumption			x
FR-22	Platform supports workload migration			x
FR-23	Platform reports network usage per machine, and per site			x
FR-24	Ability to define application deployment constraints		x	
FR-25	Ability to define data placement/storing constraints		x	
FR-26	Ability to enact (e.g. call an API) the execution of the initial deployment		x	
FR-27	Ability to receive recommendations on initial application placement		x	
FR-31	Ability to express runtime scalability and qualitative constraints at the level of individual Data-Intensive Applications(DIAs)		x	
FR-33	Ability to receive recommendations on application placement reconfiguration			x
FR-35	Ability to enact (e.g. call an API) the implementation of the application placement reconfiguration			x
FR-37	Ability to use an editor for defining a situation triggering model	x		
FR-39	Ability to detect interesting/critical situations that may lead to application reconfigurations or data			x
FR-40	Ability to extract high-level context for edge resources based on lower level monitoring data			x
FR-41	Ability to send / retrieve events to / from the Communications Broker (e.g. through an API)		x	
FR-42	Ability to register data sources (real-time) to the system		x	
FR-44	Enable an efficient and scalable access to past data			x
FR-45	Ability to provide new methods for data processing (real-time, batch)		x	x
FR-51	Ability to write/design custom selection scripts and apply it to desired tasks	x		
FR-53	Ability to build complex workflows from a WEB interface		x	
FR-55	Ability to write custom tasks based on common script language	x		
FR-56	Platform understands the notion of locality	x	x	x
FR-57	Platform estimates inter-site network cost			x
FR-58	Platform estimates inter-site delay			x
FR-59	Platform accepts placement constraints	x	x	x
FR-61	Ability to guide fragmentation of DIAs using annotations	x		
FR-62	Ability to guide the deployment of DIA fragments over cloud and edge resources using annotations	x		
FR-63	Ability to accept recommendations about DIAs fragmentation and deployment	x		
FR-69	Ability to containerize a Data Intensive Application and provide both the configuration layer and the scalability profile		x	
FR-70	Ability to wrap/upload and use a data intensive application in the PrestoCloud platform		x	
FR-2	Ability to set and attach metadata to resources based on a common description model	x		
FR-4	Ability to centralize the monitoring in a common view or place		x	
FR-10	Uniform resource interaction (harmonized API for different cloud providers)		x	x
FR-12	Platform enables the establishment of secure inter-site channels		x	
FR-15	Ability to modify network rules to maintain regulatory compliance			x
FR-28	Ability to receive recommendations on initial data placement		x	
FR-30	Ability to enact (e.g. call an API) the implementation of the initial data placement		x	
FR-32	Ability to express runtime data migration constraints			x
FR-34	Ability to receive recommendations on data migration			x
FR-36	Ability to enact (e.g. call an API) the implementation of data migration			x
FR-54	Ability to manage (export/import) custom workflows		x	
FR-64	Ability to adjust fragmentation and deployment		x	

8. Conclusions

These deliverable reports on the work done in WP2 in the context of developing a conceptual architecture that will satisfy very challenging requirements from the technical and use cases point of view. The deliverable is strongly influenced by several other deliverables:

- D2.1 that provides the analysis of the state of the art
- D2.2 that provided the list of requirements for the development of the system
- D2.4 that explains the data processing infrastructure
- D7.1 that describes use cases (and their requirements)

Since the envisioned Platform is very complex, this work also reflects the work of partners in different research and technology areas (like Cloud computing, Big Data, Task scheduling, Adaptive systems) and illustrates the connections to them, stating clearly our (unique) contributions.

The architecture intends to create a novel computing and management infrastructure. It will enable an efficient deployment and realization of the data-intensive applications, the development of a proper architecture is a correspondingly challenging task.

By using a layer-based approach, we designed an architecture which enable complex processing within a particular layer and an efficient communication between layers in order to realize complex processing pipelines

In order to reflect the distributed nature of the system and the need to make the loose coupling between components, the data communication architecture postulates on the principles of event-driven architecture (EDA), having the communication and message broker as the central interaction hub. Moreover, EDA enables a proper scaling of the platform, by supporting an easy extension with new types of data sources (resources) and data processing elements. This is one of the fundamental properties for supporting the work with data-intensive applications (real-time big data applications) which are the focal point of the platform.

In parallel, through a powerful task scheduling and adaptation mechanism, the architecture supports an efficient detection of adaptation opportunities in the underlying computing infrastructure and an effective implementation of them, creating the basis for an adaptive management of complex applications. The adaptation mechanism covers the entire stack of processing resources, starting from edge devices till multi cloud infrastructure, making the Platform unique comparing to the state of the art.

Since PrEstoCloud aims to enable a continuous improvement (reconfiguration) of a complex computing infrastructure, the architecture is based on the well-known self-adaptivity pattern: Monitor – Analyze – Plan – Execute - Knowledge (MAPE-K model).

We argue that the presented PrEstoCloud architecture is very innovative, enabling the realization of several advanced scenarios that are challenging for existing architectures in different domains, like

- Fog computing - PrEstoCloud architecture enables dynamic service replacement along the entire fog computing infrastructure, from Edge to the Cloud and back
- Big data - the architecture enables continuous monitoring and improvement of the QoS in real-time data analytics applications
- HPC – the architecture enables self-adaptive reconfiguration of deployed computing

During the actual development process it will be possible to make changes and adaptations to the conceptual architecture. The final results of the architecture will be documented in D6.1 Architecture of the PrEstoCloud platform.

9. References

[IBM05] IBM, An architectural blueprint for autonomic computing, Autonomic Computing White Paper, June 2005

[Paas] PaasWord Context-Aware Security Model, "https://www.paasword.eu/results/context-aware-security-model/." .

[SAN14] *S*anjay P. Ahuja, Naveen Mupparaju, Performance Evaluation and Comparison of Distributed Messaging Using Message Oriented Middleware, Computer and Information Science; Vol. 7, No. 4; 2014 ISSN 1913-8989 E-ISSN 1913-8997