DCs Optimization for Energy-Efficient and EnvironmentalLy Friendly INternet

Funding scheme: Specific Targeted Research Projects – STREP
Co-funded by the European Commission within the Seventh Framework Programme
Project no. 609140
Strategic objective: FP7-SMARTCITIES-2013 (ICT-2013.6.2)
Start date of project: October 1th, 2013 (36 months duration)

Deliverable D3.1

Data Centre energy consumption optimization platform (eCOP) Design

Due date:	01/03/2015
Submission date:	30/04/2015
Deliverable leader:	NXW
Author list:	T. Zini (NXW), A. Gronchi (NXW)

Dissemination Level

☒	PU:	Public
☐	PP:	Restricted to other programme participants (including the Commission Services)
☐	RE:	Restricted to a group specified by the consortium (including the Commission Services)
☐	CO:	Confidential, only for members of the consortium (including the Commission Services)

List of Contributors

Participant	Contributor
Siemens	S. Cosmin Nechifor, A. Nistor
Synelixis	A.Voulkidis, A. Aravanis, K. Pramataris, Ch. Tsiopoulou, Th. Zahariadis
Nextworks	T. Zini, A. Gronchi, G. Carrozzo
GRNET	G.Goumas, J.Siakavaras, N. Mouzakitis, L. Georgios, S. Aristeides
I2CAT	I.Bueno, I. Canyameres, E. Escalona
UCL	S. Clayman, A. Galis

Table of Contents

List of Contributors 2
Table of Contents 3
Figures Summary 4
Tables Summary 5
Executive Summary 6
1 DOLFIN architecture focusing on standalone DC Energy Optimization 8
1.1 DOLFIN System Objectives 8
1.1.1 New DC functionality offered by DOLFIN specific components 11
1.1.2 Revised DC functionality offered by components modified by DOLFIN 11
1.2 Energy Models for DC Software Platforms 12
1.2.1 Proposed KPIs for Energy Consumption 12
2 DOLFIN Distributed Components Interfaces and APIs Design 14
2.1.1 RESTful APIs 14
2.1.2 Publish Subscribe mechanisms 15
2.2 Applicability in DOLFIN 16
2.3 DOLFIN Control Flows 17
3 Energy Consumption Optimization Platform (eCOP) 18
3.1 eCOP functional description 18
3.1.1 eCOP architecture decomposition 20
3.2 Modules interfaces 22
3.2.1 Controls and data flows 23
3.3 Internal modules design 33
3.3.1 ICT Performance and Energy Supervisor 33
3.3.2 Energy Efficiency Policy Maker & Actuator 41
3.3.3 eCOP Monitor Database 49
3.3.4 Cooling Subsystem 52
4 Conclusions 54
5 References 56
6 Abbreviations 58
Annex 1 – eCOP APIs description 60
A 1.1 ICT Topology Graph Database structure 60
A 1.2 REST API common status code 62
A 1.3 ICT Performance and Energy Supervisor 65
A 1.4 Energy efficiency Policy Maker & Actuator 86
A 1.4.1 Optimizer - Actuator Vocabulary: 96
A 1.4.2 Optimizer – Policy Maker Vocabulary: 100
A 1.5 eCOP Monitor Database 100
A 1.5.1 Class definition 116

Figures Summary

Figure 1-1: From separate energy control loops to coordinated multiple DC energy control loops 8
Figure 1-2: DOLFIN Interworking DCs – High Level Architecture 9
Figure 1-3: DOLFIN Functional Architecture.....10
Figure 1-4: Overall DOLFIN architecture 10
Figure 2-1: REST use in DOLFIN 17
Figure 3-1: Overview of the interactions of the eCOP DOLFIN subsystem 22
Figure 3-2: Interface to handle VM deployment (topology) info.....24
Figure 3-3: Interface to handle ICT device performance data stream 25
Figure 3-4: Interface to handle consumption on non-ICT facilities.....26
Figure 3-5: Interface to eCOP DB.....27
Figure 3-6: Interface to get changes in VMs and metrics.....28
Figure 3-7: Interface to perform actions on local cloud management platform 29
Figure 3-8: Interface to Smart Grids.....30
Figure 3-9: Interface for SLA renegotiation.....31
Figure 3-10: Interface for cross-DC VM migration 32
Figure 3-11: Interface to DOLFIN Info DB.....32
Figure 3-12: ICT Performance and Energy Supervisor module 34
Figure 3-13: ICT Performance and Energy Supervisor internal structure 35
Figure 3-14: Metrics Engine.....36
Figure 3-15: DCO Broker.....38
Figure 3-16: The Energy Policy Maker & Actuator. 43
Figure 3-17: MAPE-K loop.....46
Figure 3-18: The eCOP Monitor Database module architecture.....50
Figure 3-19: Depiction of the eCOP Database initial design.....51
Figure 3-20: Liquid-cooling system diagram.....52

Tables Summary

Table 3-1: Overall modules functions....18
Table 3-2: WP3 modules interfaces....22
Table 3-3: References to requirements (ICT Performance and Energy Supervisor) 41
Table 3-4: Optimizer-Policy Actuator Vocabulary 47
Table 3-5: Optimizer-Policy Maker Vocabulary....48
Table 3-6: References to requirements (Energy Efficiency Policy Maker and Actuator) 48
Table 3-7: References to requirements (eCOP Monitor Data base) 51

Executive Summary

DOLFIN aims to significantly contribute towards improving the energy efficiency of Data Centres and the stabilization of Smart Grids, across networks of Data Centres and Smart Grids, following a holistic approach. DOLFIN architecture [1] presents a comprehensive set of functional components and interfaces that are structured (and grouped) in two major sub-systems, namely the Energy Consumption Optimisation Platform (eCOP) and the energy-conscious Synergetic DCs (SDC). These sub-systems together realise the energy closed control loop envisaged by the project. Due to the strict relationships and dependencies of interfaces, semantics and functionalities between eCOP and SDC, the DOLFIN team has decided to carry out a single joint design task of the two parts under the coordination of two different work package leaders (WP3 and WP4), in order to keep high the consistency of the design choices for the different components, simplify the subsequent development phase and minimize any risk of incompatibility or missing specifications.

Though the design work has been jointly executed by WP3 and WP4 teams together, it has been consolidated in two different specification documents, i.e. this D3.1 and D4.1, as per project plan. However, the two documents need to be considered as a unified design delivery of the DOLFIN system to be jointly read and analysed to derive a complete specification of the DOLFIN system.

This document D3.1 focuses on the detailed design specification attaining the functionalities, interfaces and relations to the requirements of the "Data Centre energy consumption optimisation platform" (eCOP). It is complemented by D4.1, which instead focuses on "Synergetic Data Centres for energy efficiency" functions, interfaces and links with requirements. Common design artefacts (consolidated architecture, overall design approach to interfaces and APIs, overall DOLFIN platform control flow, energy models for DC software platform) pertaining to both WP3 and WP4 work are presented in both deliverables D3.1 and D4.1, but with additional considerations that specifically highlight the rationale for the choice in the framework of a specific WP.

This document first presents the DOLFIN architecture (Section 1) as has been updated during the last few months during which the detailed architecture design processes were active. Then, a reference to the energy models that will be used in the deduction procedures related to determine real time DC status in eCOP, followed by a common WP3-WP4 part referring to interface design technologies is given in Chapter 2.

Section 3 details the internal design of each eCOP module, and specifically the ICT Performance and Energy Supervisor, Energy Efficiency Policy Maker and Actuator, and the water cooling system. In general, this document explores and expands the results of previous high level architecture design deliverables produced in DOLFIN (i.e. D2.2 [1]), and further refines DOLFIN platform requirements. In particular, section 3 is structured in four main design parts focusing on the core eCOP functional elements:

- **ICT Performance and Energy Supervisor module**: This component has the main responsibility to interact with underlying legacy DC subsystems and collect relevant information to evaluate metrics and KPIs. The module implements mechanisms to efficiently retrieve the data from various sub-

systems and distribute such information to appropriate consumers. Moreover, the ICT Performance and Energy Supervisor implements specific techniques for data analysis and their representation, that will be useful from other eCOP module for their optimization activities.

- **Energy Efficiency Policy Maker and Actuator**: This component implements much of the intelligence needed for the application of the optimization policies proposed in DOLFIN. It is responsible for the application of the energy optimization procedures based on particular predefined criteria and conditioned by the inputs (requests) provided by other DOLFIN components, for example requests from the Smart Grids environment, from federated DCs, etc. The Energy Policy Maker and Actuator is structured in a group of specialized entities, each of which has the responsibility of a part of the optimization process.

- **eCOP Monitor Database**: this logical module groups all the elements involved in the storage functionalities for measures, KPIs, DC assessment and status, etc. The eCOP Database not only implements pure database functions, but introduces specialized modules to efficiently interact with that data store and produce a time series aggregation.

- **Cooling Subsystem**: This is a water cooling server module. In the context of D3.1, the detailed design of the HW module is specified and details for the implementation of the prototype are provided.

The detailed specification of API interfaces exposed by the various eCOP components is provided in Annex 1 and is intended to be the base reference for DOLFIN developers in their implementation work.

Section 4 concludes the document, deriving hints for future development work.

1 DOLFIN architecture focusing on standalone DC Energy Optimization

This deliverable provides a detailed description of the design aspects of the DOLFIN Energy Consumption Optimization Platform (eCOP). Entities detailed in this document are intended to be developed from scratch, and are an integral part of the DOLFIN platform resulting from the integration of eCOP and SDC.

The next section §1.1 is a repetition of the DOLFIN deliverable D2.2: "DOLFIN requirements and system architecture". It is provided here in order to enable a smooth reading of this deliverable. Moreover, the deliverable is closely related to deliverable D4.1: "Synergetic Data Centres for energy efficiency (Design)". Cross reading may be required in some cases in order to better understand these two deliverables.

1.1 DOLFIN System Objectives

DOLFIN is designed to be integrated within the existing DC's infrastructure and for this reason the new developed components can interface with the legacy DC's infrastructure, through appropriate adapters. To fully understand the logic that was applied for DOLFIN platform design, it is important to bring attention to the fundamental aspects of the project, namely the need to build a new tool that is able to optimize and make efficient consumption of energy resources within the Data Center.

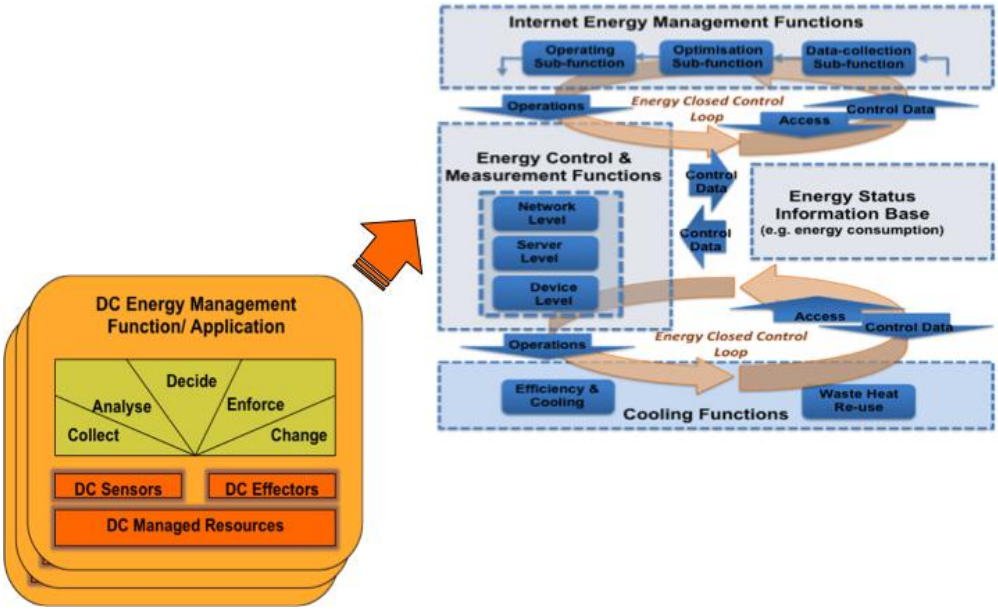

Figure 1-1: From separate energy control loops to coordinated multiple DC energy control loops

DOLFIN's primary objective is to design, develop, and validate the DC platform capable of monitoring the energy usage of the DC and react accordingly for efficient energy management. The design of the DOLFIN DC framework would explicitly accommodate energy management and it is aimed at:

- Improving capital and operational efficiencies for DC operators through the use of a common organization, automation, and operations of all energy functions across the different domains
- Migrating from an ecosystem of separate energy management functions towards a coordinated arrangement of energy management functions as represented in Figure 1-1:

The DOLFIN system is an ecosystem of collaborative DCs. DOLFIN considers a number of DCs, each one having its own DC customers and links with the energy network, as shown in Figure 1-2. From the energy network perspective, DOLFIN considers both traditional Energy Providers and Smart Grid Networks in a Smart City scenario (parts of the design activities addressed by the deliverable D4.1 "Synergetic Data Centres for energy efficiency (Design)". Moreover, DOLFIN considers that each DC may achieve further internal energy efficiency, by recycling and reusing the warm water that is used for cooling the ICT equipment for warming the DC offices (e.g. with an under floor warming system). We consider the DOLFIN System as a complete ecosystem that has a group of customers and an interaction with the energy network. These are not directly connected with a specific DC but use the DOLFIN system as a Virtual Data Centre. The main objective of the DOLFIN solution is to reduce the energy consumption of the elements within the DOLFIN ecosystem as a whole (especially the consumed brown energy) and stabilize the smart grid electricity network, wherever needed, without breaking the agreed SLAs with the end users (or renegotiating the SLAs based on signed contracts).

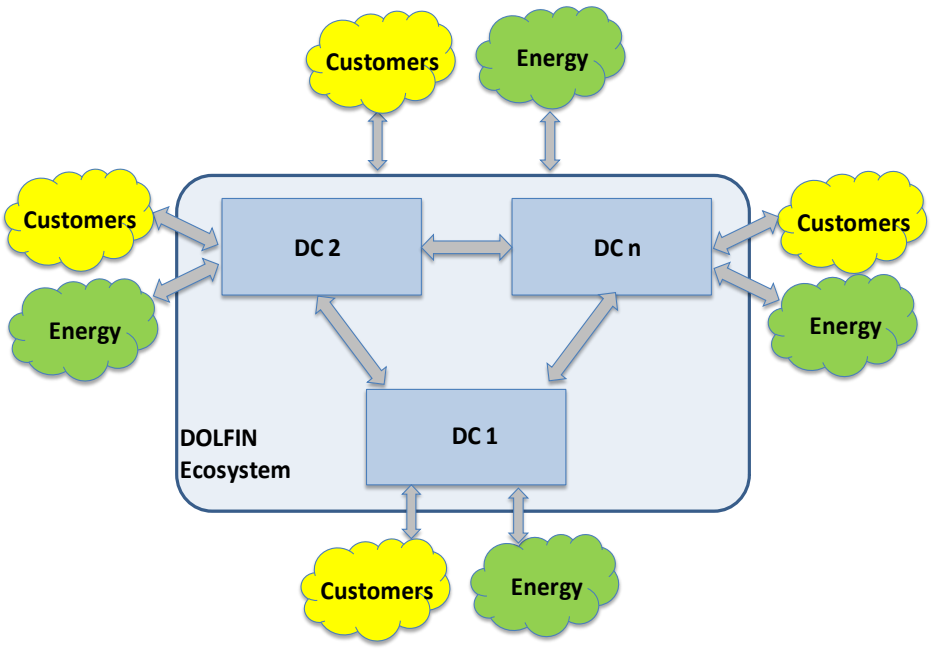

Figure 1-2: DOLFIN Interworking DCs – High Level Architecture

The DOLFIN functional architecture of each individual DC is defined in terms of new DC functional blocks, revised DC functional blocks, and the interworking interfaces needed to realise different energy closed control loops and their interactions with the normal operation of the DC. The functional architecture is depicted in the Figure 1-3.

The DOLFIN architecture is consolidated around two specialized sub-systems, namely:

- The **Energy Consumption Optimisation Platform (eCOP)**, developed in WP3, which represents the core platform in DOLFIN, where are performed the relevant actions to manage and optimize the energy consumption at Data Centre level, through the application of energy benchmarking,

dynamic control, and adaptive optimisation of the DC infrastructure. Moreover, eCOP includes a subsets of modules that allow continuous monitoring of DC resources (for both ICT and non-ICT entities) and provide specific functionalities for metrics calculation and data collection and storage.

- The **energy-conscious Synergetic DCs (SDC)**, developed in WP4, which provides a dynamic, service-effective and energy-efficient allocation of demands, across a distributed network of co-operating DCs. In addition, the SDC provides the control modules for the integration with the Smart Grid environment and that are responsible for the energy stabilisation through the interconnection with the smart grid network, providing responses on the changing demands for energy.

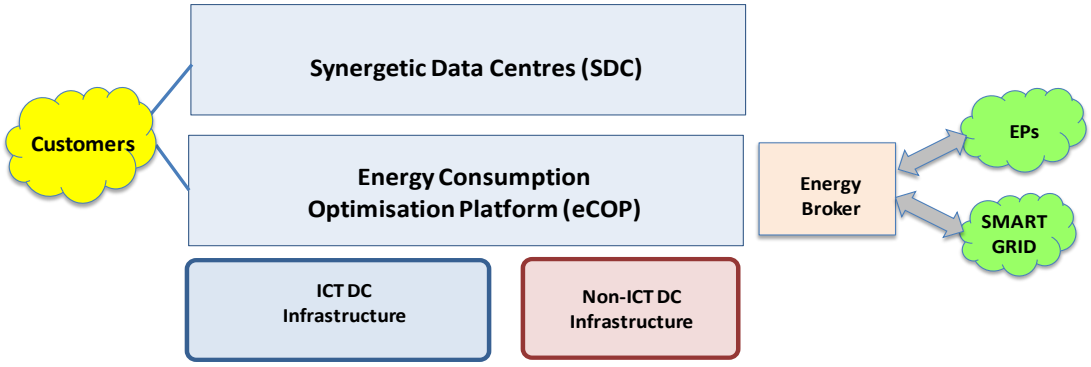

Figure 1-3: DOLFIN Functional Architecture

The overall description of DOLFIN architecture is depicted in Figure 1-4, which reports a schematic representation of components, highlights the main relations/interactions between subsystem and external entities (such as the Energy Brokers), the modules position within the DOLFIN architecture and finally how the DOLFIN are globally integrated in the normal DC environment.

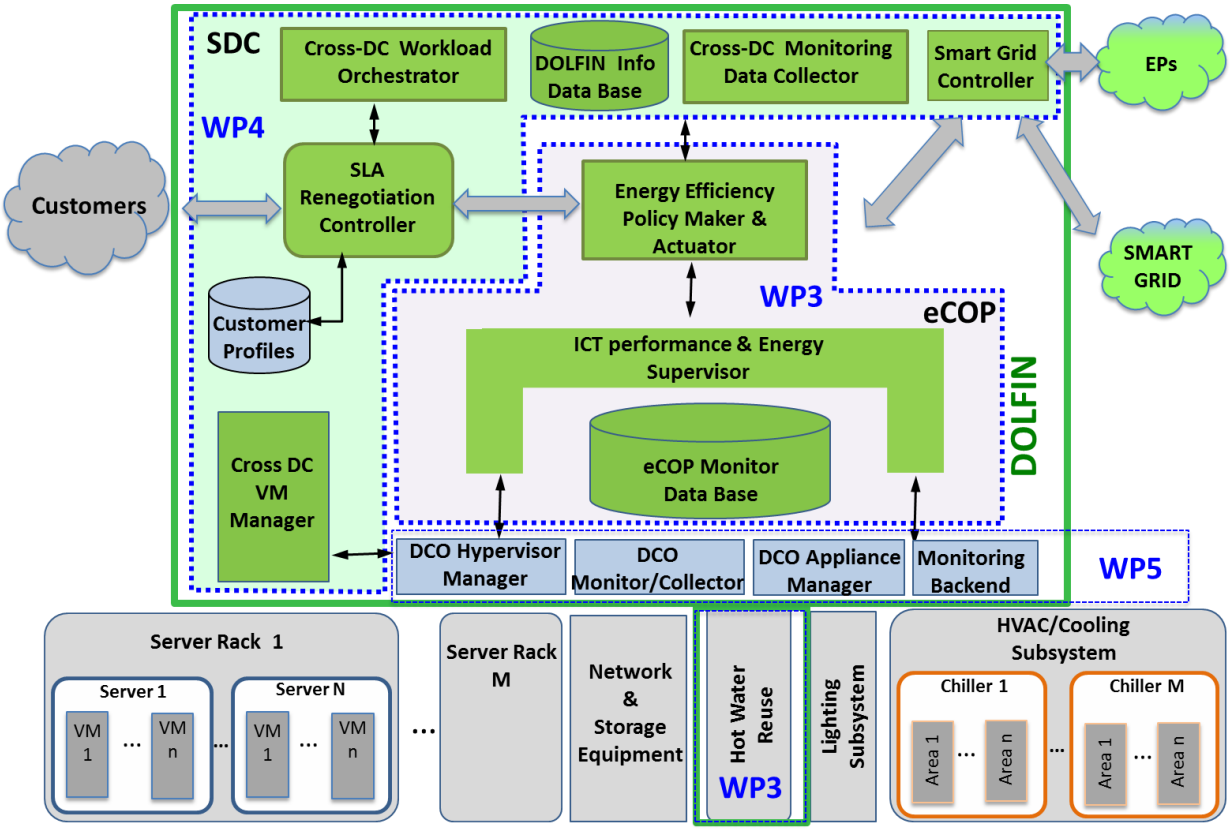

Figure 1-4: Overall DOLFIN architecture

The modules identified in the architecture are listed in two groups that identify the elements specially developed in the context of DOLFIN and those that represent a typical asset in a standard DC environment, but request some adaptation to be used in DOLFIN.

1.1.1 New DC functionality offered by DOLFIN specific components

This section describes a list of components that have been introduced by DOLFIN in the DC environment and provide the following functionality:

- **eCOP Monitor Data Base**: it has the sole purpose of storing all real-time and historical energy related data collected from DCs. This information is used for energy efficiency decisions and VM load predictions, along with data from all the components within a DC architecture.
- **ICT Performance & Energy Supervisor**: it focuses on the analysis of performance monitoring data and energy data. The objective of this component is to provide information on the actual performance of the applications (typically VMs) utilizing the resources of the DC devices and the energy consumed by the non-ICT components.
- **Energy Efficiency Policy Maker & Actuator**: this is the most intelligent part of the eCOP that makes the decisions realizing the requested policy of the DC. Following a spiral optimization approach in the context of DOLFIN: at Servers' rack level, at DC segment level, at DC level and finally at DOLFIN level, this component realizes the 3 first levels of the energy optimization, achieving optimization at DC level. Moreover, it aims at initiating a stream of control commands, which can be translated into actual actions by other DOLFIN subsystems within a DC.
- **Cross-DC Monitoring Data Collector**: it collects knowledge not only from the Synergetic DC resources, but also from the network routers and in general from the network resources point of view.
- **Cross-DC Workload Orchestrator**: it is a distributed software element that gets the SDC decisions and does most of the different types of resource optimisation including energy optimisation and management of their trade-offs in a cross DC optimization scenario.
- **Cross-DC VM Manager**: it realises a DC Interconnect (DCI) interface and performs the actual migration of VMs cross DCs.
- **DOLFIN Information Data Base**: it offers an abstracted and logically-centralised information manipulation (including information collection, aggregation/processing, storage / indexing and distribution) across all DOLFIN architectural components. It includes information on the local energy requirements e.g. Demand/Response requests from the local Smart Grid operator
- **SLA Renegotiation Controller**: it will take into account the existing approaches in modelling SLA criteria (e.g. the overall cost of an offered service) and augment these approaches with the use of energy related criteria. This module will also handle the negotiation process between the DC and the end-users.

1.1.2 Revised DC functionality offered by components modified by DOLFIN

This section describes a list of components that already exists in a DC environment and they are revised in order to provide the following DOLFIN specific functionality:

- **DCO Hypervisor Manager**: it is an adaptation layer that maps the high-level decisions taken by the DOLFIN system into low-level technology-dependent commands that control the DC ICT

infrastructure. The DCO Hypervisor Manager analyses the new DCs ICT hardware including DCs configuration and status. It is responsible for managing new ICT hardware and VMs configuration. In addition, it is also responsible for the translation of new hardware configurations to dedicated commands to manage the specific virtualization hypervisor and the execution of commands to provide a new DCs hardware configuration.

- **DCO Appliance Manager**: the counterpart of the DCO Hypervisor Manager, on the non-ICT DC infrastructure. The DCO Appliance Manager adapts and guarantees the execution of strategies elaborated by DOLFIN expert system on the non-ICT infrastructure by translating high level commands into low-level technology-agnostic actions.
- **DCO Monitor/Collector**: it is to interface with both the ICT and the non-ICT DC infrastructure and collect all operational and energy related information to be stored in the eCOP Data Base.
- **Customer profile management**: it provides the information for describing the requirements that need to be satisfied for service provision to the end customers. The Customer profile is strictly related to the service agreement contracted between customer and DC owner and directly affects the actions may be activate on the user resources.

1.2 Energy Models for DC Software Platforms

The energy models that will be used by the various components of the eCOP are common throughout the whole DOLFIN Software Platform and are presented in full details in deliverable D4.1 "*Synergetic Data Centres for energy efficiency (Design)"*. In this deliverable, we briefly summarize the proposed KPIS for energy consumption since these have influenced the specification of eCOP APIs and database contents. The rest of the discussion on energy models and results is omitted in in this deliverable for the sake of clarity and brevity.

1.2.1 Proposed KPIs for Energy Consumption

The KPIs use the energy consumers plus various coefficients that can be adapted to the hardware used, i.e., whether server processors, network cards and memories are energy efficient or not, or whether they support the multiple states.

Some of the proposed KPI functions in DOLFIN are:

$$\textbf{Physical Server Energy Consumption} = a + b * \textit{server processing load} + c * \textit{server memory utilization} + d * \textit{server network traffic}$$

$$\textbf{Average VM Energy Consumption} = \frac{\textbf{Physical Server Energy Consumption} - a}{\textbf{Number of VMs\_Hosted\_In\_this\_Server}}$$

$$\textbf{VM Energy Consumption} = b * \textit{VM processing Load} + c * \textit{VM memory consumption} + d * \textit{VM network traffic}$$

$$\textbf{Average Application Energy Consumption} = \frac{\boldsymbol{VM\ Energy\ Consumption}}{\boldsymbol{Number\_of\_applications\_running\_in\_this\_VM}}$$

$$\textbf{Application Energy Consumption} = b * Application\ processing\ load + c * Application\ memory\ consumption + d * Application\ network\ traffic$$

The coefficients used in the above KPIs are highlighted here. They represent multipliers of various consumptions. In the following a, b, c, d are hardware related.

- a = baseline energy consumption (consumption of server with 0 virtual machines).
- b = energy consumption per percentage of CPU load
- c = consumption per byte of memory used
- d = consumption per byte communicated

When using the KPIs to determine energy usage, we expect to adjust the values to the underlying real hardware.

Many more KPIs can be derived for these resources, and we expect to see more over time as more knowledge is acquired w.r.t modelling of virtual resource energy consumption.

2 DOLFIN Distributed Components Interfaces and APIs Design

This section focuses on the adaptation of REST as a communication mechanism for the request / response interactions between the components of DOLFIN in the eCOP and SDC level[1]. We have investigated the use of RESTful APIs, and have determined that REST provides flexibility with regard to languages and tools, that should be exploited during the design of the component interfaces.

The types of interaction we envisage are:

- eCOP ←→ eCOP components
- eCOP → SDC components
- SDC → SDC components
- eCOP ←→ Smart Grid Controller
- Inter domain SDC communication

The following design assumptions need to be highlighted:

i) The RESTful approach is an instance of the request-response (R-Q) model of communication, and as such is only suitable for interactions that fit the R-Q model.

ii) The RESTful approach is lightweight, allowing loosely-coupled interactions, and is used for many service-based architectures where it is the most commonly used method, by a considerable margin.

iii) The adaptation of a RESTful approach for every component interaction in a distributed network environment is infeasible - e.g. monitoring may use some real-time distribution framework.

Furthermore, REST is not a ready-to-go solution, it is a model of interaction. Consequently, each of the system elements must be carefully designed.

2.1.1 RESTful APIs

Currently the use of RESTful APIs is being considered as the state-of-the-art approach for many service-oriented platforms. Although, the first web-service APIs such as XML-RPC and SOAP used a remote

[1] Since DOLFIN aims to be presented as an integrated software platform offering distributed cloud services among cooperating DCs, the interface design principle and approach have been intentionally kept common both for eCOP and SDC.

procedure call (RPC) mechanism using XML over HTTP, these have been over-shadowed by the RESTful style of interaction.

REST-style architectures comprise clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically some encoded data that captures the current state of a resource.

RESTful architectures allow:

- Scalability of component interactions
- Generality of interfaces
- Independent deployment of components
- Intermediary components to reduce latency, enforce security and encapsulate legacy systems

The REST architectural style simplifies the component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries—proxies, gateways, and firewalls—to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability.

The major advantage of REST is that it is identical to that of the Web. Any client can talk to any server, regardless of the technologies used underneath to handle the requests and the responses governing the communication between the clients and the servers. For more information, analysis and comparison with other available HTTP technologies for web services, the interested user is requested to skip through references [2], [3], [4], [5], [6] and [7].

The REST design principles well align with the eCOP interface design needs and represent the more convincing choice in terms of impact, portability and adaptation to existing legacy contexts.

2.1.2 Publish Subscribe mechanisms

In contrast to the RESTful approaches, where synchronous messaging is achieved (a client must always initiate the communication process with a server), the publish/subscribe (pubsub) model allows for asynchronous communication between server and interested client entities through notification events. The client applications or “subscribers” express interest in certain types of information, while the server applications or “publishers” provide information to the system, which is available only to authorized subscribers. The communication between clients and servers is achieved via a mediation service, which receives publication requests, broadcasts event notifications to subscribers, and enables privileged entities to manage lists of authorized participants. The main characteristics of the pubsub model are the following [8]:

- ***anonymity:*** the server applications or publishers of information are required to publish messages without explicitly specifying recipients or having knowledge of intended recipients. Similarly, the client applications or subscribers must receive only those messages which they have interest in.
- ***time decoupling:*** communication between servers and clients is asynchronous, so that they do not need to be active simultaneously

- ***flow decoupling:*** Publishers and subscribers are not blocked during the production or consumption of notification events. As events' production and consumption do not happen in the main flow of control of the publishers and subscribers, communication can be asynchronous.

Individual point-to-point and synchronous communication architectures pose scalability challenges, leading to rigid and static applications. On the contrary, pubsub has the following benefits over alternative architectures, as underscored by Google [9] and Oracle [10]:

- It is architecturally flexible: numerous publishers and subscribers may be added in order to expand the messaging capabilities of a certain interconnected system.
- It is easily configurable: they can be easily configured to support different subscription models.
- It requires very minimum resources while exhibiting superb performance and scalability.

Commonly used pubsub/message queuing software bundles include RabbitMQ [11], MQTT [12] and Apache Kafka [13]. Interestingly, many of today's state of the art platforms with a Cloud management perspective, including OpenStack [14], which acts as the base Cloud management platform to be used in the context of DOLFIN.

The pubsub and distributed message queue systems available in state of the art (e.g. RabbitMQ) represent a valuable background to be re-used for the implementation of similar mechanisms in the eCOP modules, focusing DOFLIN developments on the customization of the message semantics more than on the implementation of the transport back-ends.

2.2 Applicability in DOLFIN

In need for a large variety of appropriate information exchange mechanisms, the DOLFIN Software Platform specification has determined that both synchronous and asynchronous approaches are required in order to guarantee proper systemic operation.

Specifically, as will be detailed in the next sections, the ICT Performance and Energy Supervisor Core component (documented in paragraph 3.3.1.2.1) features a publish-subscribe server that will allow for the rest of the eCOP (e.g. the Optimizer of the Energy Efficiency Policy Maker and Actuator presented in 3.3.2.2.5) and SDC (e.g. the SLA Renegotiation Manager detailed in D4.1) components to subscribe for streams of events and get instantaneous updates on the status of the DC, without having to continuously poll for information.

On the other side, RESTful APIs are exposed by all components of DOLFIN in order to enable synchronous cooperation among them.

In Figure 2-1 we depict the scope of REST within DOLFIN. We see the main interactions, whereby some components have their own private communication with the entities being managed, e.g. with the monitoring backend. Components that are themselves distributed systems, yet are still one component, may have their own internal communication approach.

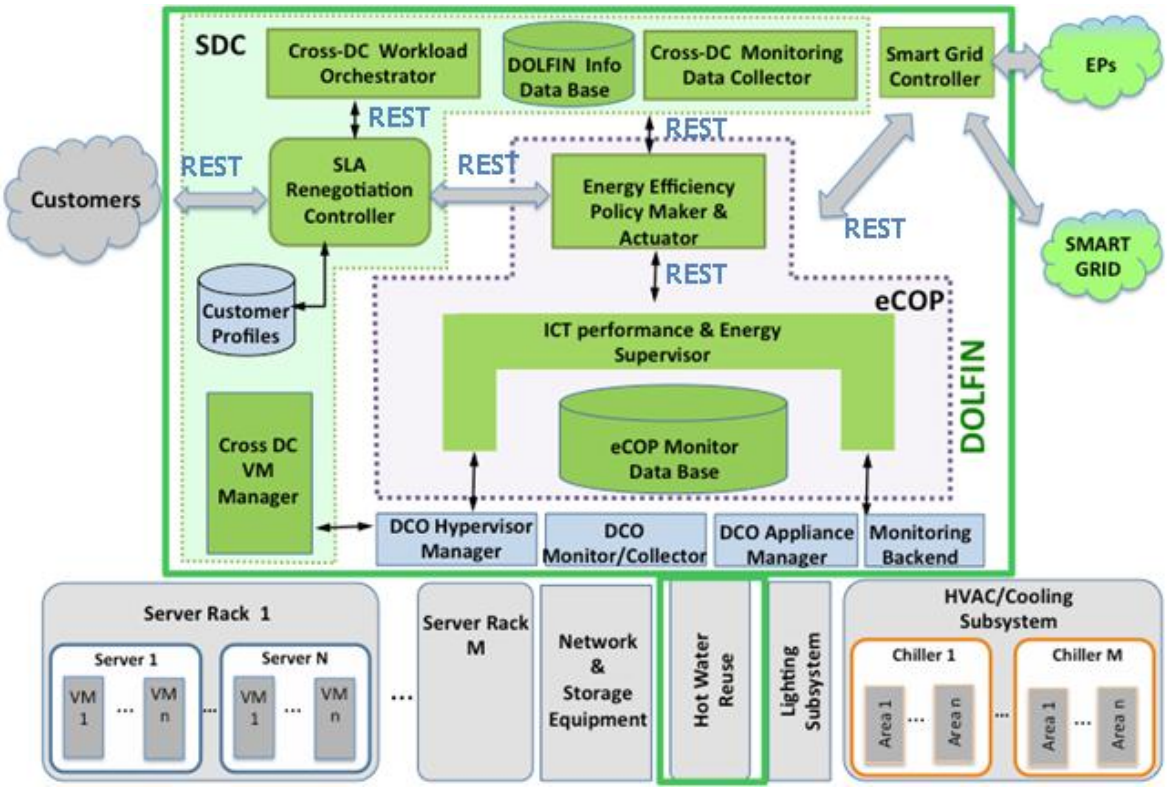

Figure 2-1: REST use in DOLFIN

2.3 DOLFIN Control Flows

Complete DOLFIN control flows involve all components featured by the general DOLFIN architecture. Hence, a broad view of the system architecture is necessary in order to present them and their presentation should refer to both eCOP and Synergetic DC (SDC) components. Based on this remark, several indicative control flows are presented in deliverable D4.1 entitled “Synergetic Data Centres for energy efficiency (Design)” in the form of consolidated control flows for the entire DOLFIN Software Platform. Their presentation in this deliverable is, therefore, omitted for reasons of brevity and clarity.

3 Energy Consumption Optimization Platform (eCOP)

3.1 eCOP functional description

The eCOP is the core optimization group of components that grants a DOLFIN-enabled DC with advanced monitoring, self-evaluation and self-optimisation capabilities. The four pillars of eCOP functionality are summarized as follows:

1. Interfaces legacy/heterogeneous DC sub-systems (including cloud management platforms and building management system platforms) via specific adapters to collect all necessary information for the optimization process.

2. Provides an engine able to process raw data and calculate appropriate metrics and KPIs that will be used throughout the whole the optimization process. This engine resides in the ICT Performance and Energy Supervisor module.

3. Stores raw data, measures and calculated metrics into the eCOP DB, and provides also efficient and abstracted interfaces to support DB operations.

4. Sets the Energy efficiency Policy Maker & Actuator module into action to perform the optimization lying in the core of the DOLFIN project scope. The module activation is based on pre-defined criteria and aggregated information from other modules (e.g. ICT Performance and Energy Supervisor, Smart Grid Controller, Cross-DC orchestrator, etc.).

The table below provides an overview of the functions performed by each composing module of eCOP and the dependencies with other modules in the DOLFIN architecture.

Table 3-1: Overall modules functions

Module	Function	Modules dependency
ICT Performance and Energy Supervisor	Collects ICT devices (hosts and VMs) performance information (i.e. CPU loads, storage usage, network link utilization)	DCO Monitor/Collector
	Collects non-ICT facility energy consumption (i.e. collects measures of power (W) and current (A) utilized by the facilities)	DCO Appliance/BMS
	Interacts with legacy VMs manager to collect information on VMs deployment and retrieving in near real-time the variation of the VMs assets	DCO Hypervisor Manager

	(e.g. after a consolidation or migration)	
	Implements a script based engine to build and calculate meaningful KPIs based on the various collected values	
	Interacts with eCOP Monitor Database to: (1) store calculated KPIs, (2) retrieve data needed for internal calculation algorithms. Moreover, it provides adapters for efficiently storing raw data from legacy DC sub-systems	eCOP Monitor Database
	Provides interfaces to retrieve KPIs and other relevant data for the Energy Efficiency Policy Maker & Actuator.	
	Provides a list of classified servers based on their energy efficiency and DC layout	Energy efficiency Policy Maker & Actuator
	Provides the energy consumption of each server	
Energy efficiency Policy Maker & Actuator	Implements optimization rules based on Smart Grid controller input. Two primal optimization approaches are followed: (1) based on the energy consumption requests, (2) based on the energy price.	Smart Grid Controller
	Interfaces the SLA Renegotiation Controller with respect to requests for SLA reduction on specific VMs.	SLA Renegotiation Controller
	Requests for cross-DC VM migration actions based on the results of internal policy evaluation.	Cross-DC Workload Orchestrator
	Interacts with legacy DC cloud management system to perform VM-oriented actions on local(within DC) level	DCO Hypervisor Manager
	Provides a policy engine coordinating the actions needed to be performed in order to achieve energy optimization	
eCOP Monitor Data Base	Provides generic interfaces used by other modules to query and put data into the database	
	Implements the data model needed to store the raw data, KPIs and aggregated data	
	Implements an engine to build and store aggregated data streams (i.e. calculates average values with different resolution)	

Due to the complex nature of the DC architecture and the optimization approach introduced by DOLFIN, the definition of a general architectural design model is required in the context of which specific data manipulation and reaction steps need to identified, in order to come up with an autonomous system exhibiting optimisation capabilities with respect to the minimisation of both energy and computational resources. In particular, the following steps have been identified and are being modelled in the context a general control framework:

- Data collection

- Monitor
- Analysis
- Planning
- Actuation

These steps implement a typical control model known as *control loop.* The latter identifies a basic mechanism according to which any action applied to the system produces a reaction, evaluated in terms of state changes of the monitored objects that produce new data to be processed. In such a control loop, the presence of an analysis and planning plane is evidently essential. This plane, apart from the definition of actions to be implemented, is responsible for the mitigation of the potential effects of oscillatory instability produced by the reaction mechanisms. As reported in subsequent paragraphs, a big part of the analysis and planning plane of the control loop is covered by the functions implemented by the Energy Efficiency & Policy Maker components. The other system components, such as the ICT Performance and Energy Supervisor, close the loop, supporting the requirements for data collection and data meta-processing.

In the following paragraphs, a functional decomposition of each DOLFIN component in the Energy Consumption Optimization Platform is attempted. Moreover, specific data flow models are introduced to assist the depiction of the relationships among interfacing modules and subsystems.

3.1.1 eCOP architecture decomposition

As described in the previous sections, the eCOP subsystem is structured in three major macro components. During the design process, these primarily modules are decomposed in specialized entities that are detailed in the subsequent sections. A brief overview is reported below:

- ***The ICT Performance and Energy Supervisor*;** the objective this component is to provide analysis and monitor functions, getting information on the actual performance of the applications (typically VMs) utilizing the resources of the DC devices and the energy consumed by the non-ICT facilities. The ICT Performance and Energy Supervisor is structured in :

  - ***ICT Performance and Energy Supervisor core*;** coordinates the communications between the other sub-modules and exposes multiple interfaces used by other higher-level DOLFIN components such as the Energy Policy Maker & Actuator. The module works as stateful data hub, continuously interfacing with a low number of other actors, and making low utilization of computational power.

  - ***Metrics Engine;*** produces updated metric values which are then made available to the ICT Performance and Energy Supervisor core sub-component and, finally, to the other DOLFIN modules. It is designed to run on-demand and deals with large sets of data; computes the metrics in one run, processing big amounts of data**.**

  - ***DCO Hypervisor Broker;*** collects data coming from the DCO Hypervisor manager, pertaining also to the status of managed DC Hypervisors and hosted applications (VMs).

  - ***DCO Monitor/Collector Broker;*** collects data coming from the DCO Monitor / Collector component. This stream of data contains updates about the status of hardware DC resources such as physical hosts, PDUs, ICT appliances, etc.

  - ***DCO Appliance Broker;*** deals with data related to non-ICT resources that are part of the DC, such as HVAC systems, ancillary and non-correlated energy consumption, temperature, computer room conditions, etc.

- ***ICT Topology Graph Database;*** contains an updated version of the data describing the physical distribution and topology of the DC assets.

- **The Energy efficiency Policy Maker & Actuator;** is the part of the eCOP where are taken the decisions and performed actions for the energy optimization, based on pre-defined criteria and the input by other DOLFIN modules (such as the Smart Grid Controller). The elements in the Energy efficiency Policy Maker & Actuator are:

  - ***VM Priority Classifier;*** is a component assessing the priority of VM streams based on their time criticality, their resource consumption and the SLA of the customers or other entities (e.g. cooperating DC).

  - ***Prediction Engine;*** provides forecasts regarding the load expected in the near future, based on the status of both the sole DC or as a part of a federated set of DOLFIN-enabled DCs. The results of prediction process are sent and used by the modules involved in the optimization phase.

  - ***Policy Repository;*** maintains information regarding all possible policies available to conduct the operation efficiency resource management, such as the energy policy used internally by the eCOP functions and the policy agreements for federated DCs.

  - ***Policy Maker;*** is the key component of the Energy Policy Maker and Actuator subsystem and is responsible for scheduling the activation of the policy enforcement, on the basis of the DC status and the information provided by other modules. This module is responsible for the efficient resource management and the acceptance or rejection of incoming requests at local or synergetic DCs level.

  - ***Optimizer;*** the role of the Optimizer is to optimize the allocation of existing and accepted load to the DC physical resources, while adjusting to the operation imposed by the selected policy, also taking into account the forecast of the Prediction Engine on the near future load and/or other information such as the Smart Grid status by the Smart Grid Controller.

  - ***Policy Actuator;*** is responsible for the implementation of the actions identified by the Optimizer and translates the optimizer plan into commands to the DCO Hypervisor Manager and/or the DCO Appliance Manager.

- ***The eCOP Monitor Data Base*;** provides the groups of functionalities and entities for storing the real-time and historical energy related data collected from DCs. This information are 'distributed' to all DOLFIN entities to perform their specific operations, for example are used by the Energy efficiency Policy Maker & Actuator to perform the prediction and optimization analysis for the efficient allocation of VMs. The eCOP Monitor Database consists of two active internal components:

  - ***eCOP Database***; consisting the basic persistence layer of the eCOP Monitor Database.

  - ***Storage Broker***; wrap the storage layer with a comprehensive RESTful API, providing access to and from other DOLFIN entities.

3.2 Modules interfaces

Focusing on the eCOP DOLFIN sub-system, the interactions and interfaces of the constituent modules are summarized in Figure 3-1. Each interface defines a host (module that exposes an interface) and a consumer (module that consumes an interface) of specifics APIs.

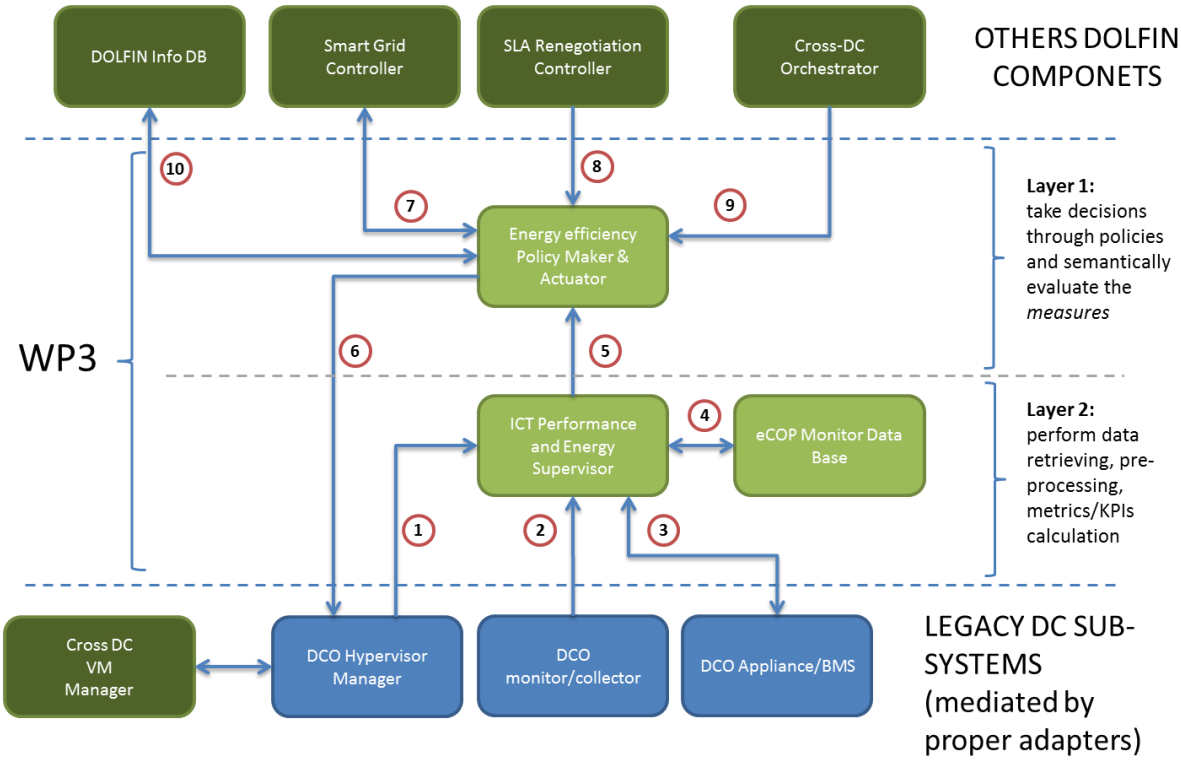

Figure 3-1: Overview of the interactions of the eCOP DOLFIN subsystem

The table below provides an overview of each interface, describing its host and consumer.

Table 3-2: WP3 modules interfaces

IF. #	Host	Consumer	Description
1	DCO Hypervisor Manager	ICT Performance & Energy Supervisor	Get VMs info and deployment structure (how the VMs are allocated)
2	DCO Monitor/Collector	ICT Performance & Energy Supervisor	Get ICT and non-ICT info (performance, energy consumption, etc.)
3	DCO Appliance/DMS	ICT Performance & Energy Supervisor	Get non-ICT info and performs controls
4	eCOP DB	ICT Performance & Energy Supervisor	Query interface to INSERT and RETRIEVE data
5	ICT Performance & Energy Supervisor	Energy Policy Maker & Actuator	Interface to access to performance/energy data/metrics and topology description
6	DCO Hypervisor Manager	Energy Policy Maker & Actuator	Interface to command action on VM sub-system (i.e. VM migration)
7	Smart Grid Controller	Energy Policy Maker & Actuator	Interface to GET the negotiated energy strategies
8	SLA Renegotiation	Energy Policy Maker &	Interface to negotiate SLA reduction

	Controller	Actuator	
9	Cross-DC Orchestrator	Energy Policy Maker & Actuator	Interface to query cross DC migration
10	DOLFIN Info DB	Energy Policy Maker & Actuator	Retrieve the static DC structure Logging EPM&A actions

As can be observed by Table 3-2, each module is responsible for the implementation of certain functions. These functions will be exposed to the services outside eCOP through appropriate APIs. It should be noted that Table 3-2 does not provide an exhaustive presentation of the APIs but, rather overviews the interactions between the respective modules. The attempted overview allows to:

- Model the data and control flows, following the interactions between modules.
- Highlight critical issues that should be addressed during the interaction. For example whether the data flow should be synchronous or involve some sort of acknowledgment.
- Visualize which components - DOLFIN components or external components (e.g. from the legacy DC) - are involved in the various communication processes.
- Offer an outline of each component operation, to proceed to the internal component design, with the detailed definition of the communication APIs.

3.2.1 Controls and data flows

The current section provides an abstract representation of the communication processes between various eCOP-related DOLFIN components, through the schematization of the controls and data flows with the supports of sequence diagrams. These diagrams refer to the interfaces depicted in Figure 3-1 and tabulated in Table 3-2. For each interface a brief description is provided, elaborating on the host and consumer components[2]. Moreover, the sequences are used to provide an immediate characterization of the nature of the interfaces and of the communication model that is expected to be adopted. For example:

- Some interfaces require a model based on SUBSCRIBE/NOTIFY approach. In this case, it is expected to identify an API able to (1) support the registration of “consumer” objects and (2) provide asynchronous events publication.
- Other interfaces should be able to support synchronous and reliable communication, to ensure the correctness of the exchanged information between the involved entities.

These interfaces provide a high level and basic abstraction of the APIs that will be subsequently defined in the dedicated modules' design paragraphs.

[2] For brevity and accuracy reasons, and due to the fact that in many cases the host-consumer hierarchy is vague, an alternative notation is used in the following, annotating the components as “Component A” and “Component B”, explicitly avoiding to use the terminology host and consumer.

3.2.1.1 Interface to handle VM deployment (topology) info

This interface provides the function to handle the information regarding the deployment of VMs within the DC. The DCO Hypervisor Manager provides an event-based interface that will trigger events upon some changes in the VM structure.

- **Interface number: #01**
- **Component A: ICT Performance and Energy Supervisor**
- **Component B: DCO Hypervisor Manager**

In this context is assumed initial steps where the ICT Performance and Energy Supervisor make a binding to the events provider. The subscription step is not a strictly required in a real implementation (for example might be introduce some mechanism to mutual subscribe the two components) but has the meaning of define a communication model where we need a clear identification of the producer and consumer.

In the paragraph 3.3.1.2.3 we introduce the design model used to manage such event-driven communication**.**

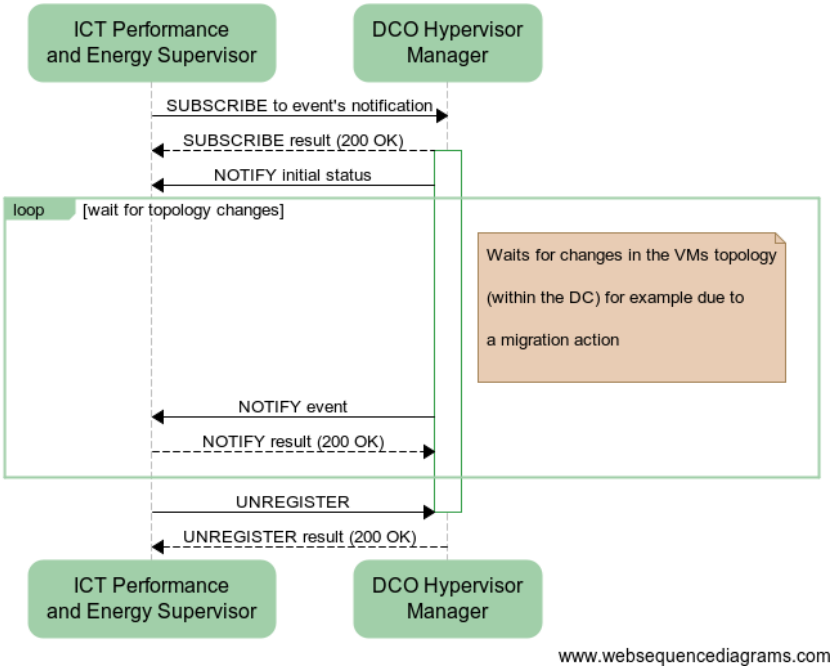

Figure 3-2: Interface to handle VM deployment (topology) info

3.2.1.2 Interface to handle ICT device performance data stream

This interface provides the function to handle the ICT device performance data stream. The DCO Monitor/Collector provides an event-based interface that will trigger new events on changes.

- **Interface number: #02**
- **Component A: ICT Performance and Energy Supervisor**
- **Component B: DCO Monitor/Collector**

The role of DCO Monitor/Collector is to export information regarding the status of hardware DC resources such as physical hosts, PDUs, ICT appliances, etc. The flow diagram is similar to what defined in the previous one but uses specific APIs due to the different nature of monitored objects.

A detailed description of the model and internal entities involved in the communication are reported in paragraph 3.3.1.2.3.

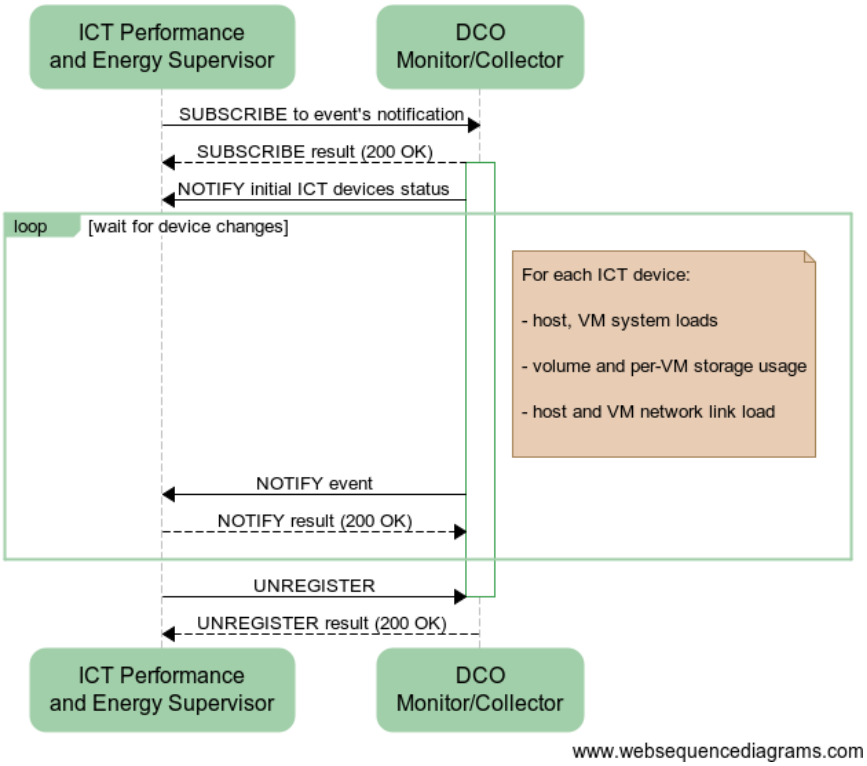

Figure 3-3: Interface to handle ICT device performance data stream

3.2.1.3 Interface to handle consumption on non-ICT facilities

This interface provides the function to handle the consumption on non-ICT facilities. The DCO Appliance/BMS provides an event-based interface that will trigger new events on changes.

- **Interface number: #03**
- **Component A: ICT Performance and Energy Supervisor**
- **Component B: DCO Appliance/BMS**

The flow diagram reports an event-driven interaction between the ICT Performance and Energy Supervisor and dedicated appliance that are supposed to be present inside the DC infrastructure and that are able to handle non-ICT facilities, such as HVAC systems, ancillary and non-correlated energy consumption, temperature, computer room conditions, etc.

Interfacing with DCO Appliance/BMS, the ICT Performance and Energy Supervisor will be able to collect extra information that (before the introduction of the DOLFIN approach) were commonly considered unrelated from other consumption resources. Instead DOLFIN try to correlate the status and consumption from non-ICT facilities with other one (for example the whole energy consumed in a rack space or a physical HW, etc.) and use the results of that correlation during the optimization process.

As explained in the preview paragraphs, a detailed description of the communication model is reported in 3.3.1.2.3.

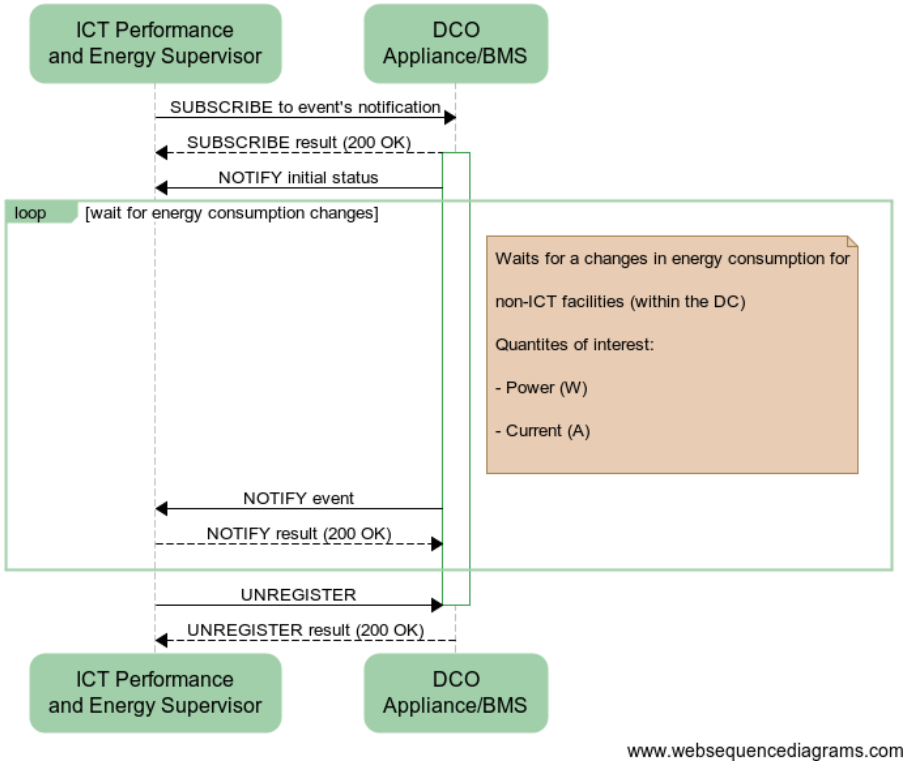

Figure 3-4: Interface to handle consumption on non-ICT facilities

3.2.1.4 Interface to eCOP DB

This interface provides the function to SET and RETRIEVE datasets from the eCOP Data Base.

- **Interface number: #04**
- **Component A: ICT Performance and Energy Supervisor**
- **Component B: eCOP DB**

The energy optimization approach used in DOLFIN require the management of different kind of information: (i) raw data from ICT and non-ICT facilities, (ii) calculated metrics and KPI, (iii) DC topology assessment, etc. These information need to be stored and retrieved when necessary, for example when a new optimization process has to be performed. For this scope the eCOP Monitor Database define dedicated APIs (reported in Annex 1) that will be used by all the DOLFIN components that need query the DB.

The paragraph 3.3.1 reports a detailed description of model functionalities provide by the eCOP DB.

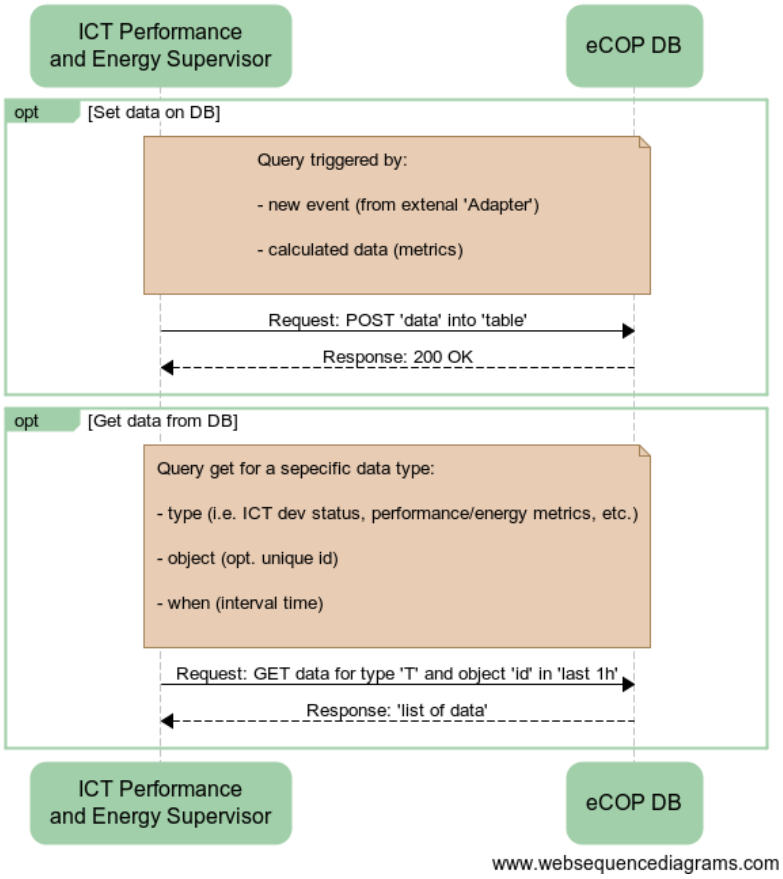

Figure 3-5: Interface to eCOP DB

3.2.1.5 Interface to get changes in VMs and metrics

This interface provides a group of APIs used by the Energy Policy Maker & Actuator to request updated information on the VMs deployment and KPIs, calculated by the ICT Performance and Energy Supervisor.

- **Interface number: #05**
- **Component A: Energy Policy Maker & Actuator**
- **Component B: ICT Performance and Energy Supervisor**

As briefly defined in the previous paragraphs the ICT Performance and Energy Supervisor is the entity in the DOLFIN architecture that provide the physical interfacing to the underlined DCs infrastructure, perform metrics calculation and build other relevant information used during the energy optimization process.

In this model the ICT Performance and Energy Supervisor is in charge to perform a lot of short-time consumed operation, in contrast with the operation performed by the Energy Policy Maker & Actuator. In this flow diagram we reuse the model of event-driven mechanism to trigger the activities of the Energy Policy Maker & Actuator only when changes are detected on objects explicitly required.

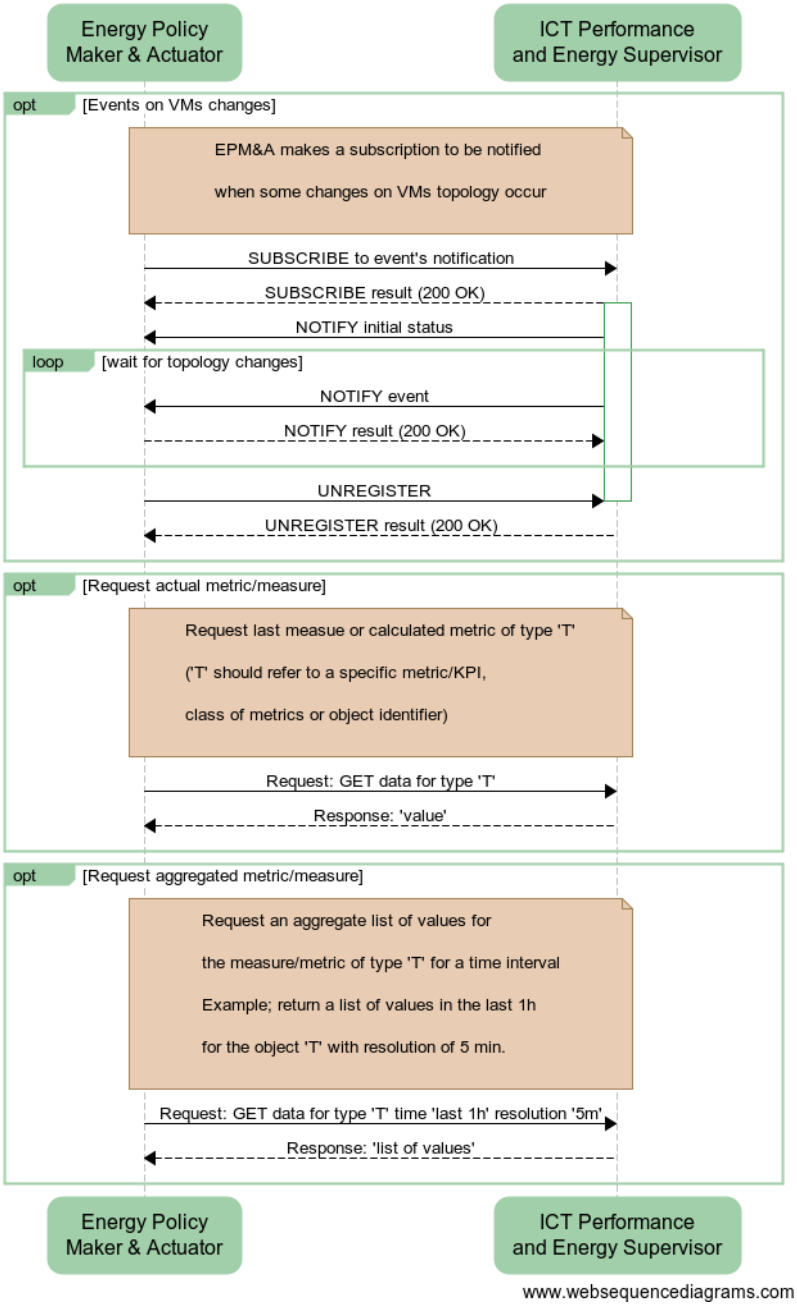

Figure 3-6: Interface to get changes in VMs and metrics

3.2.1.6 Interface to perform actions on local cloud management platform

This interface is used by the Energy Policy Maker & Actuator when there is a need to apply a set of actions on a local cloud management platform (such as OpenStack) as a result of the evaluation of an energy optimization process.

- **Interface number: #06**
- **Component A: Energy Policy Maker & Actuator**
- **Component B: DCO Hypervisor Manager**

The flow described in the below diagram represent the last stage in optimization process, where the calculation performed inside the Energy Policy Maker & Actuator are translated in a stream of operations that might request some actions on the VMs sub-system.

Although many components are involved during this stage, a lot of complexity is confined inside of the Energy Policy Maker & Actuator, but this diagram has the primarily scope to highlight the interfaces needed to request action on the VMs sub-system via the DCO Hypervisor Manager.

The detailed description of this process is reported in the section 3.3.2.

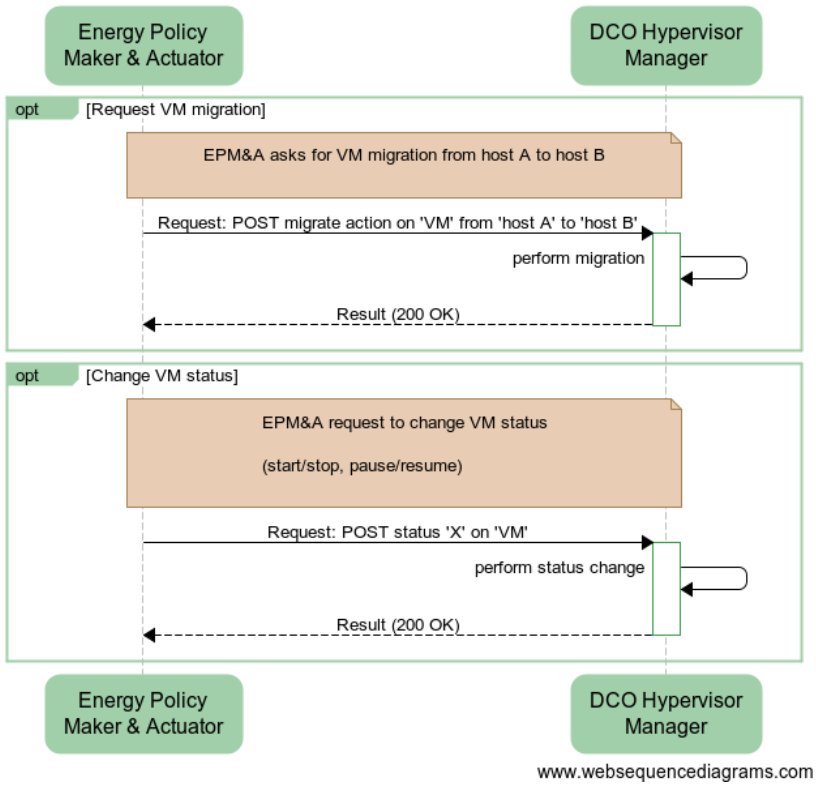

Figure 3-7: Interface to perform actions on local cloud management platform

3.2.1.7 Interface to Smart Grids

This interface provides the link between the Energy Policy Maker & Actuator and the Smart Grid Controller. Its main roles to enable information exchange regarding the status of the negotiations with the EPs and the general Smart Grid environment.

- **Interface number: #07**
- **Component A: Energy Policy Maker & Actuator**
- **Component B: Smart Grid Controller**

The Energy Policy Maker & Actuator uses the information retrieved by this interface to "adapt" the optimization process. As explained in paragraph 3.3.2.2.4, the Policy Maker components should implement a pre-analysis based various inputs, including Smart Grids energy/price requests.

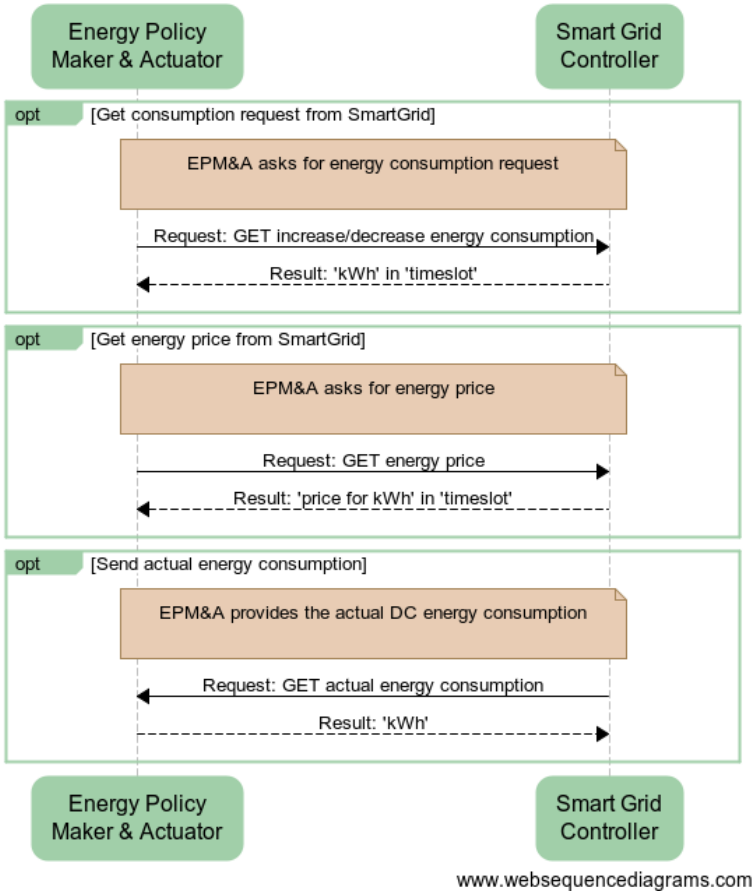

Figure 3-8: Interface to Smart Grids

3.2.1.8 Interface for SLA renegotiation

The Energy Policy Maker & Actuator uses this interface to request the customers that have agreed to a possible SLA reduction. The SLA Renegotiation Controller provides a list of possible candidates (VMs) that could be subject to lower performance SLAs as well as the maximum time duration for this performance reduction.

- **Interface number: #08**
- **Component A: Energy Policy Maker & Actuator**
- **Component B: SLA Renegotiation Controller**

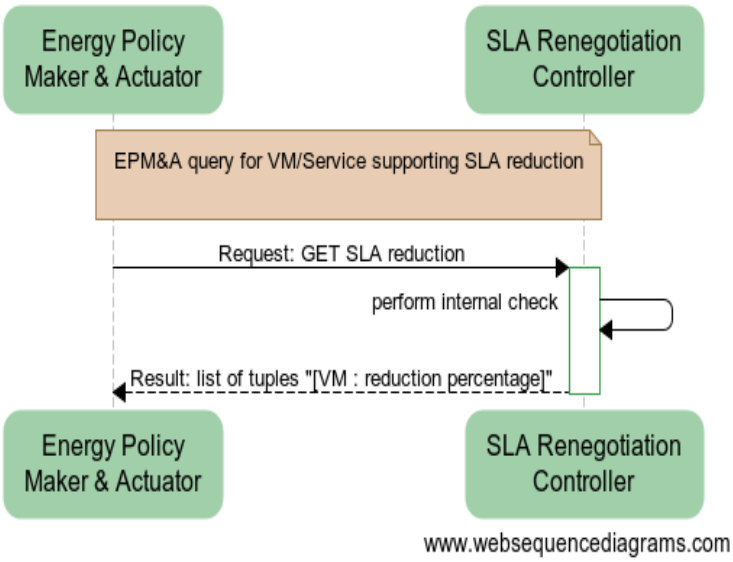

Figure 3-9: Interface for SLA renegotiation

3.2.1.9 Interface for cross-DC VM migration

This interface allows for requesting and performing a cross-DC migration. The process is supported by the Cross-DC Workload Orchestrator which is in charge of negotiating with the federated DCs.

- **Interface number: #09**
- **Component A: Energy Policy Maker & Actuator**
- **Component B: Cross-DC Workload Orchestrator**

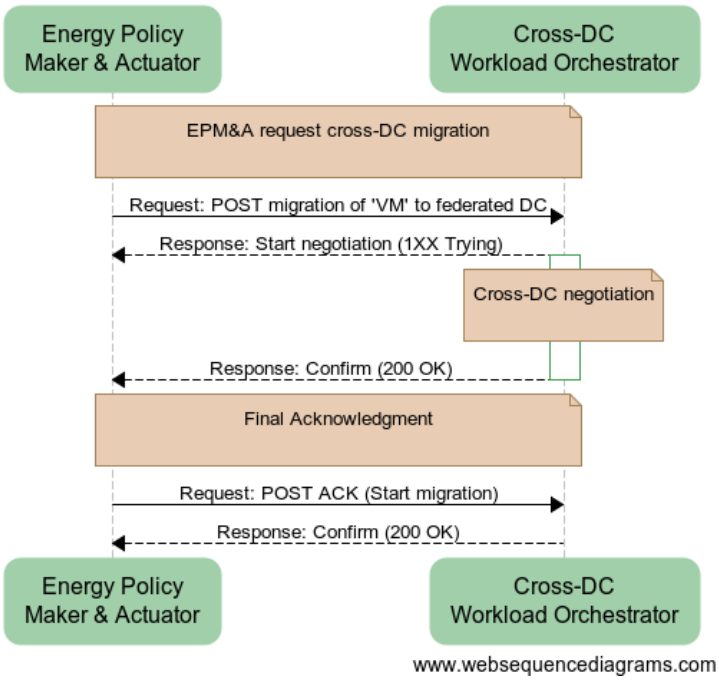

Figure 3-10: Interface for cross-DC VM migration

3.2.1.10 Interface to DOLFIN Info DB

The DOLFIN Info DB provides global information on DC characteristics and components and provides also the API for logging the activities and actions performed by the DOLFIN modules. Moreover, the Info DB, stores the asset information for the federated DOLFIN-enabled DCs that might be involved during an optimization process.

- **Interface number: #10**
- **Component A: Energy Policy Maker & Actuator**
- **Component B: DOLFIN Info DB**

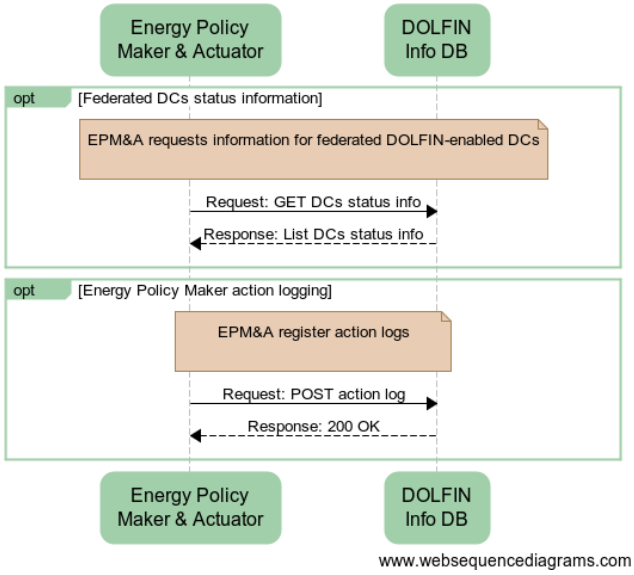

Figure 3-11: Interface to DOLFIN Info DB

3.3 Internal modules design

In the following sections a detailed description of the internal design of each module is provided, in the context of WP3.Specifically, the present section elaborates on the detailed design of:

- The ICT Performance and Energy Supervisor.
- The Energy Policy Maker and Actuator.
- The eCOP Database.
- The design of the Water Cooling Server module also detailing the implementation of the prototype.

Moreover, each section provides a detailed description of the functionalities performed by each component, along with the identification of the primary sub-elements within the component and their responsibilities and role in the optimization process.

The design of the elements is accompanied by the definition of the APIs interaction between system elements, such as the APIs for accessing (monitor and control) of resources exposed by the legacy of the DC modules, the APIs to put and retrieve information from the ECOP DB, the API for accessing the functions provided by the SDC (Synergetic DC, described in the context of WP4) subsystem, etc. The complete list of APIs identified at the design stage are described and reported in the Annex of this document.

3.3.1 ICT Performance and Energy Supervisor

The main purpose of this component is to provide information on the actual performance of the applications (typically VMs) utilizing the resources of the DC along with information on the energy consumption of both ICT and non-ICT subsystems. Furthermore, it provides generic interfaces to allow the interaction with various, legacy DC sub-systems, bridging the gap between these entities and the eCOP DB. When needed, the ICT Performance and Energy Supervisor will implement advanced algorithms to produce relevant metrics and translate raw data into meaning full metrics useful to apply specific sets of energy efficiency strategies. The role of the ICT Performance and Energy Supervisor is summarized in the following key points:

- It receives performance utilization data and energy consumption data collected by the DOLFIN DCO monitor/collector module.
- It provides information on the actual performance of the applications (typically VMs) utilizing the resources of the DC IT and non-IT equipment along with information on the actual energy consumption of this equipment.
- It analyses the utilization levels of each active DC physical server.

Figure 3-12 presents a high level description of the internal structure of the ICT Performance and Energy Supervisor module. This schema will be detailed in the next paragraphs in order to clarify the functionalities of each depicted element.

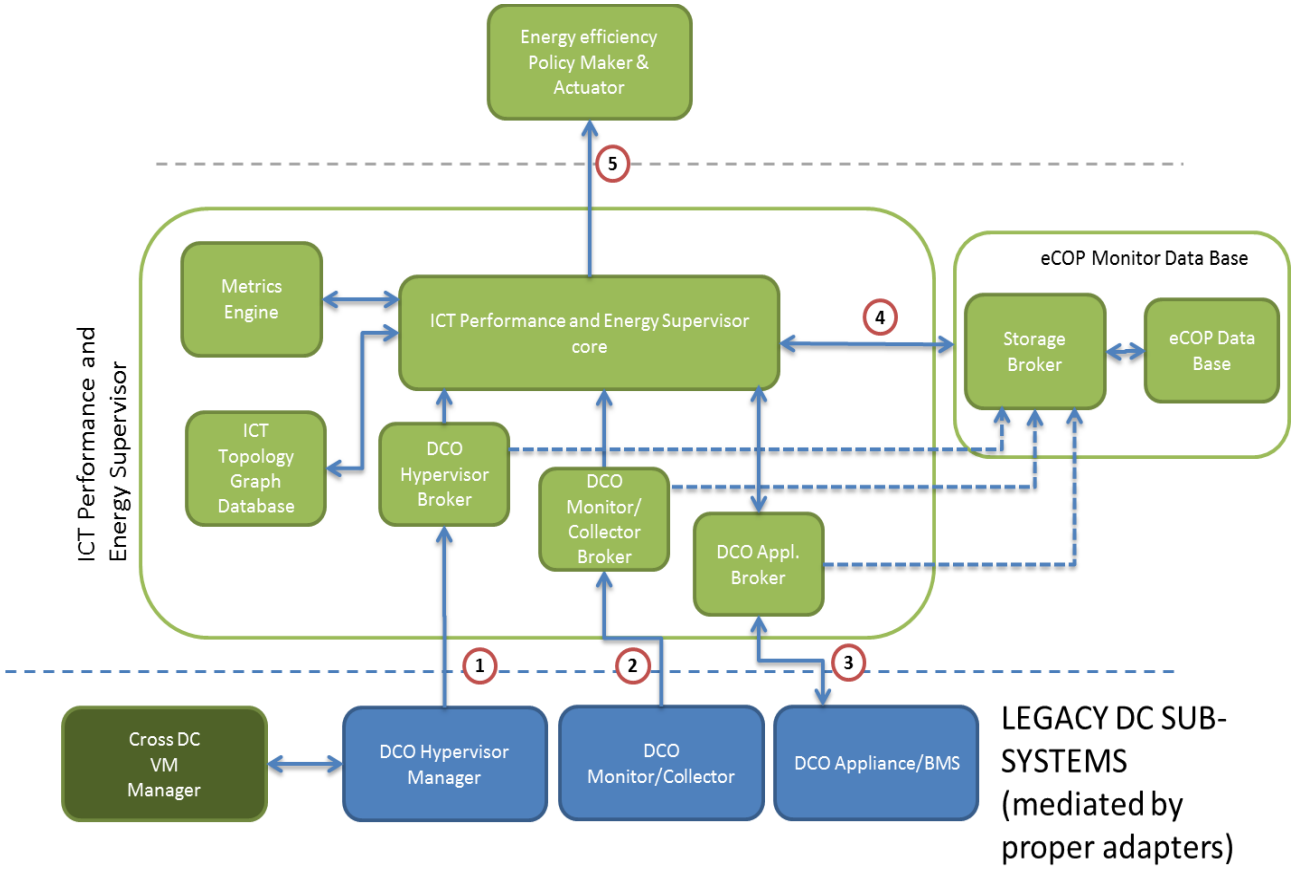

Figure 3-12: ICT Performance and Energy Supervisor module

3.3.1.1 Module detail description

The basic objective of the ICT Performance and Energy Supervisor is to provide a set of functionalities aiming at extracting the DC metrics and status evaluation.

It continuously receives updates about the status of each DC asset. This unorganized stream of data is provided by the DCO Monitor/Collector component via a dedicated communication interface. The ICT Performance and Energy Supervisor makes use of this incoming stream of data to continuously update its internal representation of the DC and, consequently, form an updated snapshot of the overall and detailed status of the DC assets.

Each inbound data event is correlated to its physical DC entity, which is present within the DC Topology DB. Once the relationship of an event with its physical host has been sorted out, each data update event is persisted into the eCOP DB component, whose purpose is to persist the status of the DC assets and metrics in the course of time.

The ICT Performance and Energy Supervisor makes use of the DC status to perform series of data aggregating procedures, leading to the generation of compound metrics, which are subject to optimization, and are detailed in deliverable D2.2 [1]. The optimization of these metrics, lying in the core of the DOLFIN project scope, is frequently recomputed in order to make sure that the DC operation is optimal from a resource utilisation perspective.

Keeping in mind the discussion above, the ICT Performance and Energy Supervisor is envisioned to comprise the following sub-modules:

- ICT Performance and Energy Supervisor core
- Metrics Engine
- DCO Hypervisor Broker
- DCO Monitor/Collector Broker
- DCO Appliance Broker
- ICT Topology Graph Database

3.3.1.2 Module decomposition

3.3.1.2.1 ICT Performance and Energy Supervisor core

This sub-component coordinates the communications between the other sub-modules and exposes multiple interfaces used by other higher-level DOLFIN components such as the Energy Policy Maker & Actuator.

This component is essentially a stateful data hub, continuously interfacing with a low number of other actors, and making low utilization of computational power. An asynchronous event-driven development framework will be used for the implementation of this component, using a reactive high level language.

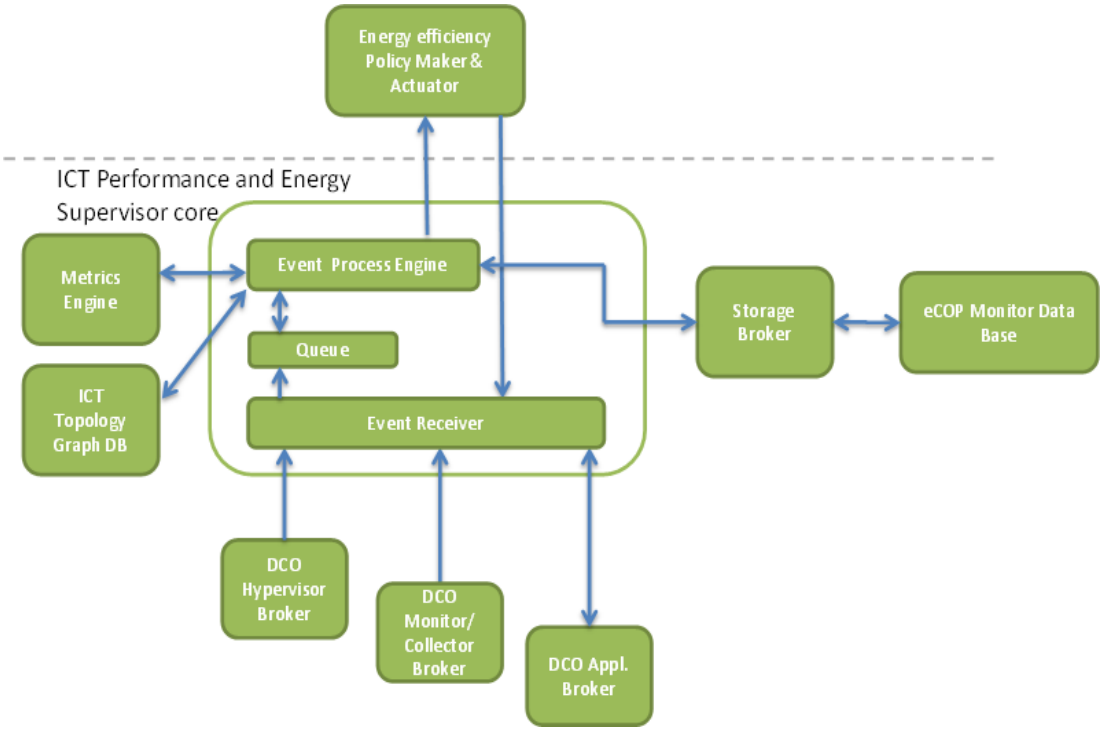

Figure 3-13: ICT Performance and Energy Supervisor internal structure

Figure 3-13 depicts the ICT Performance and Energy Supervisor internal structure. Three main modules are foreseen by the current design, those being the Event Receiver, the Queue and the Event Process Engine.

The Event Receiver detects events coming from external modules and stores the events into the Queue. These events are produced by other modules such as the DCO Hypervisor Broker, the DCO Monitor/Collector Broker, the DCO Appliance Broker or by the Energy efficiency Policy Maker & Actuator. Depending on its type, this sub-component assigns a priority level to each event, so that the events are organized in the queue based on their priority. This way, if a high priority event arrives to the Event Receiver, it will be pushed to the first position of the queue and it will be marked as the next event to be

processed. Moreover, the events will be characterized an associated timestamp. The timestamp provides information to the Metrics Engine about the time that the event has been generated.

The Queue stores the events in priority order, as explained in the previous paragraph. If two elements have the same priority, they are served according to their arriving order in the queue in a First In First Out mode.

The Event Process Engine classifies the event and executes the appropriate actions, such as metrics recalculation or ICT Topology Graph DB update, taking into consideration recent relevant data saved in the memory of the component, exposing basic caching capabilities; in case an external entity requests for metrics information, it can perform a quick search in its cache memory for a faster response. If the information is not found there, the module forwards the request to the eCOP Monitor DB.

3.3.1.2.2 Metrics Engine

This sub-component aggregates the eCOP Database data by using patterns obtained by the ICT Topology Graph Database and produces updated metric values which are then made available to the ICT Performance and Energy Supervisor core sub-component and, finally, to the other DOLFIN modules. This engine is not self-invoked; its function is triggered by the ICT Performance and Energy Supervisor core sub-component when the chosen criteria are met. It is therefore designed to run on-demand, exhibiting computationally-intensive behaviour, when active.

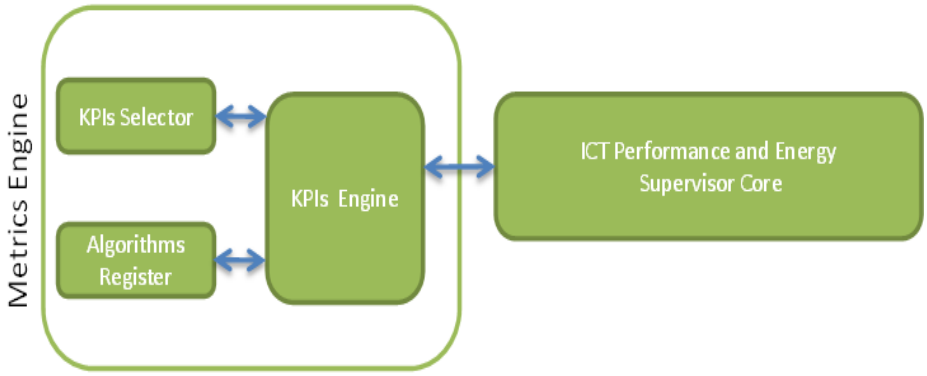

Figure 3-14: Metrics Engine

Figure 3-14 depicts the Metrics Engine. The KPIs Selector stores the different metrics the Metrics Engine can produce. When a new event triggers the Metrics Engine, it requests the KPIs to the KPIs Selector. The KPIs provided depend on several factors, for example the type of the event or the KPIs activated by the administrator. The Algorithms Register is a data base hosting the implementation details of the algorithms used for calculating the requested KPIs. The Algorithms Register together with the KPIs Selector makes the system more flexible, allowing for easy addition and update of KPIs and of the algorithms for calculating them.

The Metrics Engine deals with large sets of data; computes the metrics in one run, processing big amounts of data. Its architecture is fairly simple but tricky to scale. For this reason a development pattern suitable for intensive data computation of sparse data set will be used.

Next, a non-exhaustive list of metrics that are most likely to be included in the Metrics Engine implementation follows tabulated. The user interested in reviewing the full metrics list and the respective metric details is requested to review this list considering DOLFIN deliverable D2.2 [1] in conjunction with the document presenting the metrics produced by a cluster of EU projects related to DC energy efficiency, in which DOLFIN actively participates [15].

Moreover, the Metrics Engine is also responsible to produce the relevant energy metrics on the basis of the energy model defined in section 3 (Energy Models). For example the Algorithms Register will stores the algorithms used to calculate the consumption (i) per physical server and (ii) per VM, based on the actual distribution of VMs within the physical DC's infrastructure. A new execution of the Metrics Engine may be requested through an explicit solicitation by the ICT Performance and Energy Supervisor core when some changes in the DC internal structure is identified. In D2.2, the identification of the main Data Centre Energy Metrics and KPI relevant for the DOLFIN project is provided. The DC Cluster Collaboration identifies the existing metrics and KPIs related to DC operation, as well as provides new metrics. Taking into account the methodologies defined in the DC cluster for calculating the KPIs, the Metrics Engine will produce all these relevant metrics needed by the Energy Policy Maker & Actuator for building the Energy Models.

For instance, an event for calculating the new metrics is launched. The request is send to the Metrics Engine. The KPIs engine requests the active KPIs to the KPIs Selector and the KPIs engine loads them. For example, if the PUE (power Usage Effectiveness) is loaded, the KPIs engine will request to the Algorithm Register the algorithm stored for calculating the PUE. The algorithm will need some entries for calculating the PUE:

$$PUE = \frac{Total\ DC\ Power}{IT\ Equipment\ Power} = \frac{P_{cooling} + P_{lost} + P_{lighting} + P_{IT}}{P_{IT}}$$

where$P_{cooling}$ is the power used by the entire cooling system of the DC, P_{lost} is the power lost in the power distribution system through line-losses and other infrastructure (e.g. UPS, PDU) inefficiencies, $P_{lighting}$ is the power used to light the DC and P_{IT} is the power used by the IT equipment (server, network, storage) of the DC in hand. The Metrics Engine will request this information to the ICT Performance and Energy Supervisor Core that will re-send the request to the corresponding sub-module. Information from the IT and non-IT equipment will be needed. Once the monitoring information is gathered, the metrics engine will be able to calculate the particular KPI.

3.3.1.2.3 DCO data Brokers

The ICT Performance and Energy Supervisor comprises three sub-components specialized in handling raw data, producing a stream of data events suitable for persistence within the eCOP Monitor Database. Their primary objective is to form relationships between a data event and its corresponding record in the DC Topology Graph Database, representing the physical asset which originated the data event.

- The DCO Hypervisor Broker collects data coming from the DCO Hypervisor manager, pertaining also to the status of managed DC Hypervisors and hosted applications (VMs).
- The DCO Monitor / Collector Broker collects data coming from the DCO Monitor / Collector component. This stream of data contains updates about the status of hardware DC resources such as physical hosts, PDUs, ICT appliances, etc.
- The DCO Appliance Broker deals with data related to non-ICT resources that are part of the DC, such as HVAC systems, ancillary and non-correlated energy consumption, temperature, computer room conditions, etc.

The DCO data Brokers are basically event handlers. Similarly to the ICT Performance and Energy Supervisor core, an event-driven asynchronous development pattern, capable of aggressive data caching, is best suited for these components and will be adopted throughout the implementation phase.

To facilitate the communication to and from the DCO Broker, a proper data handling protocol needs to be adopted, allowing the data producers (publishers), the consumers (for example the ICT Performance and Energy Supervisor core) and the broker to be distributed on different physical locations/servers. The functional module is schematized as follow:

- The producer publishes the messages to an Exchanger entity that resides inside the broker.
- The Exchanger replicates the messages and it forwards them to message queues. Each queue provides two main functions: (1) data caching, (2) spike rate mitigation. Moreover, more message queues might be implemented to perform optimal load balancing.
- The consumer proceeds with the subscription to the broker to obtain the delivery of the message in the queue. The delivery model might support both an asynchronous notification and on demand, using a request/reply model.

The broker will support different delivery mechanisms which are mostly dependent on the criticality of the data that has to be transmitted. For example, raw data representing sensor measurements need to be delivered as soon as possible, possibly without any explicit acknowledgement. On the other hand, information reporting changes in the VM status should be delivered with emphasis on the reliability of the information and not on its timely delivery. To increase reliability, the broker should support acknowledgment mechanisms, counteracting the effect of networking problems or application fails. In this course, when a message is sent to the consumer, the consumer has to send an explicit notification to the broker, saying that the data has been correctly received and processed. After having received this acknowledgement, the broker will remove the data from the queue.

The workflow described above is schematized in Figure 3-15. This schema presents two typical applications of data exchanges between two specific DOLFIN modules, namely the DCO Hypervisor Broker (A) and the DCO Monitor/Collector Broker (B)). The diagram in (A) describes a mechanism that supports the message acknowledgment, while in (B) describes a simple and non-reliable model.

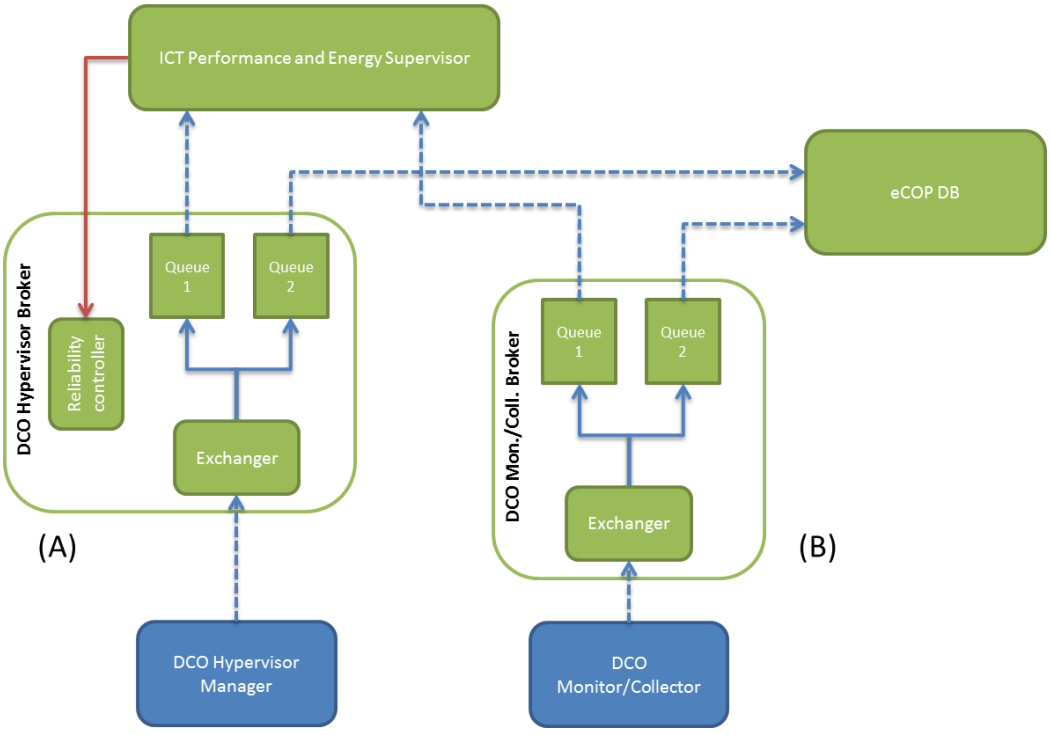

Figure 3-15: DCO Broker

3.3.1.2.4 ICT Topology Graph Database

The ICT Topology Graph Database contains an updated version of the data describing the physical distribution and topology of the DC assets. This database of intertwined data elements must reflect the actual allocation of the DC resources. In this framework, this database acts as a source of sensible data in

the context of metrics production, so that the ICT Performance and Energy Supervisor has local knowledge of the originating context of all data events.

The maintained data structure is mainly hierarchical, while still supporting cross-element references, which makes it capable of representing graph structures. In the context of the DOLFIN project, the topology held by the ICT Topology Graph Database will be structured in hierarchical tiers matching the actual physical and logical topology of the monitored DC. A list of the envisioned hierarchical tiers follows:

- ***Topology:*** The Topology element represents the root of the DC topology. Keeping in mind the complex structure of the DC, as a power consumer system, the topology root is commonly identifiable with the inlet stem of the electrical wiring. If the facility has several, independent power sources, one of these have to be selected to be mapped as the topology root node. Being a topology element itself, the topology root can be queried and used like any other. Measures associated or extracted in correspondence with this element represent global aggregated values which are for the DC as a whole

- ***Container:*** The Container element represents a nested partition of the DC topology, which can be either of physical or logical nature. Actual examples of partitions of a DC structure are floors, rooms, racks, etc. The partition mapping can then be extended to cover the deployment of the user assets, such as hypervisors and VMs. Dedicated partitions can then be envisioned for shared resources such as HVAC, network and shared storage appliance, PDUs, and other machinery which is necessary for the DC operations, but not explicitly related to a subset of it. In the case of a DC, the mapping of the topology in containers and sub-containers, de facto, forms a tree which, starting from the topmost level, closely matches the electrical wiring of the power backbone up to the detail level of the single server unit. Furthermore, the mapping will match the logical instantiation of the VMs to the hypervisor servers. For this reason, the measures associated or extracted in correspondence with a Container element, are to be interpreted as compound measures, and always relative to the subset of elements contained by the element.

- ***Device:*** the Device element represents a physical or logical device. This entity provides an abstraction of concrete element, such as a physical server, a VM, a PDU, etc. All these elements are intended as unit capable of being controlled, and inducing side effects over the elements under its influence. Structurally, this element is similar to the Container, but specializes its behaviour by determining the device type (which can be formalized via a vocabulary of terms, each representing a real DC asset). Devices form a relationship between a physical of software resource inside a DC, and can be moved (re-parented) freely inside the database, to map actual relocations of hardware and VMs. All kinds of metadata can be attached to Device elements, in order to make it available to the users of the topology database. The Device element can also host sub-elements, much like the Container element, but in case of unique, non-fractionable assets (such as VMs) it is typically used as pre-leaf element, containing only a single sub-level of Measure leaf elements.

- ***Measure***: The Measure element is always a leaf element, implying that it cannot be used to form derivations and partitions. This element will always correspond to an actual data source existing in some form in the DC. All physical quantities can be managed by this element; of special importance to the DOLFIN project are power absorption and current, voltage, electric noise, environmental and core temperatures, IOPS and network bandwidth. Purely numeric measures can also be managed, by declaring the unit element to be “number” (such is the case of datastore used percentage or network and loads).

A list of node records taking role in the instantiation of such topology records is explicitly presented in the Annex.

3.3.1.2.5 Work-flow

In this section is provided a workflow example following the generation of a new event. The interaction between the different modules is described following a change in the topology.

DCO Hypervisor Broker:

1. The DCO Hypervisor Manager publishes the message and posts to the Exchanger.
2. The Exchanger replicates the message and enquires it in two queues.
3. The ICT Performance & Energy Supervisor subscribes to the broker to obtain the message.

ICT Performance & Energy Supervisor Core:

4. Within the ICT Performance & Energy Supervisor is the "Core" who makes the subscription.
5. The Event Receiver sub-component detects the new event, assigns a priority and stores the event into the Queue.
6. The Queue serves the events with higher priority than before. It serves the "Change in the topology" event.
7. When the event arrives to the Event Process Engine, it identifies the event and launches the appropriate actions.
8. The event is published to the Energy efficiency Policy Maker & Actuator and to ICT Topology Graph Database.
9. Also, a warning is sent to the "Metrics engine" to inform about the change in the topology.

ICT Topology Graph Database:

10. The ICT Topology Graph Database updates the version of the data describing the physical and logical distribution and topology of the DC assets.

Metrics Engine:

11. The KPIs Engine requests the KPIs to the KPIs Selector. Depending on the configuration, it provides the KPIs for this type of event.
12. For each KPI, the Algorithms Register responds with the algorithm for calculating the KPI.
13. The KPIs Engine stores the KPIs and the Algorithms.
14. If needed, the Metrics Engine request the ICT Performance and Energy Supervisor core for the aggregated data produced and stored in the eCOP data base and for the patterns stored in the ICT Topology Graph Database.

ICT Performance & Energy Supervisor Core:

15. The ICT Performance & Energy Supervisor Core requests the needed information to the subcomponents and replies it to the Metrics Engine.

Metrics Engine:

16. The Metrics Engine produces updated metric values which are then made available to the ICT Performance and Energy Supervisor core sub-component.
17. The new metrics are saved in the memory of the "Core" and in the eCOP Data Base.

3.3.1.2.6 Data model description

The ICT Performance and Energy supervisor exposes the same data model with the storage Broker, documented in the Annex.

3.3.1.2.7 References to requirements

This section reports the list of requirements satisfied by this module. The complete list of identified requirements is described in the deliverable D2.2 [1].

Table 3-3: References to requirements (ICT Performance and Energy Supervisor)

Requirement ID & Name	Requirement description
Q7: MonitorParams	The monitoring subsystem shall always monitor DC parameters. This requirement necessitates the formal definition of all parameters that have to be monitored to quantify KPIs and evaluate the DC performance. This allows the implementation of the appropriate actions and policies. The list of parameters to be published should be configurable in terms of frequency of measurements and way to deliver. Also, the access to these functions should be controlled.
Q8: MonitorTranslate	The monitoring subsystem shall always translate parameters to metrics (KPI).
Q9: MonitorInterface	The monitoring subsystem shall be able to present metrics-information to the management subsystem. This interface should expose a number of metrics and their frequency in a normalized form. It could receive the metrics to retrieve their frequency and response format (e.g. publish to a message broker). The system shall monitor the DC infrastructure: IT devices, power units, other DC facilities that might be involved in the process of energy optimization.
Q10: MonitorVariousData	The monitoring subsystem shall handle multiple data of different nature (i.e. performance information, energy consumption, power dissipation, VM status and deployment, semi-static parameters, etc.). This function will complement in terms of Data the other peer Monitoring interfaces.
Q11: MonitorResilience	The monitoring subsystem should be as much resilient as possible (i.e. implement heartbeat, watchdog functions).
Q12: MonitorOpenDataFmt	The monitoring subsystem should utilize open data formats available.

3.3.2 Energy Efficiency Policy Maker & Actuator

The Energy Policy Maker & Actuator represents a central point in the DOLFIN architecture, where decisions for the application of energy policies are taken. The logical functions of this component are distributed over a sub-set of specialized entities that implement both the policy decision and enforcement.

Many actions applied by the Energy Policy Maker and Actuator require the interaction with external components, such as the Smart Grid Controller, the Cross-DC Workload Orchestrator and the SLA Renegotiation Controller. Dedicated APIs are implemented at the level of the internal entities of the Energy Policy Maker and Actuator to ensure correctness in data exchange and proper activation control between various components.

Elaborating on its role, the Energy Policy Maker and Actuator is set to

- Apply a set of well-known criteria and evaluation patterns to optimize the DC energy consumption (i.e. determining a set of operations to improve the DC energy efficiency).
- Produce a stream of requests which can be translated into actual actions by others DOLFIN servant subsystems within a DC.
- Determine corrective actions taking each action trade-offs and cost into account.

3.3.2.1.1 Module detail description

The Energy efficiency Policy Maker & Actuator consists of a number of sub-modules as depicted in Figure 3-16. These entities implement specialized functions, each of which has the responsibility of an optimization process part, for example introducing computational logics or providing the communication interfaces needed to interact with other components in the DOLFIN architecture.

As schematized in Figure 3-16, the Energy efficiency Policy Maker & Actuator encompasses a central entity (the Policy Maker) implementing the functionalities for the coordination of the actions needed to setup a new optimized energy plan. This entity receives input from several other components (for example measurements and KPIs provided by the ICT Performance and Energy Supervisor components or energy prices from the Smart Gird Controller) and, based on proper criteria, requests or schedules the activation of other components (such as the Predictor Engine or the Optimizer).

Through the interaction with the Cross-DC Orchestrator, the Energy efficiency Policy Maker & Actuator participates in the process of cross-DC VM migration. In this direction, the Energy efficiency Policy Maker & Actuator decides whether load migration, to and from a federated DC, is eligible according to the energy strategy implemented at local DC level and the current DC status assessment.

Moreover, through its interface with the Smart Grid Controller, the Energy efficiency Policy Maker & Actuators stimulated to adapt the operation of the DC to the energy policy imposed by the greater Smart Grid environment.

In the following sub-sections a detailed description of each constituent element of the Energy efficiency Policy Maker & Actuators provided, the constituent elements being:

- VM Priority Classifier
- Prediction Engine
- Policy Repository
- Policy Maker
- Optimizer
- Policy Actuator

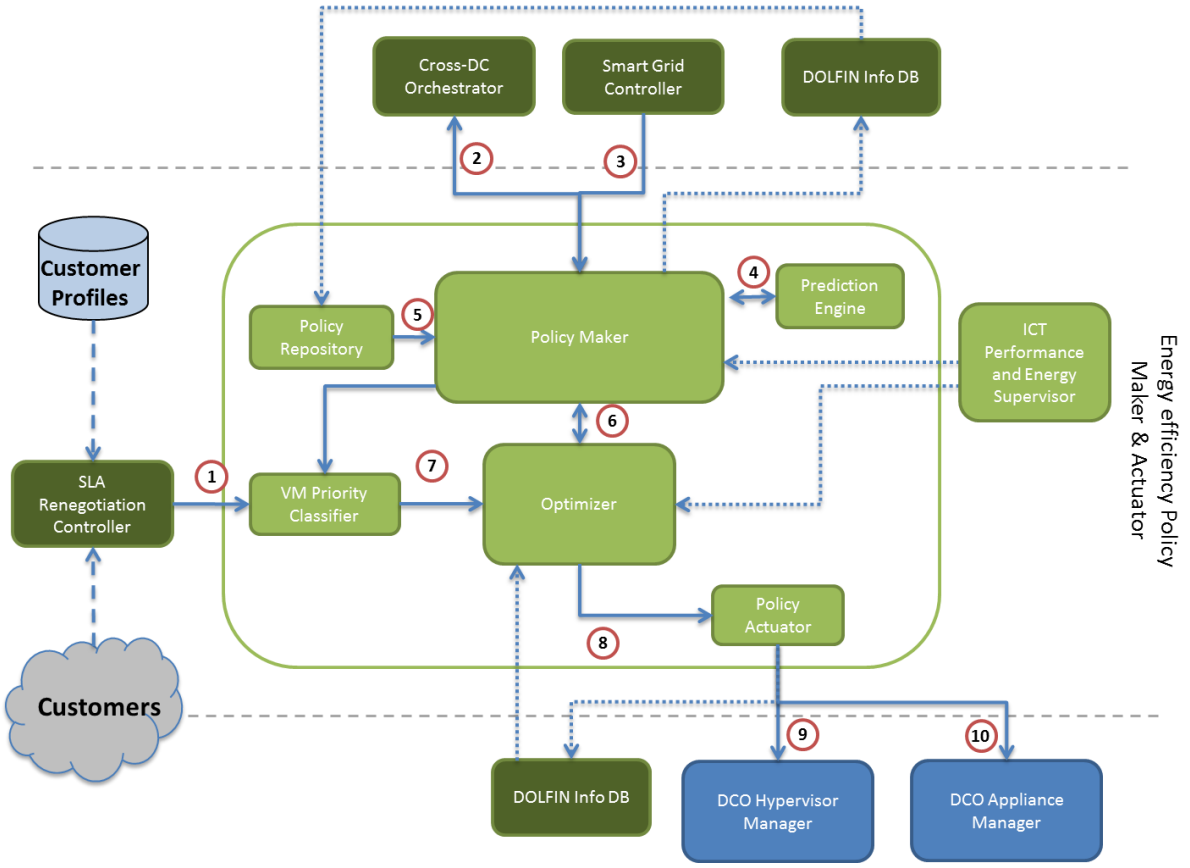

Figure 3-16: The Energy Policy Maker & Actuator.

***All dashed lines referring to DB interactions require their own interfaces with the corresponding components. These interactions are not numbered in the current architecture for clarity reasons.**

3.3.2.2 Module decomposition

3.3.2.2.1 VM Priority Classifier

The VM Priority Classifier is a component assessing the priority of VM streams based on their time criticality, their resource consumption and the SLA of the customer/entity (e.g. cooperating DC) managing the VMs in hand. In order to receive input on the SLA of the customers the VM Classifier consults the Customer Profile through the SLA Renegotiation Controller.

The VM Priority Classifier should also report specific information regarding the load optimization extent. More precisely, beyond the possibility to involve the VMs in the optimization process based on their priority, some VMs might be put (by the SLA Renegotiation Controller) in an *optimization block status*. This information will be useful for the Optimizer for two main purposes:

- first, restore the original state of these VMs to meet the original parameters of the SLAs;
- second, apply the optimization process so that these VMs are not involved in any performance reduction procedure.

On the other hand, the information regarding resource consumption of domestic load/VMs (intra-DC) is obtained by the ICT Performance and Energy Supervisor module, whereas information regarding the resource consumption of synergetic VMs is received along with the incoming VM requests. The VM Classifier is also responsible for translating the resource consumption of incoming VMs (e.g. CPU utilization)

to a baseline resource consumption (eventually using the functionality of metrics calculation provided by the ICT Performance and Energy Supervisor module), based on the differences between DC infrastructures (e.g. CPU processor).

3.3.2.2.2 Prediction Engine

The role of the Prediction Engine is to provide forecasts regarding the load expected in the near future, based on the status of both the DC as a standalone entity (assessed based on input from the ICT Performance and Energy Supervisor) and as a part of a federated set of DOLFIN-enabled DCs (assessed based on information from the DOLFIN Info DB). The predictions are sent to the Optimizer via the Policy Maker (I/F 4 of Figure 3-16). The Prediction Engine can also query the Policy Maker (I/F 4 of Figure 3-16), to revise the existing VM allocation of the DC based on the prediction of the domestic or synergetic load (e.g. VM consolidation prior to an estimated load increase).

The functionality of the prediction engine as part of the policy decision loop is summarized in the following:

- First, it provides info to the Optimizer about the near future load, while the Optimizer processes the current DC load status. For example, if the load is expected to decrease the Optimizer could proceed to a necessary massive load migration.

- Second, it hosts an internal clock, waking up itself in order to conduct a near future DC status prediction. This near forecast is pushed to the Policy Maker, possibly dictating the devise of a new optimization plan by the Optimizer. For example, if the load is expected to increase and the DC must be in the best shape to accommodate it.

3.3.2.2.3 Policy Repository

The Policy Repository maintains information regarding all possible policies available to conduct the operation efficiency resource management. This repository registers:

- The energy policy used internally by the eCOP functions. The Policy Maker queries the repository for the available policies in order to extract the best policy (that will be used to determine the optimization logics) to apply in every situation encountered. The selection is performed using sets of criteria based on inputs to the Policy Maker and its knowledge of the DC status. An energy policy is then “translated” in algorithms by the Optimizer Engine.

- The policy agreements for federated DCs. VM requests arriving at the Policy Maker through the Cross-DC Orchestrator, which provide basic functions of pre-processing of the request, such as filtering based on inter-DCs SLA agreement. Such a filtering would reject VM requests from non-federated DCs or even provide a higher priority to load from high priority DCs. This information, pertaining to the DC priority characterizes also the VM priority and is passed through the Policy Maker to the VM Priority Classifier if the VM request is accepted.

3.3.2.2.4 Policy Maker

The Policy Maker is the key component of the Energy Policy Maker and Actuator subsystem and is responsible for scheduling the activation of the policy enforcement, on the basis of the DC status and the information provided by other modules. This module is responsible for the efficient resource management and the acceptance or rejection of incoming requests at local or synergetic DCs level.

At a local, intra-DC level, it reacts to the changes being operated on the VMs by semi-continuously assessing the status of DC topology and metrics, and provide a coordination point to activate internal procedures based on policy criteria. For example, when a new VM is activated, the DOLFIN system will highlight the effects of the changes that have occurred in several ways (i.e. metrics, global consumption, VM topology, etc.). The Policy Maker will assess the new DC status and, based on these inputs and involving

in the evaluation path other internal module such as the Prediction Engine (or external, such as the Smart Grid Controller), it will trigger the activation of an optimization procedure via the Optimizer.

At a synergetic DC level, the Policy Maker interacts with the Cross-DC Orchestrator to (1) start and request a migration to a federated DC, (2) accept the migration request from other DCs (mediated by the Cross-DC Orchestrator). The choice, if made, is based on the availability of the DC resources and following the acceptance of such a request the optimizer is activated to optimize the allocation of the incoming load.

In order to decide if a new optimization procedure should be initiated (for example due to a change in some metrics or an incoming request from Cross-DC Orchestrator for a VM migration) a simplified process is defined as follows:

1. The Policy Maker performs the initial screening of incoming VM requests, based on the current DC status, topology and metrics. The screening is performed in order to decide whether the incoming VM request can be accommodated given the current DC status. Moreover, the Policy Maker accepts input from the Prediction Engine and the Smart Grid controller and based on this input the Policy Maker decides whether it is mandated to devise a new optimization plan.

2. The request then is forwarded to the Optimizer, which generates the actions required to achieve the estimated best configuration for the DC resources, in the context of the current policy activation. This component focuses on maximizing the exploitation of under-loaded resources and minimizing the number of resources which must stay active in consequence of the policy actuation.

Though phenomenally simple, these two steps imply complex activities. The Policy Maker must evaluate sets of information, which have different weights and priorities, and depends on the general strategies adopted. For example, starting from the results of business analysis performed in the context of the deliverable D2.1 [16], DOLFIN might operate with adopting general strategies with different priorities.

If in single DC environment:

- Maximize energy efficiency through workload redistribution within the DC.
- Target the optimization process to maximize the benefits/incentives from national/European authorities.
- Target the optimization process to support the energy needs in a Smart City environment through integration with Smart Grids.
- Performs SLAs renegotiation with end customers to reduce the energy consumption and providing costs reduction when uses green energy.

In a synergetic DC environment:

- Target the optimization process to support the energy needs in a Smart City environment through integration with Smart Grids.
- Maximize energy efficiency through workload redistribution with a federated DC's environment
- Redistribute the workload to federated DCs to optimize the exploitation of green resources.

Before any optimization action, the Policy Maker should implement a pre-analysis based on all possible inputs:

- Internal DC status and metrics

- Smart Grids energy/price requests
- Cross-DC MV migration requests
- SLA Renegotiation Controller events requests

These inputs, with the strategies priority and also other relevant information (such as the result of a prediction from Predictor Engine), will be used to:

1. Decide whether or not to proceed with some sort of optimization action (via the Optimizer)
2. Select the appropriate policy from the Policy Repository trough a “policy selector function”.

The result policy is then passed to the Optimizer implementing optimization algorithms that will optimize the energy consumption state of the DC in hand.

Based on the previous discussion, and granted the need to let the DC evolve its operation based on full knowledge of its external environment, an approach approximating the operation of autonomous systems will be adopted, granting the DC operation with the abilities of self-configuration, self-healing, self-optimization and self-protection [17]. To this end, a closed control loop following the principles of MAPE-K adaptation control loops [18] will be considered, implementing both monitoring, analysis, planning and executing components, as depicted in the Figure 3-17.

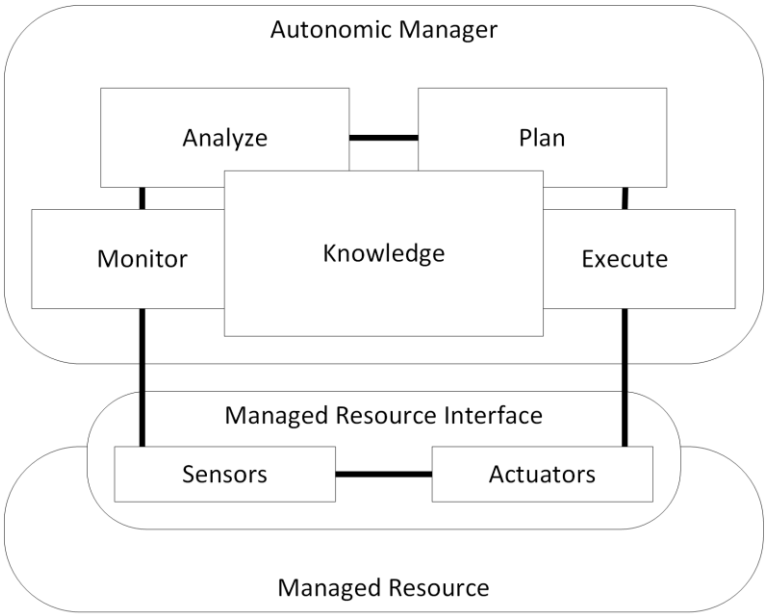

Figure 3-17: MAPE-K loop

In the DOLFIN eCOP case, the managed resource is masked by the ICT performance and Energy Supervisor which, with the help of the DCO Monitor/Collector and the eCOP Database as a whole is able to monitor the DC and provide valuable insight to the Policy Maker and Actuator module. The latter, constantly monitors the DC state (Prediction Engine) in combination with the ICT Performance and Energy Supervisor, analyses its state, plans what is the most effective way of optimizing the energy efficiency of the DC (Optimizer) and, finally, executes this plan (Policy Actuator).

Any relevant action performed by the Policy Maker will be logged on the DOLFIN Info DB.

3.3.2.2.5 Optimizer

The Optimizer is the most sophisticated component of the Energy efficiency Policy Maker & Actuator. The role of the Optimizer is to optimize the allocation of existing and accepted load to the DC physical resources, while adjusting to the operation imposed by the selected policy, also taking into account the forecast of the Prediction Engine on the near future load and/or other information such as the Smart Grid status by the Smart Grid Controller

To achieve this, the Optimizer communicates with the VM Priority Classifier (I/F 7 of Figure 3-16) and based on the current DC status (assessed with the input from the ICT Performance and Energy Supervisor), devises his plan for optimized DC operation. Subsequently, the Optimizer requests the final ACK to the Policy Maker and passes the devised set of actions to the Policy Actuator (I/F 8 of Figure 3-16) for implementation.

The Optimizer might work on the basis of heterogeneous inputs that are dependant by the policy will be applied and will be appropriately correlated by the Optimizer. A general flow is described below:

1. The Optimizer receives the control from the Policy Maker, which provides: input data sets (for example the analysis conducted by the Predictor Engine or energy prices or commands from the Smart Grid Controller) and an indication of the strategy to be applied (what energy policy leads the optimization process). Based on the energy strategy, the Optimizer selects the relevant algorithms needed to perform the optimization.

2. The actual VMs data are retrieved by the Optimizer through the VM Priority Classifier. The Optimizer expects that the VM Priority Classifier provides consolidated information on: (i) which VMs need to be managed, (ii) what is the priority of each VM and (iii) what are the constraints.

3. At the end of optimization process, the Optimizer provides a feedback to the Policy Maker with two main purposes: (1) asks the confirmation of applicability, and (2) reports results to the “manager” in the decision loop. After that pass a sequence of actions to the Actuator.

In order to communicate with the Policy Maker and the Actuator the Optimizer employs the following vocabulary:

Action	Description
VM_migrate	Migrate VM to Server
VM_shift	Postpone the instantiation of VM until time period
DVFS	Scale down the voltage and frequency of Server by given percentage
Serv _term	Enter Server into hibernation mode
Serv _oper	Enter Server into operation mode
Ancil_term	Enter ancillary equipment into hibernation mode
Ancil_oper	Enter ancillary equipment into operation mode
Air_cond_temper	Set air conditioning equipment to given temperature

Table 3-4: Optimizer-Policy Actuator Vocabulary

Action	Description
VM_relocate	Request the relocation of list of VMs to synergetic DC

Table 3-5: Optimizer-Policy Maker Vocabulary

3.3.2.2.6 Policy Actuator

The Policy Actuator is responsible for the implementation of the actions identified by the Optimizer. In this context, the Policy Actuator translates the optimizer plan into commands, addressed first to the DCO Hypervisor Manager (I/F 9 of Figure 3-16) and, subsequently, to the DCO Appliance Manager (I/F 10 of Figure 3-16). Moreover, the Policy Actuator informs the DOLFIN Info DB about the implementation of the Optimizers' actions and the alteration of the current DC status.

The Policy Actuator is organized as a dynamic set of purely sequential status machines, each taking care of a specific subtask. The policy plan fed to the Policy Actuator will be managed with a very simple workflow:

1. The Policy Plan, consisting merely of a sequence of operations to be carried out, is analysed in its entirety. A subset of nodes is selected as concurrent entry points, corresponding to the sections of the Policy Plan which can be carried out in parallel without causing mutual conflicts or hardware contentions which would invalidate established SLAs
2. One state machine is started for each entry point, with the task of executing the chain of commands leading to the full coverage of the corresponding section of the Policy Plan.
3. Each state machine begins issuing commands to the DCO Hypervisor Manager and/or the DCO Appliance Manager
4. The status of execution and correct completion of every issued command is tracked by the status machines. Any incongruence with the established plan of work will make the sequence abort and an error will be reported. The policy is flagged as failed.
5. Concurrently, a full log of operations and consequent status acknowledgments is fed to the DOLFIN Info DB, for the purpose of later retrieval and overall status reporting.

3.3.2.3 References to requirements

In this section will reports the list of requirements satisfied by this module. The complete list of identified requirements is described in the deliverable D2.2 [1].

Table 3-6: References to requirements (Energy Efficiency Policy Maker and Actuator)

Requirement ID & Name	Requirement description
Q50: Ask for PostponeExecutionPolicy	The Energy Eff. Policy Maker and Actuator shall be able to issue a request of the Workload and VM Manager to shift computing load (postpone it for later). The Optimizer produces a sequence of actions that will be activated by the Actuator
Q51: Ask for RenegotiatioPolicy	The Energy Eff. Policy Maker and Actuator shall be able to issue a request of the Renegotiation Policy. The Policy Maker will evaluate new policy requirements and will schedule a new optimization via the Optimizer, also with the support of Predictor Engine.

Q52: Ask for VMParamsAdjustment	In case a workload redistribution is necessitated, the Energy Eff. Policy Maker and Actuator should be able to ask from the Workload and VM Manager to adjust the VM characteristics of a certain set of VMs
Q37: DC2BInterface MgmntSubOptimalDetection	The management subsystem shall detect DC suboptimal states (i.e. high KPI) by utilizing current DC metrics and defined thresholds. The info concerning measures and KPIs are provided by the ICT Performance and Energy Supervisor. The Policy Maker with the supports of the Predictor Engine, evaluate if a new optimization (via the Optimizer) is needed.
Q54: DCHeatControl	In case the Smart Grid Controller retrieves a demand for a heat exchange state change from the DC part, the Energy Eff. Policy Maker and Actuator should be able to control the heat exchange towards the Smart City.
Q20: DCVMHandling	The policy enforcement subsystem shall be able to manipulate VMs (migrate them from server to server, from DC to DC, shutdown, etc.)
Q19: HVACPowerControl	The policy enforcement subsystem shall be able to manage non ICT devices power dissipation (i.e. control/shutdown HVAC equipment). As results of the optimization process, the Optimizer can requests control actions on the non-ICT facilities via the Actuator.
Q18: ICTPowerControl	The policy enforcement subsystem shall be able to manage ICT devices power dissipation (i.e. DVFS, ACPI, etc.). As results of the optimization process, the Optimizer can requests actions on the ICT facilities via the Actuator.
Q63: MgmntCostDetection	The system shall be able to enforce different power dissipation policies (i.e. using DVFS and/or ACPI functions, control HVAC equipment and also through VM migration). The policy enforcement subsystem will implement specialized interface to communicate with the management subsystem via the Actuator module.
Q5: MgmntPolicyDecision	The management subsystem shall automatically identify the actions that could be taken to optimize the DC energy state.

3.3.3 eCOP Monitor Database

The eCOP Monitor Database is the basic persistence utility of the ICT Performance and Energy Supervisor module. Its scope is twofold:

1. Persist the detailed DC monitoring information
2. Provide this data to the ICT Performance and Energy Supervisor Core (see 3.3.1, page 33 for more details) upon relevant request

From an architectural point of view, the eCOP Monitor Database consists of two active internal components; the eCOP Database and the Storage Broker, the former consisting the basic persistence layer of the eCOP Monitor Database and the latter wrapping the aforementioned persistence layer with a comprehensive RESTful API. The two components are detailed in the following paragraphs.

3.3.3.1 Module detail description

A high level depiction of the eCOP Monitor Database module architecture is given in Figure 3-18, where the bidirectional arrow in the left side of the figure indicates the interaction with other components including the ICT Performance and Energy Supervisor core, the DCO Hypervisor Manager, the DCO Monitor/Collector and the DCO Appliance Manager (see paragraph 3.3.2, page 41 for details).

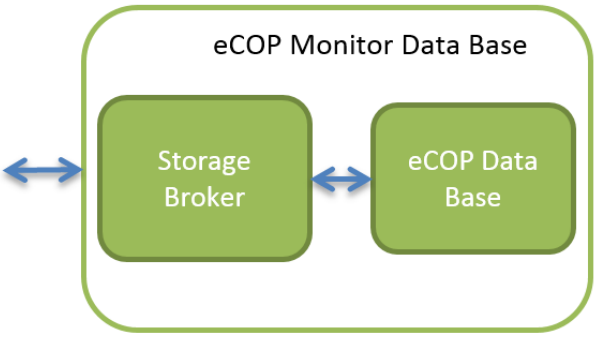

Figure 3-18: The eCOP Monitor Database module architecture

The Storage Broker exposes a RESTful API, wrapping the contents of the eCOP Database in order to:

1. facilitate the data persistence and retrieval processes for the various components interconnecting with the module under consideration,
2. restrict the access of the various components to the pieces of information authorized to interact with, in an attempt to establish fine-grained control on the various data flows to and from the eCOP Database.

3.3.3.2 Data model description

The Storage Broker exposes a simple, relatively flat data model which is based on the actual implementation of the eCOP Database persistence layer. The latter is expected to be loosely coupled, with minimal foreign key constraints in order to enhance read/write speed. Currently, a relational Database implementation has been chosen; other database architectures including no-SQL approaches will be also considered, however without affecting the data model exposed by the Storage Broker and documented in Annex 1 (paragraph A1.3).

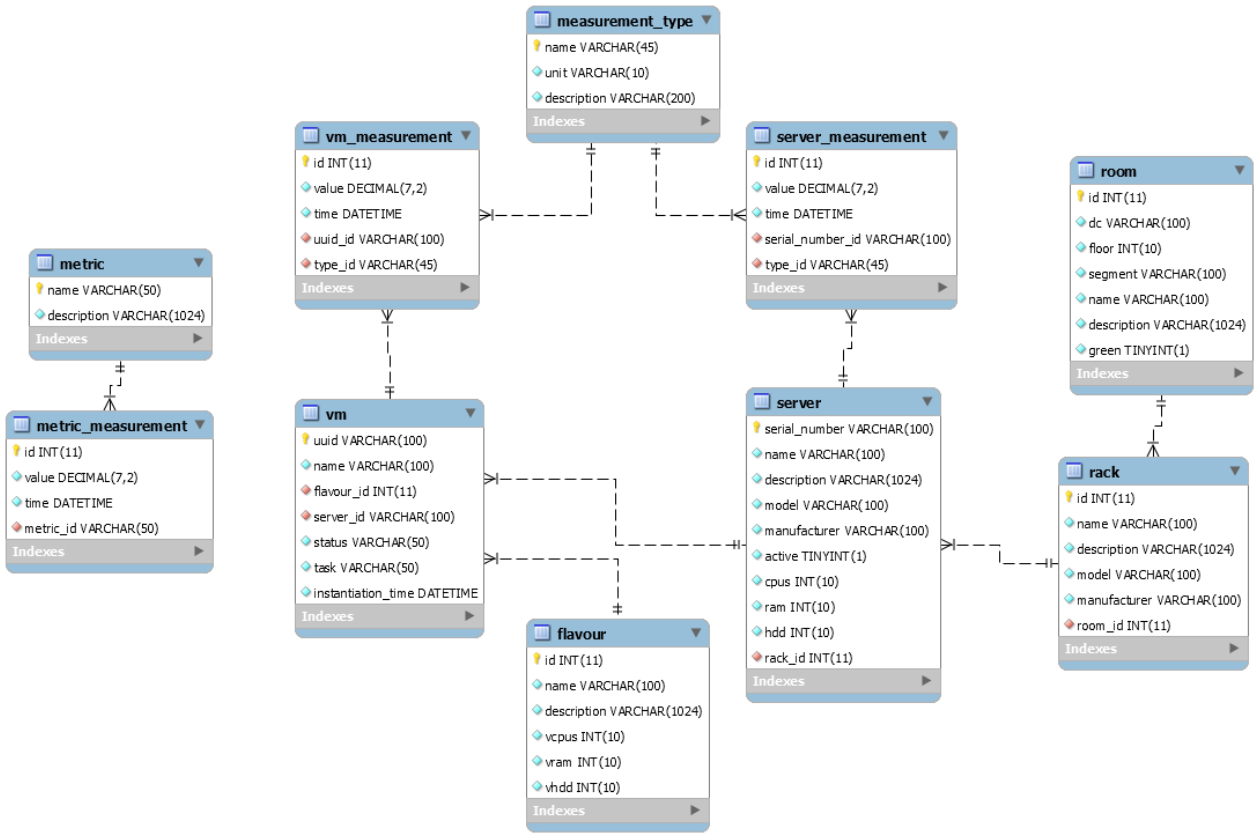

Figure 3-19: Depiction of the eCOP Database initial design

Please note that in Figure 3-19, the tables server, room and rack are only presented for reasons of clarity; normally, will tables should reside in the ICT Topology Graph Database (see paragraph 3.3.1, page 33 for details) and the foreign keys will not be present. Instead, plain references to the Serial Number of the servers and the IDs of the server racks and server rooms will be provided. This separation is also apparent in the detailed description of the Storage Broker API detailed in Annex 1 (paragraph A1.3).

3.3.3.3 Module decomposition

As already detailed, the eCOP Monitor Database simply consists of a database backend wrapped through a RESTful set of web services offering access to the former. This components architecture has been depicted in Figure 3-18. No further decomposition of the component is possible.

3.3.3.4 References to requirements

In this section we report the list of requirements satisfied by this module. The complete list of identified requirements is described in the deliverable D2.2 [1].

Table 3-7: References to requirements (eCOP Monitor Data base)

Requirement ID & Name	Requirement description
Q9: MonitorInterface	The monitoring subsystem shall be able to present metrics---information to the management subsystem. This interface is considered to expose in a normalized form a number of metrics and their frequency. It could receive the metrics to retrieve, their frequency and response format (e.g. publish to

	a message broker). The system shall monitor the DC infrastructure: IT devices, power units, other DC facilities that might be involved in the process of energy optimization.
Q10: MonitorVariousData	The monitoring subsystem shall handle multiple data and with different nature (i.e. performance information, energy consumption, power dissipation, VM status and deployment, semi-static parameters, etc.). This function will complement in terms of Data the other peer Monitoring interfaces.
Q11: MonitorResilience	The monitoring subsystem should be as much resilient as possible (i.e. implement heartbeat, watchdog functions).

3.3.4 Cooling Subsystem

Within the project scope, a "***liquid cooling subsystem***" prototype will be developed. The prototype will consist of a liquid circulation system that will bring the cooling liquid directly to the servers' hardware components (CPU, GPU, memory, etc.). Furthermore, this system will be interconnected to the legacy DC HVAC system. The gain from this interconnection will be twofold: first, the harvested thermal energy could be used for heating purposes and, second, the temperature of the cooling liquid will be lowered enough so that it can be reused for cooling again the various hardware components. The following figure depicts a rough diagram of the envisioned prototype.

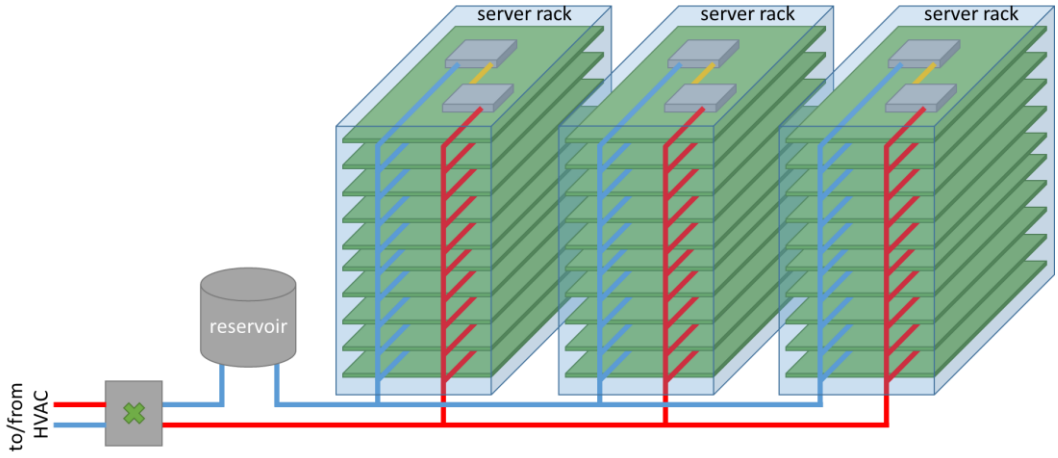

Figure 3-20: Liquid-cooling system diagram

From an initial evaluation, the main building blocks of such a liquid cooling system are:

- **water-blocks** that are mounted on the hardware components to be cooled
- **pumps** that circulate the liquid within the system
- a **reservoir** that holds the liquid of the system
- **tubing and fittings** for interconnecting the various components
- the **liquid** (actually water with some additives)
- ***a mechanism for interconnecting all the loops together***
- ***a mechanism to interconnect the liquid cooling system to the conventional HVAC system***

Currently, a plethora of direct liquid cooling systems are readily available for high-end personal computers. Due to the fact that liquid cooling for high end personal computers has become mainstream, there are several cost efficient and good quality off-the shelf components to be utilized for building isolated liquid cooling systems. Namely, there are several solutions concerning water-blocks for all major CPU sockets, numerous pumps with a wide range of characteristics, reservoirs and tubing and fittings with various degrees of security.

What is missing is:

- how to interconnect several isolated liquid-cooling loops (one per blade server) together so that they form a single liquid cooling system that share the same liquid
- and how to interconnect this liquid cooling system to a conventional HVAC system, so that thermal energy is exchanged efficiently while the liquids of the two systems do not mix.

Within the project, we will build a liquid cooling system that will utilize off-the-shelf components wherever applicable and also develop the parts that are missing. Concerning the commercially available components, it is critical that the various components are carefully selected and evaluated so that the developed system is extremely safe and also redundant wherever possible.

As stated above, the identified parts of the system that need to be developed is a mechanism to interconnect all the isolated liquid cooling loops together and a mechanism to interconnect the liquid cooling system to the DC HVAC system. Additionally, a high-volume central reservoir may also need to be developed. However, it is quite possible that during the development phase some other component may also need to be developed or modified accordingly.

Finally, a monitoring mechanism will be incorporated to the system that will enable real-time measurements of the various parameters of the system (i.e. liquid temperature, liquid level, liquid flow, etc.) and thus be able to detect and diagnose faulty conditions as soon as possible. Control mechanisms will also be incorporated (i.e. liquid flow control, additional cooling mechanisms of the loop liquid, etc.) so as to enable safe and adaptive operation under all conditions.

4 Conclusions

This document presents the detailed architectural design specification of the DOLFIN DC energy consumption optimization platform (eCOP), the latter forming the monitoring and optimization DOLFIN platform segment at local DC level. Following the architecture presented in the deliverable D2.2 [1], the updated architecture high-level view has remained identical, whereas the individual components composing eCOP have been detailed in terms of both architectural design and functionality. Regarding the functionality and internal design of each eCOP basic component, the following list summarizes D3.1 contributions:

- **ICT Performance and Energy Supervisor**: This component has the main responsibility to interact with underlying legacy DC subsystems and collect relevant information to evaluate metrics and KPIs. The module implements mechanisms to efficiently retrieve the data from various sub-systems and distribute such information to appropriate consumers. Moreover, the ICT Performance and Energy Supervisor implements specific techniques for data analysis and their representation, that will be useful from other eCOP module for their optimization activities. The functionality of this component has been divided into several smaller sub-systems, including the DCO Adapters (employed to accommodate different underlying infrastructures data models), the ICT Topology Graph Database (maintaining structured information related to the physical distribution and topology of the DC assets), the Metrics Engine (used to calculate KPIs evaluating the status of the DC at any time) and the ICT Performance and Energy Supervisor Core, undertaking the critical part of processing and distributing the incoming data streams to all interested parties (components).

- **Energy Efficiency Policy Maker and Actuator**: This component implements much of the intelligence needed for the application of the optimization policies proposed in DOLFIN. It is responsible for the application of the energy optimization procedures based on particular predefined criteria and conditioned by the inputs (requests) provided by other DOLFIN components, for example requests from the Smart Grids environment, from federated DCs, etc. The Energy Policy Maker and Actuator is structured in a group of specialized entities, each of which has the responsibility of a part of the optimization process. Specifically, the Policy Maker acts as the main decision-making component of the Energy Efficiency Policy Maker and Actuator processing requests and notifications for DC state changes both from eCOP and SDC components. The Prediction Engine is responsible for predicting and notifying the Policy Maker regarding near future trends in the DC load variations. The Optimizer generates optimization plans (e.g. optimal VM placement, etc.) for intentional DC state changes. The VM Priority Classifier is employed to provide VM prioritization services to the Optimizer so that SLA compliance is assured throughout the whole optimization procedures. Finally, the Policy Actuator implements the (optimized) plan.

- **eCOP Monitor Data base**: The eCOP Monitor Database implements the core eCOP persistence layer. It features a complete DB for storing information related to energy and (non) ICT equipment monitoring data, as well as information regarding the evaluation of the DC status by means of KPIs

(calculated via the Metrics engine of the ICT Performance and Energy Supervisor). Access to the data of the eCOP Monitor Database is facilitated by a wrapping RESTful API. Last, the eCOP Database also introduces specialized modules to efficiently interact with that data store and produce a time series aggregation.

- **Cooling Subsystem**: A major design decision that has been taken, to follow a hierarchical architectural approach. Specifically, local liquid cooling loops will exist per each server rack, these local loops will be interconnected to a ring-like liquid loop, which in its turn, will be interconnected to the existent HVAC system. This approach offers improved manageability and simplifies the installation and migration process of existing DCs. For implementing the local liquid loops, conventional, off-the-self components have been identified (i.e. pumps, water-blocks, reservoirs, etc.). For the various proprietary functions of the system (i.e. local-loop interconnection, HVAC interconnection, control and management), possible solutions and components that need to be developed have been identified.

Finally, a first attempt to consolidate the APIs that will enable the materialization of the eCOP functionality have been documented and is available under Annex 1.

5 References

[1] D2.2 - DOLFIN requirements and system architecture, [Online]. Available: http://www.dolfin-fp7.eu/wp-content/uploads/2014/10/DOLFIN_D2.2_UCL_30-09-2014_FF.pdf.

[2] "A Brief Introduction to REST," [Online]. Available: http://www.infoq.com/articles/rest-introduction.

[3] "Programmable Web - APIs," [Online]. Available: http://www.programmableweb.com/apis.

[4] Cesare Pautasso, "REST vs. SOAP: Making the Right Architectural Decision," [Online]. Available: http://www.jopera.org/files/soa-amsterdam-restws-pautasso-talk.pdf.

[5] Microsoft Corp., "A Guide to Designing and Building RESTful Web Services with WCF 3.5," [Online]. Available: http://msdn.microsoft.com/en-us/library/dd203052.aspx.

[6] "REST in peace, SOAP," [Online]. Available: http://royal.pingdom.com/2010/10/15/rest-in-peace-soap/.

[7] V. Rajadhyaksha, "Interview Question: Compare two web services type SOAP and RESTful (SOAP Vs RESTful)," [Online]. Available: http://technicalmumbojumbo.wordpress.com/2011/01/15/interview-question-soap-restful-webservices-comparison-soap-vs-restful/.

[8] P. Eugster, P. Felber, R. Guerraoui and A.-M. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Computing Surveys,* vol. 35, no. 2, pp. 114-131, 2003.

[9] "What is Google Cloud Pub/Sub?," Google Inc., [Online]. Available: https://cloud.google.com/pubsub/overview#benefits.

[10] D. Amadei, "Advanced Pub/Sub: Why Do We Need It?," Oracle, [Online]. Available: http://www.oracle.com/technetwork/articles/soa/amadei-advanced-pubsub-1836077.html.

[11] "RabbitMQ," [Online]. Available: https://www.rabbitmq.com.

[12] "MQTT," [Online]. Available: http://mqtt.org/.

[13] "A high-throughput distributed messaging system.," Apache Kafka, [Online]. Available: http://kafka.apache.org/.

[14] “Openstack Foundation,” [Online]. Available: https://www.openstack.org.

[15] Cluster of FP7 Projects Proposes New Environmental Efficiency Metrics for Data Centres, 10 09 2014. [Online]. Available: http://ec.europa.eu/digital-agenda/en/news/cluster-fp7-projects-proposes-new-environmental-efficiency-metrics-data-centres.

[16] D2.1 – Business scenarios and use case analysis, [Online]. Available: http://www.dolfin-fp7.eu/wp-content/uploads/2014/01/DOLFIN_D2.1_IRT_FF_20140207.pdf.

[17] O. J. Kephard and M. D. Chess, “The vision of autonomic computing,” *Computer,* vol. 36, no. 1, pp. 41-50, 2003.

[18] M. C. Huebscher and J. A. McCann, “A Survey of Autonomic Computing - Degrees, Models, and Applications,” *ACM Comput. Surv.,* vol. 40, no. 3, pp. 7:1-7:28, 2008.

[19] EUROPEAN COMMISSION DIRECTORATE GENERAL COMMUNICATIONS NETWORKS, CONTENT AND TECHNOLOGY UNIT E2 - SOFTWARE AND SERVICES , CLOUD , “Cloud Computing Service Level Agreements,” [Online]. Available: http://ec.europa.eu/information_society/newsroom/cf/dae/document.cfm?doc_id=249.

[20] C. S. Vladimir Stantchev, “Negotiating and Enforcing QoS and SLAs in Grid and Cloud Computing,” *Advances in Grid and Pervasive Computing,* vol. 5529, pp. 25-35, 2009.

6 Abbreviations

AP	Action Point
BMS	Building Management System
CEN	European Committee for Standardization
DB	Data Base
DC	Data Centre
DC	Data Centre
DCF	DC Federation
DCIM	Data Centre Infrastructure Management
DCO	Data Centre Optimization
DoW	Description of Work
EC	European Commission
EC	European Commission
EMS	Environmental Management System
EPs	Energy Providers
ESO	European Standards Organization (i.e. CEN, CENELEC and ETSI)
ETS	Emissions Trading System
ETSI	European Telecommunications Standards Institute
EU	European Union
FCoE	Fibre Channel over Ethernet
GUI	Graphical User Interface
ICT	Information and Communication Technology

IDB	Information Data Base
IPR	Intellectual Properties Rights
ISO	International Organization for Standardization
ITEU	IT Equipment Utilization
ITU-T	International Telecommunication Union - Telecommunication standardization sector
KPI	Environmental Key Performance Indicator
KPIEC	Energy Consumption
KPIREN	Renewable Energy Use
PM	Person Months
PMO	Project Management Office
QCT	Quality Check Team
SDC	Synergetic DCs
SG	Smart Grid
SGC	Smart Grid Controller
TMC	Technical Management Committee
ToC	Table of Content
UPS	Uninterruptible Power Supply
VM	Virtual Machines
WBD	Working Day Before Deadline
WP	Work package

Annex 1 – eCOP APIs description

A 1.1 ICT Topology Graph Database structure

In the context of the DOLFIN project, the topology held by the ICT Topology Graph Database will be structured in hierarchic tiers closely matching the actual physical and logical asset topology of the monitored DC. Here is a list of node records taking role in the instantiation of such topology record set:

	Topology
XML Element	`<topology id="…" label="MyDcName">…</topology>`
Description	The Topology element represents the origin of the DC topology. The complex structure of the DC, as a power consumer system, the topology root is commonly identifiable with the inlet stem of the electrical wiring. If the facility has several, independent power sources, one of these have to be elected to be mapped against the topology root node. Being a topology element itself, the topology root can be queried and used like any other. Measures associated or extracted in correspondence with this element represent global aggregated values which are for the DC as a whole.
Attributes	`Id` Unique uuid identifying the element across the DC federation `Label` Human-readable handle name for the element
Parent	`None (root node)`
Children	`Container, device, measure`

	Container
XML Element	`<container id="…" label="ContainerName">…</container>`
Description	The Container element represents a nested partition of the DC topology, which can be either of physical or logical nature. Actual examples of partitions of a DC structure are its internal organization in floors, rooms, racks. The partition mapping can then be extended to cover the deployment of the user assets, such as hypervisors and VMs. Dedicated partitions can then be envisioned for shared resources such as HVAC, network and shared storage appliance, PDUs, and other machinery which is necessary for the DC operations, but not explicitly related to a subset of it. In the case of a DC, the mapping of the topology in containers and sub-containers, de facto, forms a tree which, starting from the topmost level, closely matches the electrical wiring of the power backbone up to the detail level of the single server unit. Furthermore, the mapping will match the logical instantiation of the VMs on the hypervisor servers.

	For this reason, the measures associated or extracted in correspondence with a Container element, are to be interpreted as compound measures, and always relative to the subset of elements contained by the element.	
Attributes	**Id**	Unique uuid identifying the element across the DC federation
	label	Human-readable handle name for the element
Parent	**topology, container, device**	

	Device	
XML Element	**<device id="…" label="DevName" type="DevType">…</device>**	
Description	The Device element represents a physical or logical device, intended as unit capable of being controlled, and inducing side effects over the elements under its influence. Structurally, this element is similar to the Container, but specializes its behaviour by specifying the device type (which can be formalized via a vocabulary of terms, each representing a real DC asset). Devices form a relationship between a physical of software resource inside a DC, and can be moved (re-parented) freely inside the database, to map actual relocations of hardware and VMs. All kinds of metadata can be attached to Device elements, in order to make it available to visitors of the topology database, and this may be useful to also store in it credentials and reference data necessary to track the mapped resource across the network (such as MAC addresses, hypervisor instance IDs, IP addresses, etc). The Device element can also host sub-elements, much like the Container element, but in case of unique, non-fractionable assets (such as VMs) it is typically used as pre-leaf element, containing only a single sub-level of Measure leaf elements.	
Attributes	**Id**	Unique uuid identifying the element across the DC federation
	label	Human-readable handle name for the element
	type	Type of the represented element (server, PDU, VM, etc)
Parent	**topology, container, device**	

	Measure	
XML Element	**<measure id="…" label="MeasName" unit="MeasUnit">…</measure>**	
Description	The Measure element is always a leaf element, implying that it cannot be used to form derivations and partitions. This element will always correspond to an actual data source existing in some form in the DC. All sort of physical quantities can be managed by this element; of special importance to the DOLFIN project are power absorption and current, voltage, electric noise, environmental and core temperatures, IOPS and network bandwidth. Purely numeric measures can also be managed, by declaring the unit element to be “number” (such is the case of datastore used percentage or network and loads).	
Attributes	**Id**	Unique uuid identifying the element across the DC federation
	label	Human-readable handle name for the element
	unit	I.S. measure unit, representing scale and physical quantity of the stored measure
Parent	**topology, container, device**	

Hereafter provide the detail description of each API exposed and used by this module that will be provided by other modules in the DOLFIN architecture. With “external interface" refers to all the interfaces out from the module’s boundary.

A 1.2 REST API common status code

The following is a table of HTTP status codes that may be returned by the REST APIs described in in the rest of the document. If not otherwise specified, each of the following REST API definition can be expected to return one of the described codes.

After an HTTP request, the consumer code should make no assumptions and explicitly check the returned HTTP status code and react accordingly.

Reference: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

Status code	Status text	Description
200	OK	The request has succeeded. The meaning of a success varies depending on the HTTP method: • GET: The resource has been fetched and is transmitted in the message body. • HEAD: The entity headers are in the message body. • POST: The resource describing the result of the action is transmitted in the message body. • TRACE: The message body contains the request message as received by the server
201	Created	The request has succeeded and a new resource has been created as a result of it. This is typically the response sent after a PUT request.
202	Accepted	The request has been received but not yet acted upon. It is non-committal, meaning that there is no way in HTTP to later send an asynchronous response indicating the outcome of processing the request. It is intended for cases where another process or server handles the request, or for batch processing.
203	Non-Authoritative Information	This response code means returned meta-information set is not exact set as available from the origin server, but collected from a local or a third party copy. Except this condition, 200 OK response should be preferred instead of this response.
204	No Content	There is no content to send for this request, but the headers may be useful. The user-agent may update its cached headers for this resource with the new ones.
205	Reset Content	This response code is sent after accomplishing request to tell user agent reset document view which sent this request.
206	Partial Content	This response code is used because of range header sent by the client to separate download into multiple streams.
300	Multiple Choice	The request has more than one possible responses. User-agent or user should choose one of them. There is no standardized way to choose one of the responses.
301	Moved Permanently	This response code means that URI of requested resource has been changed. Probably, new URI would be given in the response.
302	Found	This response code means that URI of requested resource has been

		changed *temporarily*. New changes in the URI might be made in the future. Therefore, this same URI should be used by the client in future requests.
303	See Other	Server sent this response to directing client to get requested resource to another URI with a GET request.
304	Not Modified	This is used for caching purposes. It is telling to client that response has not been modified. So, client can continue to use same cached version of response.
305	Use Proxy	This means requested response must be accessed by a proxy. This response code is not largely supported because security reasons.
306	*unused*	This response code is no longer used, it is just reserved currently. It was used in a previous version of the HTTP 1.1 specification.
307	Temporary Redirect	Server sent this response to directing client to get requested resource to another URI with same method that used prior request. This has the same semantic than the `302 Found` HTTP response code, with the exception that the user agent *must not* change the HTTP method used: if a `POST` was used in the first request, a `POST` must be used in the second request.
308	Permanent Redirect	This means that the resource is now permanently located at another URI, specified by the `Location:` HTTP Response header. This has the same semantics as the `301 Moved Permanently` HTTP response code, with the exception that the user agent *must not* change the HTTP method used: if a `POST` was used in the first request, a `POST` must be used in the second request. **Note:** This is an experimental response code whose specification is currently in draft form.
400	Bad Request	This response means that server could not understand the request due to invalid syntax.
401	Unauthorized	Authentication is needed to get requested response. This is similar to 403, but in this case, authentication is possible.
402	Payment Required	This response code is reserved for future use. Initial aim for creating this code was using it for digital payment systems however this is not used currently.
403	Forbidden	Client does not have access rights to the content so server is rejecting to give proper response.
404	Not Found	Server cannot find requested resource. This response code probably is most famous one due to its frequency to occur in web.
405	Method Not Allowed	The request method is known by the server but has been disabled and cannot be used. The two mandatory methods, `GET` and `HEAD`, must never be disabled and should not return this error code.
406	Not Acceptable	This response is sent when the web server, after performing server-driven content negotiation, doesn't find any content following the criteria given by the user agent.
407	Proxy Authentication Required	This is similar to 401 but authentication is needed to be done by a proxy.
408	Request Timeout	This response is sent on an idle connection by some servers, even without any previous request by the client. It means that the server would like to shut down this unused connection. This response is used much more since some browsers, like Chrome or IE9, use HTTP preconnection mechanisms to speed up surfing (see bug 634278,

		which tracks the future implementation of such a mechanism in Firefox). Also note that some servers merely shut down the connection without sending this message.
409	Conflict	This response would be sent when a request conflict with current state of server.
410	Gone	This response would be sent when requested content has been deleted from server.
411	Length Required	Server rejected the request because the `Content-Length` header field is not defined and the server requires it.
412	Precondition Failed	The client has indicated preconditions in its headers which the server does not meet.
413	Request Entity Too Large	Request entity is larger than limits defined by server; the server might close the connection or return a `Retry-After` header field.
414	Request-URI Too Long	The URI requested by the client is too long for the server to handle.
415	Unsupported Media Type	The media format of the requested data is not supported by the server, so the server is rejecting the request.
416	Requested Range Not Satisfiable	The range specified by the `Range` header field in the request can't be fulfilled; it's possible that the range is outside the size of the target URI's data.
417	Expectation Failed	This response code means the expectation indicated by the `Expect` request header field can't be met by the server.
500	Internal Server Error	The server has encountered a situation it doesn't know how to handle.
501	Not Implemented	The request method is not supported by the server and cannot be handled. The only methods that servers are required to support (and therefore that must not return this code) are `GET` and `HEAD`.
502	Bad Gateway	This error response means that the server, while working as a gateway to get a response needed to handle the request, got an invalid response.
503	Service Unavailable	The server is not ready to handle the request. Common causes are a server that is down for maintenance or that is overloaded. Note that together with this response, a user-friendly page explaining the problem should be sent. This responses should be used for temporary conditions and the `Retry-After:` HTTP header should, if possible, contain the estimated time before the recovery of the service. The webmaster must also take care about the caching-related headers that are sent along with this response, as these temporary condition responses should usually not be cached.
504	Gateway Timeout	This error response is given when the server is acting as a gateway and cannot get a response in time.
505	HTTP Version Not Supported	The HTTP version used in the request is not supported by the server.

A 1.3 ICT Performance and Energy Supervisor

DCO Hypervisor Broker injection API				
Description	Used by DCO Hypervisor Broker to wait event updates from the DCO Hypervisor Manager			
Endpoint Name	`dco_hyperv_exchanger`			
RPC Call name(s)	`dco_hyperv_wait_events`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	originId	N	String	Specifies the origin context of the data
	payload	N	JSON	Completely describes the occurred status change
Provides response	NO			
Response Parameters	none			
-Requester	DCO Hypervisor Manager			
Managed errors	312 - No Route: The message is being sent to a destination that does not exist. 313 - No Consumers: The message is marked as immediate delivery, but no consumers are able to receive the message at this time. 403 - Access Refused: Implies that you've been refused access. 404 - Not Found: The client attempted an operation on an entity which does not exist. 405 - Already Exists: The client attempted to create an entity which already exists. 406 - In Use: The client attempted to delete an entity which is currently being used. 407 - Invalid Routing Key: The client attempted to use an invalid routing key. 408 - Request Timeout: The requested operation could not be completed in time. 409 - Invalid Argument: The client provided an argument which the server did not recognise, e.g. invalid JMS selector. 530 - Not Allowed: The client attempted an operation which it does not have permission for. 542 - Unsupported Protocol Version: The server does not support the requested AMQP version.			

DCO Hypervisor Broker archival API				
Description	Used by DCO Hypervisor Broker to post event updates toward the Storage Broker			
Endpoint Name	`storage_broker`			
RPC Call name(s)	`dco_hyperv_update`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	originId	N	String	Specifies the origin context of the data
	payload	N	JSON	Completely describes the occurred status change
Provides response	NO			
Response Parameters	none			
Requester	DCO Hypervisor Broker			

Managed errors	312 - No Route 313 - No Consumers 403 - Access Refused 404 - Not Found 405 - Already Exists 406 - In Use 407 - Invalid Routing Key 408 - Request Timeout 409 - Invalid Argument 530 - Not Allowed 542 - Unsupported Protocol Version

				DCO Monitor Broker injection API
Description	Used by DCO Monitor Broker to update events from the DCO Monitor Manager			
Endpoint Name	`dco_monitor_exchanger`			
RPC Call name(s)	`dco_monitor_wait_events`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	originId	N	String	Specifies the origin context of the data
	payload	N	JSON	Completely describes the occurred status change
Provides response	NO			
Response Parameters	none			
Requester	DCO Monitor Manager			
Managed errors	312 - No Route 313 - No Consumers 403 - Access Refused 404 - Not Found 405 - Already Exists 406 - In Use 407 - Invalid Routing Key 408 - Request Timeout 409 - Invalid Argument 530 - Not Allowed 542 - Unsupported Protocol Version			

DCO Monitor Broker archival API				
Description	Used by DCO Monitor Broker to post event updates toward the Storage Broker			
Endpoint Name	`storage_broker`			
RPC Call name(s)	`dco_monitor_update`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	originId	N	String	Specifies the origin context of the data
	payload	N	JSON	Completely describes the occurred status change
Provides response	NO			
Response Parameters	none			
Requester	DCO Monitor Broker			
Managed errors	312 - No Route 313 - No Consumers 403 - Access Refused 404 - Not Found 405 - Already Exists 406 - In Use 407 - Invalid Routing Key 408 - Request Timeout 409 - Invalid Argument 530 - Not Allowed 542 - Unsupported Protocol Version			

DCO Appl. Broker injection API				
Description	Used by DCO Appl. Broker to update events from the DCO Appl. Manager			
Endpoint Name	`dco_appl_exchanger`			
RPC Call name(s)	`dco_appl_wait_events`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	originId	N	String	Specifies the origin context of the data
	payload	N	JSON	Completely describes the occurred status change
Provides response	NO			
Response Parameters	none			
Requester	DCO Appl. Manager			
Managed errors	312 - No Route 313 - No Consumers 403 - Access Refused 404 - Not Found 405 - Already Exists 406 - In Use 407 - Invalid Routing Key 408 - Request Timeout 409 - Invalid Argument 530 - Not Allowed 542 - Unsupported Protocol Version			

DCO Appl. Broker archival API				
Description	Used by DCO Appl. Broker to post event updates toward the Storage Broker			
Endpoint Name	`storage_broker`			
RPC Call name(s)	`dco_appl_update`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	originId	N	String	Specifies the origin context of the data
	payload	N	JSON	Completely describes the occurred status change
Provides response	NO			
Response Parameters	none			
Requester	DCO Appl. Broker			
Managed errors	312 - No Route 313 - No Consumers 403 - Access Refused 404 - Not Found 405 - Already Exists 406 - In Use 407 - Invalid Routing Key 408 - Request Timeout 409 - Invalid Argument 530 - Not Allowed 542 - Unsupported Protocol Version			

Measurement Types				
Description	This service allows for requesting measurement types			
Endpoint Name	`/api/measurement_types/`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	-	-	-	-
Request body type				
Response class type	[GET] List<Measurement_type>			
Used by	Energy efficiency Policy Maker & Actuator			
Example	**Request:** GET /api/measurement_types/ **Response (application/json):** [{ "name": "cpu_util", "unit": "%", "description": "The CPU utilization of the entity" }, { "name": "memory", "unit": "MB", "description": "The total RAM available to the entity"			

```
  },
  {
    "name": "memory.usage",
    "unit": "MB",
    "description": "The current memory utilization of the entity"
},
  {
    "name": "hdd",
    "unit": "GB",
    "description": "The total HDD available to the entity"
  },
  {
    "name": "hdd.usage",
    "unit": "GB",
    "description": "The current HDD utilization of the entity"
}
]
```


	Measurement Types			
Description	This service allows for specific measurement types, based on their name			
Endpoint Name	`/api/measurement_types/{name}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	name	N	String	The name of the measurement type to handle
Request body type	-			
Response class type	[GET] Measurement_type			
Used by	Energy efficiency Policy Maker & Actuator			

Example	**Request:**`GET /api/measurement_types/cpu_util`
	Response (application/json):
	`{`
	`"name": "cpu_util",`
	`"unit": "%",`
	`"description": "The CPU utilization of the entity"`
	`}`			

	VM			
Description	This service allows for retrieving existing information related to the VM instances of the DC			
Endpoint Name	`/api/vms/`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	-	-	-	-

Request body type	-
Response class type	[GET] List<VM>
Used by	Energy efficiency Policy Maker & Actuator

Example	**Request:**GET /api/vms/
	Response (application/json):
	[
	{
	"name": "VM Instance name",
	"uuid": "uuid-12234567",
	"flavour": 4,
	"server": "SN12347",
	"status": "ACTIVE",
	"task": "None",
	"instantiation_time": "2015-02-17T20:00:52.239000Z"
	},
	{
	"name": "Another VM",
	"uuid": "uuid-12234568",
	"flavour": 4,
	"server": "SN12347",
	"status": "ACTIVE",
	"task": "None",
	"instantiation_time": "2015-02-17T20:00:52.239000Z"
	},
	{
	"name": "test-vm-entry",
	"uuid": "1234",
	"flavour": 4,
	"server": "SN12346",
	"status": "ACTIVE",
	"task": "None",
	"instantiation_time": "2015-02-17T20:00:52.239000Z"
	}
	]

				VM
Description	This service allows for retrieving information related to the VM instances of the DC, based on their uuid			
Endpoint Name	/api/vms/{uuid}			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	uuid	N	String	The UUID of the VM to handle
Request body type	-			
Response class type	VM			
Used by	ICT Performance and Energy Supervisor Core			

Example	**Request:** GET /api/vms/uuid-12234567
	
	Response (application/json):
	{
	"name": "VM Instance name",
	"uuid": "uuid-12234567",
	"flavour": 4,
	"server": "SN12347",
	"status": "ACTIVE",
	"task": "None",
	"instantiation_time": "2015-02-17T20:00:52.239000Z"
	}

				VM Measurement
Description	This service allows for retrieving existing Measurements of VM instances of the DC			
Endpoint Name	/api/vm_measurements/			
Allowed methods	GET, POST			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	-	-	-	-
Request body type	-			
Response class type	List<VM_ measurement>			
Used by	Energy efficiency Policy Maker & Actuator			

Example	**Request:**GET /api/vm_measurements/
	Response (application/json):
	[
	{
	"id": 2,
	"value": "2.00",
	"time": "2015-02-17T06:02:51.445000Z",
	"uuid": "uuid-12234567",
	"type": "cpu_util"
	}
	{
	"id": 6,
	"value": "15.00",
	"time": "2015-02-17T10:59:00Z",
	"uuid": "uuid-12234568",
	"type": "cpu_util"
	}
	]			

VM Measurement				
Description	This service allows for retrieving measurements of the VM instances of the DC, based on their id			
Endpoint Name	`/api/vm_measurements/{id}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	id	N	Integer	The ID of the VM measurement to handle
Request body type	-			
Response class type	VM_ measurement			
Used by	ICT Performance and Energy Supervisor Core			

Example	**Request:**GET `/api/vm_measurements/6`
	Response (application/json):
	{
	"id": 6,
	"value": "15.00",
	"time": "2015-02-17T10:59:00Z",
	"uuid": "uuid-12234568",
	"type": "cpu_util"
	}			

VM Measurement				
Description	This service allows for retrieving VM measurements that fall under a specific time range			
Endpoint Name	`/api/vm_measurements/by-date/{start}/{end}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	start	N	String	The start time of the time range, in ISO8601 format
	end	N	String	The end time of the time range, in ISO8601 format
Request body type	-			
Response class type	[GET] List<VM_ measurement>			
Used by	Energy efficiency Policy Maker & Actuator			
Example	**Request:**GET `/api/vm_measurements/by-date/2015-02-17/2015-02-18/` **Response (application/json):** [{ "id": 2, "value": "2.00", "time": "2015-02-17T06:02:51.445000Z", "uuid": "uuid-12234567", "type": "cpu_util"			

```
        },
        {
            "id": 3,
            "value": "2.00",
            "time": "2015-02-17T08:02:51.445000Z",
            "uuid": "uuid-12234567",
            "type": "cpu_util"
        },
        {
            "id": 6,
            "value": "15.00",
            "time": "2015-02-17T10:59:00Z",
            "uuid": "uuid-12234568",
            "type": "cpu_util"
        },
        {
            "id": 5,
            "value": "512.00",
            "time": "2015-02-17T11:58:00Z",
            "uuid": "uuid-12234567",
            "type": "memory"
        },
        {
            "id": 1,
            "value": "2.00",
            "time": "2015-02-17T10:01:51.445000+02:00",
            "uuid": "uuid-123456",
    "type": "1"
        }
    ]
```

<table>
<tr><th colspan="5">VM Measurement</th></tr>
<tr><td>Description</td><td colspan="4">This service allows for retrieving the VM measurements of a specific VM, based on their UUID</td></tr>
<tr><td>Endpoint Name</td><td colspan="4">/api/vm_measurements/by-uuid/{uuid}</td></tr>
<tr><td>Allowed methods</td><td colspan="4">GET</td></tr>
<tr><td>Request Parameters</td><td>Parameter</td><td>Optional (Y/N)</td><td>Type</td><td>Description</td></tr>
<tr><td></td><td>uuid</td><td>N</td><td>String</td><td>The UUID of the VM to fetch information for</td></tr>
<tr><td>Request body type</td><td colspan="4">-</td></tr>
<tr><td>Response class type</td><td colspan="4">[GET] List<VM_ measurement></td></tr>
<tr><td>Used by</td><td colspan="4">Energy efficiency Policy Maker & Actuator</td></tr>
<tr><td>Example</td><td colspan="4">Request:GET /api/vm_measurements/by-uuid/uuid-12234568/
Response (application/json):
[
{
"id": 6,
"value": "15.00",
"time": "2015-02-17T10:59:00Z",
"uuid": "uuid-12234568",
"type": "cpu_util"
}
]</td></tr>
</table>

<table>
<tr><th colspan="5">VM Measurement</th></tr>
<tr><td>Description</td><td colspan="4">This service allows for retrieving VM measurements of a specific VM that fall under a specific time range</td></tr>
<tr><td>Endpoint Name</td><td colspan="4">/api/vm_measurements/by-uuid/{uuid}/{start}/{end}</td></tr>
<tr><td>Allowed methods</td><td colspan="4">GET</td></tr>
<tr><td>Request Parameters</td><td>Parameter</td><td>Optional (Y/N)</td><td>Type</td><td>Description</td></tr>
<tr><td></td><td>uuid</td><td>N</td><td>String</td><td>The UUID of the VM to fetch information for</td></tr>
<tr><td></td><td>start</td><td>N</td><td>String</td><td>The start time of the time range, in ISO8601 format</td></tr>
<tr><td></td><td>end</td><td>N</td><td>String</td><td>The end time of the time range, in ISO8601 format</td></tr>
<tr><td>Request body type</td><td colspan="4">-</td></tr>
<tr><td>Response class type</td><td colspan="4">[GET] List<VM_ measurement></td></tr>
<tr><td>Used by</td><td colspan="4">Energy efficiency Policy Maker & Actuator</td></tr>
<tr><td>Example</td><td colspan="4">Request:GET /api/vm_measurements/by-uuid/uuid-12234567/2015-02-17T06:02:51.445000Z/2015-02-17T11:58:00Z/
Response (application/json):
[
{
"id": 2,
"value": "2.00",
"time": "2015-02-17T06:02:51.445000Z",
"uuid": "uuid-12234567",
"type": "cpu_util"
},
{
"id": 3,
"value": "2.00",
"time": "2015-02-17T08:02:51.445000Z",
"uuid": "uuid-12234567",
"type": "cpu_util"
},
{
"id": 5,
"value": "512.00",
"time": "2015-02-17T11:58:00Z",
"uuid": "uuid-12234567",
"type": "memory"
}
]</td></tr>
</table>

	Server Measurement
Description	This service allows for retrieving existing measurements of servers of the DC
Endpoint Name	`/api/server_measurements/`
Allowed methods	GET
Request Parameters	**Parameter** / **Optional (Y/N)** / **Type** / **Description** - / - / - / -
Request body type	-
Response class type	[GET] List<Server_measurement>
Used by	Energy efficiency Policy Maker & Actuator
Example	**Request:**`GET /api/server_measurements/` **Response (application/json):** `[ { "id": 1, "value": "16384.00", "time": "2015-02-16T23:00:00Z", "type": "memory.usage", "serial_number": "SN12346" }, { "id": 2, "value": "12125.00", "time": "2015-02-17T12:01:00Z", "type": "memory.usage", "serial_number": "SN12347" } ]`

	Server Measurement
Description	This service allows for retrieving measurements of the servers of the DC, based on their id
Endpoint Name	`/api/server_measurements/{id}`
Allowed methods	GET
Request Parameters	**Parameter** / **Optional (Y/N)** / **Type** / **Description** id / N / Integer / The ID of the Server measurement to handle
Request body type	-
Response class type	[GET] Server_ measurement
Used by	Energy efficiency Policy Maker & Actuator
Example	**Request:**`GET /api/server_measurements/2` **Response (application/json):** `{ "id": 2, "value": "12125.00", "time": "2015-02-17T12:01:00Z", "type": "memory.usage", "serial_number": "SN12347" }`

Server Measurement				
Description	This service allows for retrieving server measurements that fall under a specific time range			
Endpoint Name	`/api/server_measurements/by-date/{start}/{end}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	`start`	N	String	The start time of the time range, in ISO8601 format
	end	N	String	The end time of the time range, in ISO8601 format
Request body type	-			
Response class type	[GET] List<Server_ measurement>			
Used by	Energy efficiency Policy Maker & Actuator			
Example	**Request:**`GET /api/server_measurements/by-date/2015-02-16T23:00:00Z/2015-02-17/` **Response (application/json):** `[ { "id": 1, "value": "16384.00", "time": "2015-02-16T23:00:00Z", "type": "memory.usage", "serial_number": "SN12346" } ]`			

Server Measurement				
Description	This service allows for retrieving the server measurements of a specific server, based on their serial number			
Endpoint Name	`/api/vm_measurements/by-serial_number/{serial_number}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	serial_number	N	String	The serial number of the server to fetch information for
Request body type	-			
Response class type	[GET] List<Server_ measurement>			
Used by	Energy efficiency Policy Maker & Actuator			
Example	**Request:**`GET /api/server_measurements/by-serial_number/SN12346/` **Response (application/json):** `[ { "id": 1, "value": "16384.00", "time": "2015-02-16T23:00:00Z", "type": "memory.usage", "serial_number": "SN12346" } ]`			

Server Measurement				
Description	This service allows for retrieving server measurements of a specific server that fall under a specific time range			
Endpoint Name	`/api/server_measurements/by-serial_number/{serial_number}/{start}/{end}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	`serial_number`	N	String	The serial number of the server to fetch information for
	`start`	N	String	The start time of the time range, in ISO8601 format
	`end`	N	String	The end time of the time range, in ISO8601 format
Request body type	-			
Response class type	[GET] List<Server_ measurement>			
Used by	Energy efficiency Policy Maker & Actuator			

Example	**Request:**`GET /api/server_measurements/by-serial_number/SN12346/2015-02-15/2015-02-17/`
	Response (application/json):
	`[`
	`{`
	`"id": 1,`
	`"value": "16384.00",`
	`"time": "2015-02-16T23:00:00Z",`
	`"type": "memory.usage",`
	`"serial_number": "SN12346"`
	`}`
	`]`			

Metric				
Description	This service allows for creating new and retrieving existing metrics of the DC performance			
Endpoint Name	`/api/metrics/`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	-	-	-	-
Request body type	-			
Response class type	[GET] List<Metric>			
Used by	Energy efficiency Policy Maker & Actuator			

Example	**Request:**`GET /api/metrics/`
	Response (application/json):
	`[`
	`{`
	`"name": "PUE",`			

```
            "description": "Power usage effectiveness (PUE) is a
                            measure of how efficiently a computer
                            data center uses energy; specifically,
                            how much energy is used by the computing
                            equipment (in contrast to cooling and
                            other overhead)."

        },

        {

            "name": "DCA",

            "description": "DCAdapt (DCA) indicates the ability of a
                            DC to adapt to external changes in terms
                            of energy consumption shifting"

        },

        {

            "name": "DCiE",

            "description": "It is the inverse of the PUE"

    }
    ]
```

				Metric
Description	This service allows for retrieving information related to a specific metric of the DC performance, based on their name			
Endpoint Name	`/api/metrics/{name}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	name	N	String	The abbreviated term of the metric name
Request body type	-			
Response class type	[GET] Metric			
Used by	Energy efficiency Policy Maker & Actuator			

Example	**Request:**`GET /api/metrics/PUE/`
	Response (application/json):
	`{`
	`"name": "PUE",`
	`"description": "Power usage effectiveness (PUE) is a measure of how efficiently a computer data center uses energy; specifically, how much energy is used by the computing equipment (in contrast to cooling and other overhead)."`
	`}`			

Metric Measurement				
Description	This service allows for creating new and retrieving existing measurements of metrics of the DC performance			
Endpoint Name	`/api/metric_measurements/`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	-	-	-	-
Request body type	-			
Response class type	[GET] List<Metric_measurement>			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			
Example	**Request:**`GET /api/metric_measurements/` **Response (application/json):** `[ { "id": 1, "value": "1.60", "time": "2015-02-17T10:59:00Z", "metric": "PUE" }, { "id": 2, "value": "0.63", "time": "2015-02-17T10:59:00Z", "metric": "DCiE" } ]`			

Metric Measurement				
Description	This service allows for retrieving measurements of the metrics of the DC performance, based on their id			
Endpoint Name	`/api/metric_measurements/{id}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	id	N	Integer	The ID of the metric measurement to handle
Request body type	Metric _ measurement (for PUT method only)			
Response class type	[GET] Metric_ measurement			
Used by	Energy efficiency Policy Maker & Actuator			
Example	**Request:**`GET /api/metric_measurements/1` **Response (application/json):** `{ "id": 1, "value": "1.60", "time": "2015-02-17T10:59:00Z", "metric": "PUE" }`			

<table>
<tr><th colspan="5">Metric Measurement</th></tr>
<tr><td>Description</td><td colspan="4">This service allows for retrieving metric measurements that fall under a specific time range</td></tr>
<tr><td>Endpoint Name</td><td colspan="4">/api/metric_measurements/by-date/{start}/{end}</td></tr>
<tr><td>Allowed methods</td><td colspan="4">GET</td></tr>
<tr><td>Request Parameters</td><td>Parameter</td><td>Optional (Y/N)</td><td>Type</td><td>Description</td></tr>
<tr><td></td><td>start</td><td>N</td><td>String</td><td>The start time of the time range, in ISO8601 format</td></tr>
<tr><td></td><td>end</td><td>N</td><td>String</td><td>The end time of the time range, in ISO8601 format</td></tr>
<tr><td>Request body type</td><td colspan="4">-</td></tr>
<tr><td>Response class type
Used by</td><td colspan="4">[GET] List<Metric_ measurement>
Energy efficiency Policy Maker & Actuator</td></tr>
<tr><td>Example</td><td colspan="4">Request:GET /api/metric_measurements/by-date/2015-02-17/2015-02-18/
Response (application/json):
[
{
"id": 1,
"value": "1.60",
"time": "2015-02-17T10:59:00Z",
"metric": "PUE"
},
{
"id": 2,
"value": "0.63",
"time": "2015-02-17T10:59:00Z",
"metric": "DCiE"
}
]</td></tr>
</table>

Metric Measurement				
Description	This service allows for retrieving the metric measurements of a specific metric, based on their name			
Endpoint Name	`/api/vm_measurements/by-metric/{metric}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	metric	N	String	The name of the metric to fetch information for
Request body type	-			
Response class type	[GET] List<Metric_ measurement>			
Used by	Energy efficiency Policy Maker & Actuator			

Example	**Request:**GET `/api/metric_measurements/by-metric/PUE/`
	Response (application/json):
	`[`
	`{`
	`"id": 1,`
	`"value": "1.60",`
	`"time": "2015-02-17T10:59:00Z",`
	`"metric": "PUE"`
	`}`
	`]`			

Metric Measurement				
Description	This service allows for retrieving the metric measurements of a specific metric that fall under a specific time range			
Endpoint Name	`/api/metric_measurements/by-metric/{metric}/{start}/{end}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	`metric`	N	String	The name of the metric to fetch information for
	`start`	N	String	The start time of the time range, in ISO8601 format
	`end`	N	String	The end time of the time range, in ISO8601 format
Request body type	-			
Response class type	[GET] List<Metric_ measurement>			
Used by	Energy efficiency Policy Maker & Actuator			

Example	**Request:**GET `/api/metric_measurements/by-metric/PUE/2015-02-15/2015-02-18/`
	Response (application/json):
	`[`
	`{`
	`"id": 1,`
	`"value": "1.60",`
	`"time": "2015-02-17T10:59:00Z",`
	`"metric": "PUE"`
	`}`
	`]`			

DCO Hypervisor Broker injection API				
Description	Used by DCO Hypervisor Broker to post event updates toward the ICT Performance and Energy Supervisor core			
Endpoint Name	`ict_core`			
RPC Call name(s)	`dco_hyperv_update`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	originId	N	String	Specifies the origin context of the data
	payload	N	JSON	Completely describes the occurred status change
Provides response	NO			
Response Parameters	none			
Requester	DCO Hypervisor Broker			
Managed errors	312 - No Route 313 - No Consumers 403 - Access Refused 404 - Not Found 405 - Already Exists 406 - In Use 407 - Invalid Routing Key 408 - Request Timeout 409 - Invalid Argument 530 - Not Allowed 542 - Unsupported Protocol Version			

DCO Monitor Broker injection API				
Description	Used by DCO Monitor Broker to post event updates toward the ICT Performance and Energy Supervisor core			
Endpoint Name	`ict_core`			
RPC Call name(s)	`dco_monitor_update`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	originId	N	String	Specifies the origin context of the data
	payload	N	JSON	Completely describes the occurred status change
Provides response	NO			
Response Parameters	none			
Requester	DCO Monitor Broker			
Managed errors	312 - No Route 313 - No Consumers 403 - Access Refused 404 - Not Found 405 - Already Exists 406 - In Use 407 - Invalid Routing Key 408 - Request Timeout 409 - Invalid Argument 530 - Not Allowed 542 - Unsupported Protocol Version			

DCO Appl. Broker injection API				
Description	Used by DCO Appl. Broker to post event updates toward the ICT Performance and Energy Supervisor core			
Endpoint Name	`ict_core`			
RPC Call name(s)	`dco_appl_update`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	originId	N	String	Specifies the origin context of the data
	payload	N	JSON	Completely describes the occurred status change
Provides response	NO			
Response Parameters	none			
Requester	DCO Appl. Broker			
Managed errors	312 - No Route 313 - No Consumers 403 - Access Refused 404 - Not Found 405 - Already Exists 406 - In Use 407 - Invalid Routing Key 408 - Request Timeout 409 - Invalid Argument 530 - Not Allowed 542 - Unsupported Protocol Version			

DCO Topology update API				
Description	Used by ICT Performance and Energy Supervisor core to post topology updates to the DCO Topology Graph Database and Energy Policy Maker & Actuator			
Endpoint Name	`dco_topology`			
RPC Call name(s)	`dco_topology_update`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	hypervisorId	N	String	Notifying hypervisor system
	eventType	N	Enum	VM activation or deactivation
	vmId	N	String	VM asset object of notification
	vmMetadata	Y	JSON	VM metadata & assets description
Provides response	YES			
Response Parameters	Standard ACK			
Requester	ICT Performance and Energy Supervisor core			
Managed errors	312 - No Route 313 - No Consumers 403 - Access Refused 404 - Not Found 405 - Already Exists 406 - In Use 407 - Invalid Routing Key 408 - Request Timeout 409 - Invalid Argument 530 - Not Allowed 542 - Unsupported Protocol Version			

<table>
<tr><th colspan="5">Topology retrieval</th></tr>
<tr><td>Description</td><td colspan="4">This API allows to retrieve the bulk of the DC topology</td></tr>
<tr><td>Endpoint Name</td><td colspan="4">/api/topology/{startingNode}/{depth}</td></tr>
<tr><td>Allowed methods</td><td colspan="4">GET</td></tr>
<tr><td>Request Parameters</td><td>Parameter</td><td>Optional (Y/N)</td><td>Type</td><td>Description</td></tr>
<tr><td></td><td>startingNode</td><td>Y</td><td>String</td><td>UUID if the topology node to inspect. The tree structure of this node is returned. If not specified, or “0” (zero), the topology root node is selected</td></tr>
<tr><td></td><td>depth</td><td>Y</td><td>Integer</td><td>Depth of tree visit. At most, {depth} levels of the subtree are reported. If omitted, the whole subtree is reported.</td></tr>
<tr><td>Response class type</td><td colspan="4">Topology_subtree (json)</td></tr>
<tr><td>Used by</td><td colspan="4">ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager</td></tr>
<tr><td>Example</td><td colspan="4">Request:/api/topology/{startingNode}/{depth}

Response (application/json):

{
"nodeType": "<<type of the node>>",
"id": "<<some-uuid>>",
"metadata": …,
"children": [
<<recursive structure>>,
<<recursive structure>>,
<<recursive structure>>, …
]
}</td></tr>
</table>

Metrics Extraction API				
Description	Used by ICT Performance and Energy Supervisor core to request an update of the DC metrics			
Endpoint Name	`metrics`			
RPC Call name(s)	`metrics_recompute`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	dcData	N	JSON	Sequenced partial aggregation of DC measurement and performance data
Provides response	YES			
Response Parameters	Dictionary of metric values			
Requester	ICT Performance and Energy Supervisor core			
Managed errors	-			

A 1.4 Energy efficiency Policy Maker & Actuator

In this section will be provide the detail description of each APIs exposed and used by this module that will be provided by other modules in the DOLFIN architecture. With “external interface" refers to all the interfaces out from the module’s boundary.

Policy Plan Submit				
Description	This service allows a Policy Plan to be submitted for execution			
Endpoint Name	`/api/policy_plan/`			
Allowed methods	POST			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	-	-	-	-
Request body type	application/json			
Response class type	[POST] Policy_plan_type			
Used by	ICT Performance and Energy Supervisor			
Example	**Request:**POST `/api/policy_plan/`**(application/json):** [{ "action_id": "<<uuid>>", "actor": "<<uuid>>", "action": { …json structure describing the action…}, "final_status": "STATUS_NAME", }, {			

```
    "action_id": "<<uuid>>",
    "actor": "<<uuid>>",
    "action": { …json structure describing the action…},
    "final_status": "STATUS_NAME",
  },
  {
    "action_id": "<<uuid>>",
    "actor": "<<uuid>>",
    "action": { …json structure describing the action…},
    "final_status": "STATUS_NAME",
  },
  {
    "action_id": "<<uuid>>",
    "actor": "<<uuid>>",
    "action": { …json structure describing the action…},
    "final_status": "STATUS_NAME",
  },
  {
    "action_id": "<<uuid>>",
    "actor": "<<uuid>>",
    "action": { …json structure describing the action…},
    "final_status": "STATUS_NAME",
  },
]
```


Notes	**The "action" JSON key describes the action to be undertaken by the DOLFIN system, as a compound of steps, in order to consider an action in the policy plan completed. It consists of a tuple of basic operations involving DC physical and logical assets, such as server and VMs. The typical policy optimization plan focuses mostly on VM migrations and server consolidation so, it can be inferred that a vocabulary of terms specifying these actions will have to be formulated at a later phase.**

Subscribe Data Event Feed				
Description	This service allows to the caller to request the notification of data events from the ICT Performance and Energy supervisor, via a callback			
Endpoint Name	`/api/subscribe_data_event`			
Allowed methods	PUT			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	streamId	N	String	UUID of the data stream to subscribe to
	subscriberId	N	String	Name of the subscriber actor
	callbackUri	N	URI	REST Callback address
Request body type	Empty			
Response class type	Subscription ID (string)			
Used by	Energy Policy Maker & Actuator			

Example	**Request:**PUT `/api/subscribe_data_event`
	`{`
	`" streamId ": "<<some-uuid>>",`
	`"subscriberId": "<<some-uuid>>",`
	`"callbackUri": "amqp://queue/exchangename?topic=duh"`
	`}`
	Response (application/json):
	`{`
	`"subscriptionId": "<<some-uuid>>"`
	`}`			

Data change event notification				
Description	This API is invoked by the ICT Performance and Energy supervisor upon the subscriber of a data event stream in order to deliver data events			
Endpoint Name	`/api/data_event/{eventId}`			
Allowed methods	PUT			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	eventId	N	String	UUID of the data event
	streamId	N	String	UUID if the subscribed data stream
	sata	N	String	New status of the data stream
Request body type	JSON			
Response class type	Empty			
Used by	ICT Performance and Energy Supervisor			

Example	**Request:**PUT `/api/data_event/<<some-uuid>>`
	Request (application/json):
	`{`
	`"streamId": "<<some-uuid>>",`
	`"data": "new value"`
	`}`			

Unsubscribe Data Event Feed				
Description	This service allows to the caller to unsubscribe from the notification of data events from the ICT Performance and Energy supervisor			
Endpoint Name	`/api/subscribe_data_event/{subscriptionId}`			
Allowed methods	DELETE			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	subscriptionId	N	String	UUID of the subscription channel
Request body type	Empty			
Response class type	Empty			
Used by	Energy Policy Maker & Actuator			

Request metric / measure				
Description	This service allows to the caller to request the notification of data events from the ICT Performance and Energy supervisor, via a callback			
Endpoint Name	`/api/measure/{streamId}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	streamId	N	String	UUID of the data stream to read
Request body type	Empty			
Response class type	JSON			
Used by	Energy Policy Maker & Actuator			

Example	**Response (application/json):**
	`{`
	`"timestamp": "<<ISO-8601 UTC timestamp>>",`
	`"data": "<<most recent value>>"`
	`}`			

VM Migration				
Description	This API is used to trigger the migration of an existing VM, changing the topology of the DC by reparenting one leaf asset			
Endpoint Name	`/api/vm_migrate/{vmId}`			
Allowed methods	PUT			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	vmId	N	String	ID if the VM asset, subject of the migration operation
	newParentId	N	String	ID of the new parent asset, which shall be an hypervisor object capable of hosting the VM
Request body type	Asset reparent (json)			
Response class type	Acknowledge structure (json)			
Used by	Energy Policy Maker & Actuator			
Example	**Request:**PUT `/api/vm_migrate/<<some-uuid>>` `{`			

```
    "newParentId": "<<some-uuid>>"
}
```

Response (application/json):

```
{
    "status": "OK/ERROR",
    "error_code": "optional",
    "error_msg": "optional"
}
```

	VM Status Change			
Description	This API is used to initiate a status change for an existing VM			
Endpoint Name	`/api/vm_status/{vmId}`			
Allowed methods	POST			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	vmId	N	String	ID if the VM asset, subject of the migration operation
	newStatus	N	String	Name of the new status to put the VM into
Request body type	Asset status change (json)			
Response class type	Acknowledge structure (json)			
Used by	Energy Policy Maker & Actuator			

Example	**Request:**POST `/api/vm_status/<<some-uuid>>`
	`{`
	`"newStatus": "<<status name>>"`
	`}`
	Response (application/json):
	`{`
	`"status": "OK/ERROR",`
	`"error_code": "optional",`
	`"error_msg": "optional"`
	`}`			

	Consumption Trend Inquiry
Description	This service allows to discover the trend of the optimal consumption rates produced by the Smart Grid for the next timeslots
Endpoint Name	`/api/consumption_trend/`
Allowed methods	GET
Response class type	[GET] Consumption trend timeslotted sequence
Used by	Energy Policy Maker & Actuator

Example	**Request:**GET `/api/consumption_trend/`
	Response (application/json):
	`{`
	`"trend": {`
	`"start_timestamp": "<<ISO-8601 UTC timestamp>>",`
	`"timeslot_size": <<number of seconds>>,`
	`"expected_kWh": [`
	`<<kWh_timeslot1>>,`
	`<<kWh_timeslot2>>,`
	`<<kWh_timeslot3>>,`
	`<<kWh_timeslot4>>,…`
	`],`
	`}`
	`}`

	Consumption Energy Price
Description	This service allows to discover the energy consumption prices applied by the Smart Grid for the next timeslots
Endpoint Name	`/api/consumption_pricing/`
Allowed methods	GET
Response class type **Used by**	[GET] Consumption trend timeslotted sequence Energy Policy Maker & Actuator

Example	**Request:**GET `/api/consumption_pricing/`
	Response (application/json):
	`{`
	`"trend": {`
	`"start_timestamp": "<<ISO-8601 UTC timestamp>>",`
	`"timeslot_size": <<number of seconds>>,`
	`"kWh_cost": [`
	`<<cost_timeslot1>>,`
	`<<cost_timeslot2>>,`
	`<<cost_timeslot3>>,`
	`<<cost_timeslot4>>,…`
	`],`
	`}`
	`}`

**The Smart Grid Controller should provide a List<Energy_prices> to the optimizer via the Policy Maker with energy prices over time.*

	SLA Downgrade Plan			
Description	This service queries the SLA Renegotiation Controller in order to obtain a list of VMs which are available for performance reduction			
Endpoint Name	`/api/sla_downgrade/{interval}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	interval	N	Integer	Expresses (in sec) the time interval length which will see the VMs downgraded
Response class type **Used by**	List_of_VM_Constraints (json) Energy Policy Maker & Actuator			

Example	**Request:**GET `/api/sla_downgrade/{interval}`
	Response (application/json):
	`{`
	`"vms": [`
	`{`
	`"vmId": "<<some-uuid>>",`
	`"minPerformances": "<<status_name>>",`
	`"maxDowngradeTime": <<number_of_seconds>>,`
	`},`
	`{`
	`"vmId": "<<some-uuid>>",`
	`"minPerformances": "<<status_name>>",`
	`"maxDowngradeTime": <<number_of_seconds>>,`
	`},`
	`{`
	`"vmId": "<<some-uuid>>",`			

```
                    "minPerformances": "<<status_name>>",
                    "maxDowngradeTime": <<number_of_seconds>>,
                },…
            ]

        }
```

Notes	**This query returns, for every pertinent and known VM instance a descriptor stating the maximum computational downgrade which the corresponding SLA allows for (minPerformances) and the maximum duration of the downgrade which may be tolerated by the SLA, on a monthly basis (maxDowngradeTime)**

				Cross boundary Inter-DC VM Migration (Inquiry)
Description	This API is used to inquiry about the potential migration of an existing VM across two DC boundaries			
Endpoint Name	`/api/xdc_vm_migrate/{vmId}`			
Allowed methods	PUT			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	vmId	N	String	ID if the VM asset, subject of the migration operation
	newDatacenterId	Y	String	ID of the receiving datacenter. If unspecified, all cooperating DC controllers are queried
Request body type	Asset XDC migration (json)			
Response class type	Acknowledge structure (json)			
Used by	Energy Policy Maker & Actuator			

Example	**Request:**PUT `/api/xdc_vm_migrate/<<some-uuid>>`
	`{`
	`" newDatacenterId ": "<<some-uuid>>"`
	`}`
	Response (application/json):
	`{`
	`" newDatacenterId ": "<<some-uuid>>"`
	`"xDCMigrationRequestId": "<<some-uuid>>",`
	`"error_code": "optional",`
	`"error_msg": "optional"`
	`}`			

				Cross boundary Inter-DC VM Migration (Trigger)
Description	This API is used to trigger the actual migration of an existing VM across two DC boundaries			
Endpoint Name	`/api/xdc_vm_migrate/{vmId}/{xDCMigrationRequestId}`			
Allowed methods	PUT			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	vmId	N	String	ID if the VM asset, subject of the migration operation
	xDCMigrationRequestId	N	String	ID of the prepared and authorized migration transaction
Response class type	Acknowledge structure (json)			

Used by	Energy Policy Maker & Actuator

Example	**Request:**PUT xdc_vm_migrate/{vmId}/{xDCMigrationRequestId}
	Response (application/json):
	{
	"status": "OK/ERROR",
	"error_code": "optional",
	"error_msg": "optional"
	}

				Log notification
Description	This service allows other DOLFIN subsystem to save log entries in a global system catalog, working as a registry of operations			
Endpoint Name	/api/log/{actorId}/{level}			
Allowed methods	PUT			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	actorId	N	String	ID of the notificator subsystem (origin of the log entry)
	level	N	String	Canonical log entry level (INFO, DEBUG, ERROR, etc..)
Request body type	application/text (raw UTF-8 body)			
Used by	*			

				Load Predictions
Description	Provides a list of predictions regarding the DC load in the near future			
Endpoint Name	prediction_engine			
RPC Call name(s)	get_predictions			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	duration	NO	Integer	The duration of the predictions
	dc_segment			
Provides response	YES			
Response Type	List<Future_load>			
Requester	Optimizer			
Managed errors	**TBD**			

		Class: Future_load
Description	This class represents the near future load over time	
Attribute name	**Attribute type**	**Description**
time	String	Time of the prediction, in ISO8601 format
cpu_util	double	% of the DC aggregate CPU utilization
ram_util	double	% of the DC aggregate RAM utilization
hdd_util	double	% of the DC aggregate HDD utilization
network_util	double	% of the DC network utilization

	VM priorities
Description	Provides a list of classified VMs based on their resource utilization and the SLAs of the user

Endpoint Name	`VM_priority_classifier`			
RPC Call name(s)	`get_VM_priorities`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	vm_list	NO	List<VMs>	The list of accepted VMs
	sla_map	NO	Map<String,List<SLA>>	SLA priority of each VM
	vm_origin	NO	String	ID of DC sending the VM
	cpu_util_map	NO	Map<String, List<Measurement>>	CPU utilization of each VM
	ram_util_map	NO	Map<String, List<Measurement>>	RAM utilization of each VM
	network_util_map	NO	Map<String, List<Measurement>>	Network utilization of each VM
	hdd_util_map	NO	Map<String, List<Measurement>>	HDD utilization of each VM
	server_cpu_freq	YES	Map<String, Integer>	The VM - CPU frequency of the VM host map
Provides response	YES			
Response Type	List<Vm_priorities>			
Requester	Optimizer			
Managed errors	**TBD**			

Class: Vm_priorities		
Description	This class represents the priority classes of a VM	
Attribute name	**Attribute type**	**Description**
vm_uuid	String	The uuid of the VM
resource_priority	Integer	The priority of the VM based on its recent resource usage
sla_priority	Integer	The SLA priority of the VM

Server Energy Classification				
Description	Provides a list of classified servers based on their energy efficiency and DC layout			
Endpoint Name	`ict_performance`			
RPC Call name(s)	`get_server_priorities`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	-	-	-	-
Provides response	YES			
Response Type	List<Server_priorities>			
Requester	Optimizer			
Managed errors	**TBD**			

Class: Server_priorities		
Description	This class represents the priority of a server. Lower priority numbers indicate servers that should be filled with a higher priority.	
Attribute name	**Attribute type**	**Description**
server_sn	String	The serial number of the server
priority	Integer	The priority of the server

Server Energy Consumption				
Description	Provides the energy consumption of each server			
Endpoint Name	`ict_performance`			
RPC Call name(s)	`get_server_consumption`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	start_time	NO	String	Start time of the measurement, in ISO8601 format
	end_time	NO	String	End time of the measurement, in ISO8601 format
Provides response	YES			
Response Type	List<Server_consumption>			
Requester	Optimizer			
Managed errors	**TBD**			

Class: Server_consumption		
Description	This class represents the power consumption of each server at a given time	
Attribute name	**Attribute type**	**Description**
server_sn	String	The serial number of the server under consideration
power	Double	The power consumption of the server
start_time	String	Start time of the measurement, in ISO8601 format
end_time	String	End time of the measurement, in ISO8601 format

VM Allocation				
Description	Provides a map of the VM allocation to the DC servers			
Endpoint Name	`policy_actuator`			
RPC Call name(s)	`set_vm_allocation`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	allocation	NO	Map<String vm_uuid,String server_sn>	VM allocation to Servers
Provides response	YES			
Response Type				
Requester	Optimizer			
Managed errors	**TBD**			

A 1.4.1 Optimizer - Actuator Vocabulary:

	VM_migrate			
Description	Provides a map of the VM migration to the DC servers			
Endpoint Name	`Policy_actuator`			
RPC Call name(s)	`set_vm_migration`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	migration	NO	Map<String vm_uuid,String server_sn>	VM migration to Servers
Provides response	YES			
Response Type	Status Codes			
Requester	Optimizer			
Managed errors	**TBD**			

	VM_shift			
Description	Provides the time period at which a VM should be instantiated			
Endpoint Name	`Policy_actuator`			
RPC Call name(s)	`set_vm_shift`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	shift	NO	Map<String vm_uuid,String start_time>	Start time of VMs. Start time of the VMs, in ISO8601 format
Provides response	YES			
Response Type	Status Codes			
Requester	Optimizer			
Managed errors	**TBD**			

				DVFS
Description	Provides a map of the DVFS of servers			
Endpoint Name	`Policy_actuator`			
RPC Call name(s)	`set_DFVS`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	dvfs	NO	Map<String server_sn, Integer, scale>	Voltage and frequency scaling of servers by a percentage scale%
Provides response	YES			
Response Type	Status Codes			
Requester	Optimizer			
Managed errors	**TBD**			

				Serv_term
Description	Provides a list of the DC servers to be put in hibernation mode			
Endpoint Name	`Policy_actuator`			
RPC Call name(s)	`set_server_termination`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	Serv_term	NO	List<server_sn>	Servers to be put in hibernation mode
Provides response	YES			
Response Type	Status Codes			
Requester	Optimizer			
Managed errors	**TBD**			

				Serv_oper
Description	Provides a list of the DC servers to be put back in operation mode			
Endpoint Name	`Policy_actuator`			
RPC Call name(s)	`set_server_operation`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	Serv_oper	NO	List<server_sn>	Servers to be put in operation mode
Provides response	YES			
Response Type	Status Codes			
Requester	Optimizer			
Managed errors	**TBD**			

				Ancil_term
Description	Provides a list of the DC ancillary equipment that should be put in hibernation mode			
Endpoint Name	`Policy_actuator`			
RPC Call name(s)	`set_ancillary_termination`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	Ancil_term	NO	List<Air_cond_sn>	Ancillary equipment that should be put in hibernation mode
Provides response	YES			
Response Type	Status Codes			
Requester	Optimizer			
Managed errors	**TBD**			

				Ancil_oper
Description	Provides a list of the DC ancillary equipment that should be put back in operation mode			
Endpoint Name	`Policy_actuator`			
RPC Call name(s)	`set_ancillary_operation`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	Ancil_oper	NO	List<Air_cond_sn>	Ancillary equipment that should be put in operation mode
Provides response	YES			
Response Type	Status Codes			
Requester	Optimizer			
Managed errors	**TBD**			

				Air_cond _temper
Description	Provides a map of the DC air conditioners and their operating temperature			
Endpoint Name	`Policy_actuator`			
RPC Call name(s)	`set_air_cond_temperature`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	Air_cond_temper	NO	Map<String Air_cond_sn, Integer, temperature>	Operating temperature of DC air conditioners
Provides response	YES			
Response Type	Status Codes			
Requester	Optimizer			
Managed errors	**TBD**			

A 1.4.2 Optimizer – Policy Maker Vocabulary:

				VM_relocate
Description	Requests the relocation of a VM set			
Endpoint Name	`Policy Maker`			
RPC Call name(s)	`set_vm_relocation`			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**
	vm_list	NO	List<VMs>	The list of VMs to be relocated
	sla_map	NO	Map<String,List<SLA>>	SLA priority of each VM
	vm_origin	NO	String	ID of DC sending the VM
	cpu_util_map	NO	Map<String, List<Measurement>>	CPU utilization of each VM
	ram_util_map	NO	Map<String, List<Measurement>>	RAM utilization of each VM
	network_util_map	NO	Map<String, List<Measurement>>	Network utilization of each VM
	hdd_util_map	NO	Map<String, List<Measurement>>	HDD utilization of each VM
Provides response	YES			
Response Type	Status Codes			
Requester	Optimizer			
Managed errors	**TBD**			

A 1.5 eCOP Monitor Database

In the following tables, the various interfaces exposed by the Storage Broker are documented. As the Storage Broker exposes RESTful APIs to interact with the various modules of the general DOLFIN architecture, all relevant output formats are supported, including JSON, XML and YAML. However, for brevity reasons, only JSON responses are presented in the examples documented in the following service description tables.

Please, note that these tables refer to the initial eCOP Monitor Database design and will be subject to future changes, following the developments related to the actual implementation of the component.

	Measurement Types
Description	This service allows for creating new and retrieving existing measurement types.
Endpoint Name	`/api/measurement_types/`
Allowed methods	GET, POST
Request Parameters	**Parameter** / **Optional (Y/N)** / **Type** / **Description** - / - / - / -
Request body type	Measurement_type (for POST method only)
Response class type	[GET] List<Measurement_type> [POST] Measurement_type
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager

Example	**Request:**`GET /api/measurement_types/`
	Response (application/json):
	[
	{
	"name": "cpu_util",
	"unit": "%",
	"description": "The CPU utilization of the entity"
	},
	{
	"name": "memory",
	"unit": "MB",
	"description": "The total RAM available to the entity"
	},
	{
	"name": "memory.usage",
	"unit": "MB",
	"description": "The current memory utilization of the entity"
	},
	{
	"name": "hdd",
	"unit": "GB",
	"description": "The total HDD available to the entity"
	},
	{
	"name": "hdd.usage",
	"unit": "GB",
	"description": "The current HDD utilization of the entity"
	}
	]

Measurement Types				
Description	This service allows for retrieving, updating and deleting specific measurement types, based on their name			
Endpoint Name	`/api/measurement_types/{name}`			
Allowed methods	GET, PUT, DELETE			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	name	N	String	The name of the measurement type to handle
Request body type	Measurement_type (for PUT method only)			
Response class type	[GET, PUT] Measurement_type			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			

Example	**Request:**GET `/api/measurement_types/cpu_util`
	Response (application/json):
	`{`
	`"name": "cpu_util",`
	`"unit": "%",`
	`"description": "The CPU utilization of the entity"`
	`}`			

VM				
Description	This service allows for creating new and retrieving existing information related to the VM instances of the DC			
Endpoint Name	`/api/vms/`			
Allowed methods	GET, POST			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	-	-	-	-
Request body type	VM (for POST method only)			
Response class type	[GET] List<VM> [POST] VM			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			

Example	**Request:**GET `/api/vms/`
	Response (application/json):
	`[`
	`{`
	`"name": "VM Instance name",`
	`"uuid": "uuid-12234567",`
	`"flavour": 4,`
	`"server": "SN12347",`
	`"status": "ACTIVE",`
	`"task": "None",`			

```
            "instantiation_time": "2015-02-17T20:00:52.239000Z"
        },
        {
            "name": "Another VM",
            "uuid": "uuid-12234568",
            "flavour": 4,
            "server": "SN12347",
            "status": "ACTIVE",
            "task": "None",
            "instantiation_time": "2015-02-17T20:00:52.239000Z"
        },
        {
            "name": "test-vm-entry",
            "uuid": "1234",
            "flavour": 4,
            "server": "SN12346",
            "status": "ACTIVE",
            "task": "None",
            "instantiation_time": "2015-02-17T20:00:52.239000Z"
    }
    ]
```


VM				
Description	This service allows for retrieving, updating and deleting information related to the VM instances of the DC, based on their uuid			
Endpoint Name	`/api/vms/{uuid}`			
Allowed methods	GET, PUT, DELETE			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	uuid	N	String	The UUID of the VM to handle
Request body type	VM (for PUT method only)			
Response class type	[GET, PUT] VM			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			
Example	**Request:**`GET /api/vms/uuid-12234567` **Response (application/json):** {			

```
    "name": "VM Instance name",
    "uuid": "uuid-12234567",
    "flavour": 4,
    "server": "SN12347",
    "status": "ACTIVE",
    "task": "None",
    "instantiation_time": "2015-02-17T20:00:52.239000Z"
}
```


	VM Measurement			
Description	This service allows for creating new and retrieving existing Measurements of VM instances of the DC			
Endpoint Name	`/api/vm_measurements/`			
Allowed methods	GET, POST			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	-	-	-	-
Request body type	VM_measurement (for POST method only)			
Response class type	[GET] List<VM_ measurement> [POST] VM_measurement			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			
Example	**Request:**GET `/api/vm_measurements/` **Response (application/json):**			


```
[
    {
        "id": 2,
        "value": "2.00",
        "time": "2015-02-17T06:02:51.445000Z",
        "uuid": "uuid-12234567",
        "type": "cpu_util"
    }
    {
        "id": 6,
        "value": "15.00",
        "time": "2015-02-17T10:59:00Z",
        "uuid": "uuid-12234568",
        "type": "cpu_util"
    }
]
```


	VM Measurement			
Description	This service allows for retrieving, updating and deleting measurements of the VM instances of the DC, based on their id			
Endpoint Name	`/api/vm_measurements/{id}`			
Allowed methods	GET, PUT, DELETE			
Request Parameters	**Parameter**	**Optional**	**Type**	**Description**

	(Y/N)			
	id	N	Integer	The ID of the VM measurement to handle
Request body type	VM_ measurement (for PUT method only)			
Response class type	[GET, PUT] VM_ measurement			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			

Example	**Request:**GET /api/vm_measurements/6
	Response (application/json):
	{
	"id": 6,
	"value": "15.00",
	"time": "2015-02-17T10:59:00Z",
	"uuid": "uuid-12234568",
	"type": "cpu_util"
	}			

				VM Measurement
Description	This service allows for retrieving VM measurements that fall under a specific time range			
Endpoint Name	/api/vm_measurements/by-date/{start}/{end}			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	start	N	String	The start time of the time range, in ISO8601 format
	end	N	String	The end time of the time range, in ISO8601 format
Request body type	-			
Response class type **Used by**	[GET] List<VM_ measurement> ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			
Example	**Request:**GET /api/vm_measurements/by-date/2015-02-17/2015-02-18/ **Response (application/json):** [{ "id": 2, "value": "2.00", "time": "2015-02-17T06:02:51.445000Z", "uuid": "uuid-12234567", "type": "cpu_util"			

```
                },
                {
                    "id": 3,
                    "value": "2.00",
                    "time": "2015-02-17T08:02:51.445000Z",
                    "uuid": "uuid-12234567",
                    "type": "cpu_util"
                },
                {
                    "id": 6,
                    "value": "15.00",
                    "time": "2015-02-17T10:59:00Z",
                    "uuid": "uuid-12234568",
                    "type": "cpu_util"
                },
                {
                    "id": 5,
                    "value": "512.00",
                    "time": "2015-02-17T11:58:00Z",
                    "uuid": "uuid-12234567",
                    "type": "memory"
                },
                {
                    "id": 1,
                    "value": "2.00",
                    "time": "2015-02-17T10:01:51.445000+02:00",
                    "uuid": "uuid-123456",
            "type": "1"
                }
            ]
```

VM Measurement				
Description	This service allows for retrieving the VM measurements of a specific VM, based on their UUID			
Endpoint Name	`/api/vm_measurements/by-uuid/{uuid}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	uuid	N	String	The UUID of the VM to fetch information for
Request body type	-			
Response class type	[GET] List<VM_ measurement>			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			

Example	**Request:**GET `/api/vm_measurements/by-uuid/uuid-12234568/`
	Response (application/json):
	`[`
	`{`
	`"id": 6,`
	`"value": "15.00",`
	`"time": "2015-02-17T10:59:00Z",`
	`"uuid": "uuid-12234568",`
	`"type": "cpu_util"`
	`}`
	`]`			

VM Measurement				
Description	This service allows for retrieving VM measurements of a specific VM that fall under a specific time range			
Endpoint Name	`/api/vm_measurements/by-uuid/{uuid}/{start}/{end}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	`uuid`	N	String	The UUID of the VM to fetch information for
	`start`	N	String	The start time of the time range, in ISO8601 format
	`end`	N	String	The end time of the time range, in ISO8601 format
Request body type	-			
Response class type	[GET] List<VM_ measurement>			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			
Example	**Request:**GET `/api/vm_measurements/by-uuid/uuid-12234567/2015-02-17T06:02:51.445000Z/2015-02-17T11:58:00Z/` **Response (application/json):**			

```
[
    {
        "id": 2,
        "value": "2.00",
        "time": "2015-02-17T06:02:51.445000Z",
        "uuid": "uuid-12234567",
        "type": "cpu_util"
    },
    {
        "id": 3,
        "value": "2.00",
        "time": "2015-02-17T08:02:51.445000Z",
        "uuid": "uuid-12234567",
        "type": "cpu_util"
    },
    {
        "id": 5,
        "value": "512.00",
        "time": "2015-02-17T11:58:00Z",
"uuid": "uuid-12234567",
"type": "memory"
    }
]
```


				Server Measurement
Description	This service allows for creating new and retrieving existing measurements of servers of the DC			
Endpoint Name	`/api/server_measurements/`			
Allowed methods	GET, POST			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	-	-	-	-
Request body type	Server_measurement (for POST method only)			
Response class type	[GET] List<Server_measurement> [POST] Server_measurement			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			
Example	**Request:**`GET /api/server_measurements/` **Response (application/json):**			

```
[
    {
        "id": 1,
        "value": "16384.00",
        "time": "2015-02-16T23:00:00Z",
        "type": "memory.usage",
        "serial_number": "SN12346"
    },
    {
        "id": 2,
        "value": "12125.00",
        "time": "2015-02-17T12:01:00Z",
        "type": "memory.usage",
        "serial_number": "SN12347"
    }
]
```


				Server Measurement
Description	This service allows for retrieving, updating and deleting measurements of the servers of the DC, based on their id			
Endpoint Name	`/api/server_measurements/{id}`			
Allowed methods	GET, PUT, DELETE			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	id	N	Integer	The ID of the Server measurement to handle
Request body type	Server_ measurement (for PUT method only)			
Response class type	[GET, PUT] Server_ measurement			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			

Example	**Request:**`GET /api/server_measurements/2`
	Response (application/json):
	`{`
	`"id": 2,`
	`"value": "12125.00",`
	`"time": "2015-02-17T12:01:00Z",`
	`"type": "memory.usage",`
	`"serial_number": "SN12347"`
	`}`			

Server Measurement				
Description	This service allows for retrieving server measurements that fall under a specific time range			
Endpoint Name	`/api/server_measurements/by-date/{start}/{end}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	`start`	N	String	The start time of the time range, in ISO8601 format
	end	N	String	The end time of the time range, in ISO8601 format
Request body type	-			
Response class type	[GET] List<Server_ measurement>			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			

Example	**Request:**`GET /api/server_measurements/by-date/2015-02-16T23:00:00Z/2015-02-17/`
	Response (application/json):
	`[`
	`  {`
	`  "id": 1,`
	`  "value": "16384.00",`
	`  "time": "2015-02-16T23:00:00Z",`
	`  "type": "memory.usage",`
	`  "serial_number": "SN12346"`
	`  }`
	`]`			

Server Measurement				
Description	This service allows for retrieving the server measurements of a specific server, based on their serial number			
Endpoint Name	`/api/vm_measurements/by-serial_number/{serial_number}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	serial_number	N	String	The serial number of the server to fetch information for
Request body type	-			
Response class type	[GET] List<Server_ measurement>			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			

Example	**Request:**`GET /api/server_measurements/by-serial_number/SN12346/`
	Response (application/json):
	`[`
	`  {`
	`  "id": 1,`
	`  "value": "16384.00",`
	`  "time": "2015-02-16T23:00:00Z",`
	`  "type": "memory.usage",`
	`  "serial_number": "SN12346"`
	`  }`			

Server Measurement				
Description	This service allows for retrieving server measurements of a specific server that fall under a specific time range			
Endpoint Name	`/api/server_measurements/by-serial_number/{serial_number}/{start}/{end}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	`serial_number`	N	String	The serial number of the server to fetch information for
	`start`	N	String	The start time of the time range, in ISO8601 format
	`end`	N	String	The end time of the time range, in ISO8601 format
Request body type	-			
Response class type	[GET] List<Server_ measurement>			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			

Example	**Request:**`GET /api/server_measurements/by-serial_number/SN12346/2015-02-15/2015-02-17/`
	Response (application/json):
	`[`
	`{`
	`"id": 1,`
	`"value": "16384.00",`
	`"time": "2015-02-16T23:00:00Z",`
	`"type": "memory.usage",`
	`"serial_number": "SN12346"`
	`}`
	`]`			

Metric				
Description	This service allows for creating new and retrieving existing metrics of the DC performance			
Endpoint Name	`/api/metrics/`			
Allowed methods	GET, POST			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	-	-	-	-
Request body type	Metric (for POST method only)			
Response class type	[GET] List<Metric> [POST] Metric			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			
Example	**Request:**`GET /api/metrics/`			

Response (application/json):

```
[
    {
        "name": "PUE",
        "description": "Power usage effectiveness (PUE) is a
                        measure of how efficiently a computer
                        data center uses energy; specifically,
                        how much energy is used by the computing
                        equipment (in contrast to cooling and
                        other overhead)."
    },
    {
        "name": "DCA",
        "description": "DCAdapt (DCA) indicates the ability of a
                        DC to adapt to external changes in terms
                        of energy consumption shifting"
    },
    {
        "name": "DCiE",
        "description": "It is the inverse of the PUE"
}
]
```

	Metric			
Description	This service allows for retrieving, updating and deleting information related to a specific metric of the DC performance, based on their name			
Endpoint Name	`/api/metrics/{name}`			
Allowed methods	GET, PUT, DELETE			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	name	N	String	The abbreviated term of the metric name
Request body type	Metric (for PUT method only)			
Response class type	[GET, PUT, DELETE] Metric			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			

Example	**Request:**`GET /api/metrics/PUE/`
	Response (application/json):
	`{`
	`"name": "PUE",`
	`"description": "Power usage effectiveness (PUE) is a measure of how efficiently a computer data center uses energy; specifically, how much energy is used by the computing equipment (in contrast to cooling and other overhead)."`
	`}`			

Metric Measurement				
Description	This service allows for creating new and retrieving existing measurements of metrics of the DC performance			
Endpoint Name	`/api/metric_measurements/`			
Allowed methods	GET, POST			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	-	-	-	-
Request body type	Metric_measurement (for POST method only)			
Response class type	[GET] List<Metric_measurement> [POST] Metric_measurement			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			

Example	**Request:**GET `/api/metric_measurements/`
	Response (application/json):
	`[`
	`{`
	`"id": 1,`
	`"value": "1.60",`
	`"time": "2015-02-17T10:59:00Z",`
	`"metric": "PUE"`
	`},`
	`{`
	`"id": 2,`
	`"value": "0.63",`
	`"time": "2015-02-17T10:59:00Z",`
	`"metric": "DCiE"`
	`}`
	`]`			

Metric Measurement				
Description	This service allows for retrieving, updating and deleting measurements of the metrics of the DC performance, based on their id			
Endpoint Name	`/api/metric_measurements/{id}`			
Allowed methods	GET, PUT, DELETE			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	id	N	Integer	The ID of the metric measurement to handle
Request body type	Metric _ measurement (for PUT method only)			
Response class type	[GET, PUT] Metric_ measurement			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			
Example	**Request:**GET `/api/metric_measurements/1`			

Response (application/json):

```
{
    "id": 1,
    "value": "1.60",
    "time": "2015-02-17T10:59:00Z",
    "metric": "PUE"
}
```

				Metric Measurement
Description	This service allows for retrieving metric measurements that fall under a specific time range			
Endpoint Name	`/api/metric_measurements/by-date/{start}/{end}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	`start`	N	String	The start time of the time range, in ISO8601 format
	`end`	N	String	The end time of the time range, in ISO8601 format
Request body type	-			
Response class type	[GET] List<Metric_ measurement>			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			
Example	**Request:**`GET /api/metric_measurements/by-date/2015-02-17/2015-02-18/` **Response (application/json):** `[ { "id": 1, "value": "1.60", "time": "2015-02-17T10:59:00Z", "metric": "PUE" }, { "id": 2, "value": "0.63", "time": "2015-02-17T10:59:00Z", "metric": "DCiE" } ]`			

				Metric Measurement
Description	This service allows for retrieving the metric measurements of a specific metric, based on their name			
Endpoint Name	`/api/vm_measurements/by-metric/{metric}`			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	metric	N	String	The name of the metric to fetch information for
Request body type	-			
Response class type	[GET] List<Metric_ measurement>			

Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager

Example	**Request:**GET /api/metric_measurements/by-metric/PUE/
	Response (application/json):
	[
	{
	"id": 1,
	"value": "1.60",
	"time": "2015-02-17T10:59:00Z",
	"metric": "PUE"
	}
	]

				Metric Measurement
Description	This service allows for retrieving the metric measurements of a specific metric that fall under a specific time range			
Endpoint Name	/api/metric_measurements/by-metric/{metric}/{start}/{end}			
Allowed methods	GET			
Request Parameters	**Parameter**	**Optional (Y/N)**	**Type**	**Description**
	metric	N	String	The name of the metric to fetch information for
	start	N	String	The start time of the time range, in ISO8601 format
	end	N	String	The end time of the time range, in ISO8601 format
Request body type	-			
Response class type	[GET] List<Metric_ measurement>			
Used by	ICT Performance and Energy Supervisor, DCO Hypervisor Manager, DCO Monitor/Collector, DCO Appliance Manager			

Example	**Request:**GET /api/metric_measurements/by-metric/PUE/2015-02-15/2015-02-18/
	Response (application/json):
	[
	{
	"id": 1,
	"value": "1.60",
	"time": "2015-02-17T10:59:00Z",
	"metric": "PUE"
	}
	]			

A 1.5.1 Class definition

Class: Measurement_type		
Description	This class represents a measurement type.	
Attribute name	**Attribute type**	**Description**
name	String	The type of the measurement
unit	String	The unit of the measurement type
description	String	An optional description of the measurement type

Class: VM		
Description	This class represents a virtual machine in a DC	
Attribute name	**Attribute type**	**Description**
name	String	The name of the VM
uuid	String	The UUID of the VM
flavour	Integer	The id of the flavour (combination of CPU, RAM, HDD) of the VM
server	String	The serial number of the physical server hosting the VM
status	String	The current status of the VM
task	String	The current action of the VM
instantiation_time	Datetime	The instantiation time of the VM, in ISO8601 format

Class: VM_measurement		
Description	This class represents a measurement of a VM.	
Attribute name	**Attribute type**	**Description**
value	Double	The measurement value
time	Datetime	The time of the record, in ISO8601 format
uuid	String	The UUID of the monitored VM
type	String	The name of the measurement type

Class: Server_measurement		
Description	This class represents a measurement of a server.	
Attribute name	**Attribute type**	**Description**
value	Double	The measurement value
time	Datetime	The time of the record, in ISO8601 format
serial_number	String	The serial number of the monitored server
type	String	The name of the measurement type

Class: Metric		
Description	This class represents a measurement of a server.	
Attribute name	**Attribute type**	**Description**
name	String	The abbreviated term of the metric name
description	String	An optional description of the metric

Class: Metric_measurement		
Description	This class represents a measurement of a metric.	
Attribute name	**Attribute type**	**Description**
value	Double	The measurement value
time	Datetime	The time of the record, in ISO8601 format
metric	String	The abbreviated term of the metric name