Project acronym: **CoolEmAll**

Project full title: **Platform for optimising the design and operation of modular configurable IT infrastructures and facilities with resource-efficient cooling**

D4.1 Architecture of the Module Operation Platform

Tomasz Piontek (PSNC)

Version: 1.0

Date: 29/03/2012

Deliverable Number:	D4.1
Contractual Date of Delivery:	31/03/2012
Actual Date of Delivery:	31/03/2012
Title of Deliverable:	Architecture of the Module Operation Platform
Dissemination Level:	Public
WP contributing to the Deliverable:	WP4
Author:	Tomasz Piontek (PSNC)
Co-Authors:	Micha vor dem Berge (Christmann) Leandro Cupertino (Christmann) Georges Da Costa (UPS) Piotr Grabowski (PSNC) Tomasz Kuczyński (PSNC) Bogdan Ludwiczak (PSNC) Juan Luis Prieto Martinez (ATOS) Ariel Oleksiak (PSNC) Wojciech Piątek (PSNC) Eugen Volk (USTUTT-HLRS)

History			
Version	Date	Author	Comments
0.1	10/02/2012	Tomasz Piontek	Structure of the deliverable
0.2	21/02/2012	Micha vor dem Berge Eugen Volk	Sections: 4.1, 4.2, 4.3
0.3	26/02/2012	Leandro Cupertino Georges Da Costa Juan Luis Prieto Martinez	Sections: 3.2, 3.3, 4.4.2
0.4	08/03/2012	Piotr Grabowski Tomasz Kuczyński Bogdan Ludwiczak Tomasz Piontek Ariel Oleksiak Wojciech Piątek	Chapters: 1, 2, 5 Sections: 3.1, 4.4, 4.4.1, 4.5
0.5	14/03/2012	Tomasz Piontek	Final draft for QM
0.6	27/03/2012	Tomasz Piontek Tomasz Kuczyński Juan Luis Prieto Martinez Ariel Oleksiak Eugen Volk	Modifications according to the Quality Control Feedback
1.0	29/03/2012	Tomasz Piontek	Final Version

Approval		
Date	Name	Signature
30/03/2012	Ariel Oleksiak	

Abstract

This document defines requirements and the general architecture of the Module Operation Platform (MOP) and its components. The key requirement for MOP is to deliver advanced software and hardware environment in order to conduct experiments needed by CoolEmAll validation scenarios, especially those requiring comparison of simulation models with measurements from real systems. Therefore, MOP should consist of all the software tools needed to monitor, control, and perform experiments on the CoolEmAll testbed.

The document includes a definition of main use cases that must be supported by MOP in order to conduct experiments. We also specify main possible settings of the testbed and workloads. Based on these use cases, a generic set of requirements is specified in this deliverable. The requirements are split up into four main classes concerning infrastructure monitoring and control, application monitoring, execution of workloads, and graphical user interface. On the basis of requirements and use cases we propose a general architecture and functionality of MOP.

Deliverable D4.1 should be taken as an input to those deliverables that refer to tests on real computing infrastructure at least within WP4, WP5 and WP6.

Keywords

Module Operation Platform (MOP), testbed, monitoring, HPC, Cloud

Acronym	Meaning
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
COVISE	COllaborative VIsualization and Simulation Environment
CPU	Central Processing Unit
CRAC	Computer Room Air Conditioner
DEBB	Datacenter Efficiency Building Block - defined in [D3.1]
GPI	Green Performance Indicator - defined in [D5.1rel]
GPU	Graphics processing unit
GSSIM	Grid Scheduling Simulator
GUI	Graphical User Interface
GWF	Grid Workload Format
HD	High Definition
HPC	High Performance Computing
HTTP	Hypertext Transfer Protocol
MOP	Module Operation Platform
OS	Operating System
OVF	Open Virtualization Format
PSNC	Poznan Supercomputing and Networking Center
RAM	Random Access Memory
RECS	Resource Efficient Cluster Server
REST	Representational State Transfer
RPC	Remote Procedure Call
SLM	Service Life-cycle Manager
SVD Toolkit	Simulation, Visualisation, and Decision Support Toolkit
SWF	Standard Workload Format
TIMaCS	Tools for Intelligent Management of very large Computing Systems
UC	Use Case
UI	User Interface
VM	Virtual Machine
VTK	The Visualization ToolKit

Table 1 Acronyms

Table of Contents

Table of Tables

Table of Figures

1 Introduction

Workpackage 4 is going to deliver two major results: the Module Operation Platform (MOP) and workload management policies. The former includes all the software tools needed to monitor, control, and perform experiments on the CoolEmAll testbed. The latter contains energy- and thermal-aware resource management and scheduling policies that can be inserted into simulations within the SVD Toolkit.

The main goal of this deliverable is to define requirements and the general architecture of MOP and its components. The key requirement for MOP is to enable conducting experiments needed by validation scenarios defined in Workpackage WP6, especially those requiring comparison of simulation models with measurements from real systems. To this end, we identify such scenarios (called verification scenarios in the deliverable) and based on them derive general requirements.

The structure of this deliverable is the following. In Section 2 we define main use cases that must be supported by MOP in order to conduct experiments. We also specify main possible settings of the testbed and executed workloads. Section 3 contains requirements derived from the verification scenarios. The requirements are split up into four classes concerning infrastructure monitoring and control, application monitoring, execution of workloads, and graphical user interface. In Section 4 we propose a general architecture of MOP based on requirements defined in Section 3. The section includes description of the planned hardware layer, monitoring tools, resource access and workload execution module, and graphical user interface. Section 5 contains conclusions and summarizes next steps.

2 Verification scenarios

The goal of CoolEmAll validation scenarios (to be defined by WP6) is to prove correctness and usability of project results: SVD Toolkit and ComputeBox blueprints. To this end, two classes of scenarios will be defined: *verification* and *demonstration* scenarios. The main goal of the former class will be to verify CoolEmAll models and simulation tools by comparison of their results with the data collected from the real infrastructure - CoolEmAll testbed. The latter class of scenarios will demonstrate how CoolEmAll models and simulation tools can be applied to analysis, planning and design of data centers with respect to their energy efficiency. The demonstration scenarios include all cases which cannot be tested in a real environment. The reasons of this inability include limited project budget (dedicated testbed cannot contain multiple racks or the whole data center), missing energy or temperature measurement data (lack of required sensors or monitoring infrastructure in a data center selected for experiments), nonexistence of the infrastructure (planning new data center), and time constraints (it is not possible to run hundreds or thousands of test scenarios that would be needed to identify best parameters for controlling mechanisms).

The main goal of the Module Operation Platform (MOP) is to allow project members to conduct experiments within the real settings and to compare results of these experiments to outcomes of modeling and simulation tools delivered by the project. Verification scenarios must cover all classes of these experiments and real settings. For this reason, requirements for the MOP design must be derived from the verification scenarios.

The verification scenarios, although limited in scope compared to demonstration scenarios, will be based on diverse settings in order to cover possibly largest set of various options that may be found in existing data centers. These settings will include various types of IT equipment, cooling, workloads, resource and workload management policies. Additionally, users will be able to use MOP in several main use cases. Main components of verification scenarios are illustrated in Figure 1 whereas general classes of use cases and settings are defined in Figures 2, 3, and 4. Both settings and use case are shortly described in the sections of this chapter. This classification and descriptions are based on initial work of WP6 and WP4 done till M6. More detailed description of scenarios will be included in D6.1 in M12.

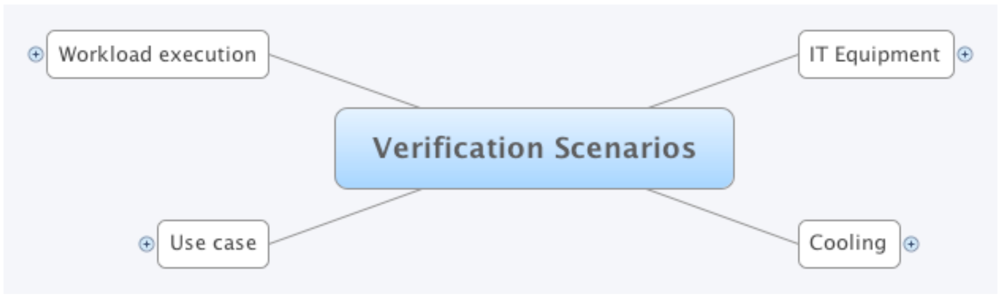

Figure 1 Verification scenarios

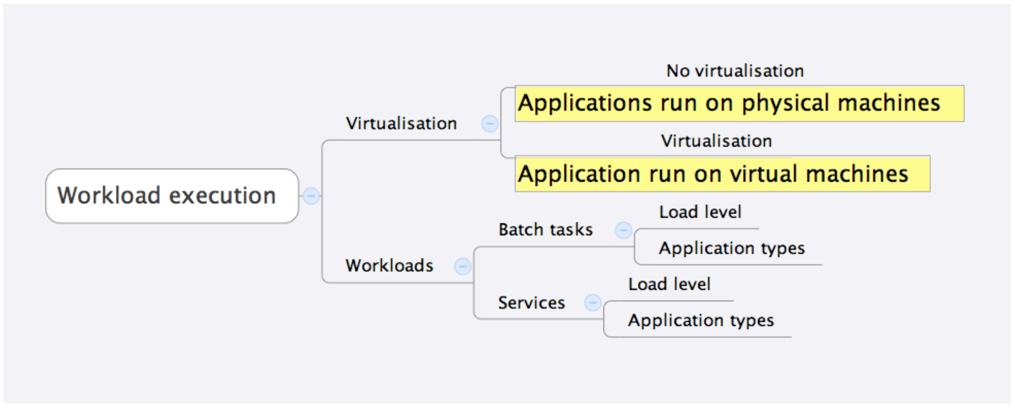

Figure 2 Workload execution types

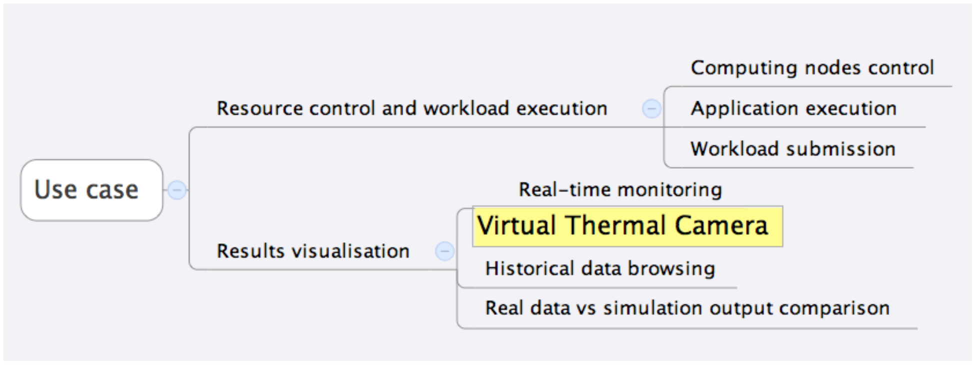

Figure 3 Use cases

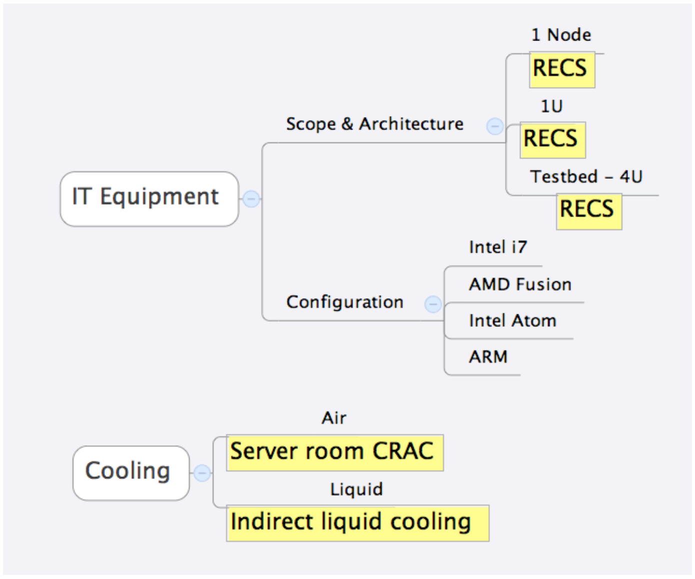

Figure 4 Hardware configurations

2.1 Scenario settings

Verification scenarios will include experiments with a wide set of settings. In this way, they will enable verification of various models of hardware, cooling techniques and applications defined within simulations tools. Details of the experiments will be chosen during the project lifetime depending on research directions and preliminary results. Precise definition of scenarios will be provided by WP6 while a specification of the testbed within WP3 (hardware) and WP6 (specific configurations). However, we summarize in this section general classes of settings that must be supported by MOP in order to conduct important types of experiments.

Users should be able to perform experiments with various *IT Equipment scope (SC):*

- SC1: 1 RECS node,
- SC2: 1 RECS unit,
- SC3: The CoolEmAll testbed - 4 RECS units (72 nodes or more depending on CPU types used).

Obviously experiments on a subset of the testbed will be possible too. In simulation and visualization tools parts of the IT equipment will be defined by Data centre Efficiency Building Blocks (DEBBs) – an abstraction for computing and storage hardware – defined in D3.1 [D3.1]. DEBBs will be used by MOP GUI to present hardware models consistently to visualisation of simulation models and results. However, in experiments concerning each IT Equipment scope DEBBs on various levels of granularity can be used depending on required precision of results. For instance, SC3 can be presented as 4 DEBBs (for each RECS unit), 72 DEBBs (for each node) or even as 1 DEBB (the whole testbed).

IT Equipment configuration (HW) will include at least four types of computing nodes:

- HW1: Intel i7,
- HW2: AMD Fusion,
- HW3: Intel Atom,
- HW4: ARM.

In this way, we will be able to build models and analyse impact on energy consumption and heat generation of diverse CPUs: from the powerful designed for HPC to very low power ones. Details of possible options are defined in D3.1 [D3.1].

Two distinct *Cooling techniques (COOL)* will be used:

- COOL1: air cooling (server room CRAC),

- COOL2: integrated cooling (indirect liquid-based).

Experiments with various input temperatures will be performed depending on constraints of the server room. Possible cooling solutions both for air and liquid cooling are presented in D3.1 [D3.1].

In order to address both HPC and cloud cases two main options related to *Virtualization (V)* will be provided:

- V1: No virtualization, applications run on physical machines,
- V2: Virtualization used, application executed on virtual machines.

Within scenario setting V2 a number of management policies will be applied to observe their impact on energy efficiency and temperatures.

Various *Workloads (W)* from two major distinct classes will be used:

- W1: batch tasks, typical workloads of queuing systems in HPC and data centers,
- W2: services that manage incoming requests, e.g. web servers, email servers, etc.

For each of these classes, various levels of load will be available by controlling job inter-arrival time, number of requests per second, etc.

Using the settings presented above, members of the consortium will be able to perform specific tests. An example of a combination of these settings is submitting batch tasks using virtualization to 1 air-cooled RECS Unit equipped with Intel i7 processors. Of course, possible hardware configuration will depend on the final specification of the CoolEmAll testbed so they are subject to further changes. Similarly, specific application benchmarks, workloads and policies will be developed within the project lifetime depending on scenarios defined by WP6 and specific research directions.

2.2 Scenario use cases

Similarly as in the case of scenario settings the detailed use cases will be defined in D6.1. Nevertheless, based on initial work of WP6 and discussions within WP4, this section contains a description of a way in which project consortium will be able to use MOP for experiments and tests. The proposed use cases are divided into 2 main groups:

- Uses cases related to execution of specific applications or full workloads on the testbed,
- Use cases related to testbed monitoring, controlling, and analysis of experimental results.

The main use cases related to execution of workloads on the testbed are as follows (UC1):

- **UC1.1: Computing nodes control.**

This use case will allow users to change states of resources, in particular P-states, C-states, and switching on/off. This use case will include selection of a node (or nodes), check and change of its state. Use case UC1 will be often used in combination with UC1.2 and UC1.3.

- **UC1.2: Execution of applications/benchmarks on selected computing nodes.**

This use case will enable users to execute selected real applications or benchmarks provided by WP5 on selected computing nodes. Depending on the workload execution settings application can run directly on physical machine or using virtualization. The goal of this use case will be usually to test the impact of specific applications on temperature and energy consumption. Based on these tests, appropriate benchmarks as well as models within simulation environment will be developed.

- **UC1.3: Submission of a workload to resource management system.**

This use case will allow users to execute a set of applications either generated synthetically or taken from real systems in use. This workload can consist of batch jobs or services (see 4.4.2) depending on the scenario settings. Similarly as in UC1.2 workload can be executed directly on physical machines (using a queuing system) or within virtual machines (using virtualization management software). This use case will enable repeating the whole experiment in the simulation and real infrastructure. In this way, CoolEmAll simulation tools will be verified and researchers will get more insight with regard to specific management policies and models of applications and hardware.

Three main use cases related to visualization and analysis of results has been identified (UC2):

- **UC2.1: Virtual Thermal Camera (Real-time monitoring).**

In this use case, an end user will be able to visualize the current state of all entities in the CoolEmAll testbed including compute nodes, RECS base boards, RECS back planes, RECS micro-controllers, storage/network subsystems, racks and any other monitored units. While all entities will be visualized in the 3D form, an end user will be able to take "a virtual walk" around the testbed, pick an entity and explore entity's detailed metrics. Although main information exposed in the Virtual Thermal Camera use case will be current temperature of entities, an end user will be presented with other metrics which can be obtained from the monitoring system - metrics listed in the Infrastructure monitoring and control section (3.1).

Functionality that will be accessible in this use case is not limited to choosing of an entity and exploring its metrics, but includes also possibility of interaction with monitored entities. List of commands is also provided within Section 3.1.

Virtual Thermal Camera will also present a basic information about applications such as names and nodes they are running on to the end user (visualisation of other application metrics via GUI can be considered within project lifetime but is not mandatory).

- **UC2.2: Historical data browsing.**

In the historical data browsing use case, an end user will be enabled with the same functionality as the Virtual Thermal Camera, with except that, in this case data will not be real-time, but historical. An end user will be able to specify which point in time is to be visualized.

- **UC2.3: Real data vs. simulation output comparison.**

Real data vs. simulation output comparison will enable a user to compare historical data with an output of the simulation. In this use case, an end user will be presented with two (or more) "windows" displaying the same 3D scene, from the same point of view, and placed side by side. Control for both "windows" as well as point in time should be shared, but displayed content should reflect real data in one "window" and outcome of simulation in the other one.

In the subsequent sections of this deliverable we will refer to the scenario settings and use cases proposed above in order to define requirements for MOP. Precise specifications of verification scenarios along with detailed settings will be defined in D6.1.

3 Requirements and functionality

The requirements and functionalities of the Module Operation Platform (MOP) defined in CoolEmAll focuses in two main areas which are also the two defined blocks of the platform. These areas are the monitoring system and the resources and workload management system implemented during the project. The monitoring system provides a collection of small monitors to gather information from the physical or virtual resources, and applications. Once collected, the infrastructure monitoring and control system will aggregate and centralize this information and will expose it the rest of the MOP components can query and use that information.

The second block of requirements refers to resource- and workload-management capabilities of the Module Operation Platform, this is the one in charge of the managing the experiments on the project testbed. To this end, it will enable executing workloads and managing resources including setting the nodes in the

Compute Boxes in different performance levels, starting, suspending or switching off nodes depending on their workload, and submitting workloads to resource management systems (such as queuing systems in HPC case or virtual management tools in a cloud case).

The following sections define requirements for the four main aspects of MOP: infrastructure monitoring and control, application monitoring, running workloads, and graphical user interface.

3.1 Infrastructure monitoring and management

The project requirements concerning monitoring and management infrastructure tools and services are mainly determined by needs of validation scenarios [D6.1]. Some of the requirements for monitoring and management also come from technical constraints and need for easy use of MOP user interface and visualization tools. These tools are going to be used to visualize and analyze the current and historical power efficiency of the testbed and to compare the simulated virtual server room with the actual behavior of the real-life testbed. The validation scenarios planed in the project as well as needs for application and hardware modeling impose mainly requirements regarding available metrics. UI and visualization tools have additional requirements regarding functionality and technical aspects of the monitoring and controlling system for providing advanced tools of data analysis, testbed management and data visualization (e.g. virtual thermal camera).

The list of monitoring functionality presented below considers the designed capabilities of the CoolEmAll UI and management tools and the needs of validation scenarios. These tools need access to both current and historical power-efficiency status of the testbed. They may need to query for all or specific metrics values for that whole testbed or only for its given parts. In order to visualize on the fly the current status of the testbed, there must be a way of notifying UI tool of the vital testbed events including metric value changes and emergency/alarm occurrences. The required metrics include various hardware status parameters, software (operating system and job queuing system) and basic parameters for running applications like application to node mapping, application performance and running time. Metrics that should be provided by the monitoring system should include metrics defined in [D5.1rel] and any potential new metrics proposed in [D5.1]. Some of these metrics are low-level power-efficiency metrics or testbed status metrics introduced here for the needs of project's visualization and data analysis tools whereas others are more complex metrics such as Green Performance Indicators (GPIs) also defined in [D5.1rel].

Required functions

This paragraph lists main functional requirements of the MOP infrastructure monitoring and control layer.

- List units – provide list of all entities in the testbed including compute nodes, RECS base boards, RECS back planes, RECS micro-controllers, storage/network subsystems, racks and any other monitored units,
- Get units' hierarchy – similar as above, provide list of all entities but in way which enables recreation of the actual unit hierarchy. Actually, this function can be merged with the previous one,
- List units by unit type or location (server room, rack, server type/architecture) – list entities of the given type or location. The location can reflect the actual physical location of an entity in the server room (e.g. rack, rack row or RECS name, number or coordinates). The type should reflect the defined DEBB types [DEBBDef], but it may also regard other elements of the testbed including storage and network nodes and monitoring infrastructure units like RECS micro-controllers. This functionality is needed to visualize the testbed on various levels of details, e.g. different units' metrics are required to show the whole server room or server rack or only singe RECS unit,
- List all available metrics and list available metrics for particular unit type– provide list of all monitored metrics (final list will be defined within D5.1 but at least power usage of all nodes, temperature of nodes, inlet/outlet temperature, and nodes load should be available),
- List hosts which provides given metrics – list units which provide given metric.
- Get last [all/selected]metric values – this is the core functionality needed to show the current state of the testbed – the UI/visualization tools need this function to pull for the all or selected metrics values at the beginning of visualization session. Later metrics value changes should be provided by notification mechanism,
- Get historical [all/selected]metric values by location (server room, rack, server type/architecture) – functionality similar as above function but for historical data. The monitoring system should provide means for storing and accessing historical metrics values. The system should gather and store all available metrics for all monitored units by default. This is the key functionality for the visualization and analysis of past status of the testbed and thus for the validation of project simulation and application scenarios. Metrics values should be accessible for the whole testbed and for selected location to visualize the testbed at various levels of details,
- Get average in given period [all/selected]metric values – for the whole testbed
- Get average in given period[all/selected]metric values by location (server room, rack, server type/architecture) – metrics average values in given

period of time are also vital for the visualization and validation tools. Averaging values of the metrics which tend to change very quickly in wide range may provide better user understanding of the visualization than attempting to display rapid changes every fraction of a second,

- Send metric value/host status changed notification
- Send emergency condition/alert notification – a mechanism of notifying of the metrics value changes. After initial acquisition of the testbed status by pulling monitoring system, the monitoring/UI tools should be notified by the system about all vital changes in the testbed status. There should be also an ability of selecting/masking(or in other words disabling notifications regarding given metrics) metrics and or units whose status changes should be notified,
- Get metric availability by location/host type – a function to check if given metric is available at certain testbed location (node/RECS unit/rack/...) or at given host or hosts or by given node type,
- Get metric update frequency [by host/location], Get historical and average metric data ranges[by host/location] – an informative functions to provide UI/visualization tools with the knowledge of what can possibly be presented to the user,
- Change component state – power on/off, suspend, hibernate given component or group of components. The component can be a part of a computing node (cpu, network card, etc.), the whole node, RECS unit or rack,
- Change cpu dynamic frequency state – dynamically the compute node's CPU[s] throttling,
- Manage virtual machines: creating VM on the specific host, shut downing VM, migrating VM between the given hosts, suspending, restarting, saving and resuming VM.

Unit types and locations – as implicitly mention in the description of required functionality, we assume that the whole monitored testbed has a hierarchical structure. This structure may be build up by combination of different types of monitored units, their location and mutual relations between units. A unit may be any separable and monitorable testbed host that performs vital server room function like running application and providing network connectivity or storage capacity, but the unit as defined by this section may also be a group of hosts or other units. In general, available unit types should reflect all DEBB types as defined in [D3.1, but to visualize the coherent testbed status also other unit types are required. These additional unit types include at least: network and storage hosts and RECS and other micro-controllers. We assume that, at the physical level the whole server room configuration may be reflected as combination of various DEBBs with addition of aforementioned other units. I.e. the server room consists of various configurations of rows (ComputeBox2, see [D3.1]) of server racks (ComputeBox1, see [D3.1]). A rack may be a regular rack without any extraordinary monitoring functionality (Node Units or Node Groups, see [D3.1]) or RECS housing unit (Node Units organized into Node Groups, see [D3.1]) with sophisticated CoolEmAll monitoring and management unit. A single rack consists of server nodes which may be compute (Node Unit, see [D3.1]), network or storage node (storage and network nodes are not covered by defined DEBB types). A single compute (Node Unit) node may by a physical node running applications' processes that is able to provide all hardware monitoring metrics. It can be also virtual node deployed on physical node and running only applications' processes. Virtual nodes do not provide power-efficiency metrics but their presence might be required to properly visualize the actual application to node mapping.

Required Metrics

This paragraph contains the initial list of metrics required to implement planed MOP functionality. Different nature of various types of testbed units makes that some metrics are only measurable for given type while others are universal and possibly available at every node.

Simple metrics - metrics listed below are simple hardware metrics generated at various components of the server room equipment. They are basic metrics that are considered as required to visualize current load and power consumption of the testbed. This set will be extended by deliverables [D5.1rel] and [D5.1].

- Component state (power on/off, suspended, hibernated, broken, inactive) – available at every unit type,
- Host load (CPU usage, Memory usage I/O device usage, Storage usage - as defined by [D5.1rel]) – available at compute/network/storage nodes,
- Network load/usage,
- CPU dynamic frequency scaling – current state and possible range – available at compute nodes,
- Component temperatures (various host thermometers, chassis, ambient, cpu, mem, chipset) – available at every unit type,
- Component power consumption (by node, by cpu) – available at every unit type,
- Unit fan speeds (node, RECS) – available at every unit type,
- Unit (RECS, rack) inlet/outlet air temperature – available at RECS unit inlet/outlet.

Complex metrics - these metrics will be defined in [D5.1rel] and [D5.1] and thus they are not listed or redefined here. For the full definition and reference please see the aforementioned documents. In general, these metric are built as an appropriate composition of various simple metrics. They do not allow to observe detailed power status of the testbed, but they are meant to measure its overall power efficiency. Thus, they are required to visualize the actual power-efficiency status of the testbed. In general, to meet project's requirements, we need access to energy impact GPIs and aggregated values for GPIs (see [D5.1rel]).

Nonfunctional requirements

There are also nonfunctional requirements to the MOP monitoring and control infrastructure, which include:

Data availability – required minimum metric update frequency, available historical data range and availability of average values. The range of historical data availability is determined by project validation scenarios and it should be at least as long as the longest possible scenario run time[D6.1]. The granularity of average data availability depends on metric tendency to change rapidly in wide range and the maximum pace of value change which can be visualized in a readable way. In general, we predict that historical date should be available for at least 1 month with average values available for number of seconds specified within WP5.

Monitoring system interface – well defined API or access protocol technical requirements (access method RPC or REST service). The UI/visualization/management tools are going to communicate with monitoring system remotely. These tools will be implemented in one of the following programing languages: C++, Java, Python. Thus, the monitoring system has to have accompanying API libraries implementing remote access to aforementioned functionality with bindings for these languages or it has to expose well defined remote interface based on standardize paradigm like RPC or REST. Detailed technical documentation of the system API or remote interface is required to integrate UI, visualization and management tools.

3.2 Monitoring of applications

Monitoring of applications is concerned with two distinct fields: HPC and Cloud. In HPC, applications are composed of several MPI processes distributed in one or more compute nodes (hosts), these nodes can execute more than one application process at the same time. While for Cloud, applications tend to be executed in the same virtual machine (VM) but one physical machine can have more than one VM running at once. So the purpose of the monitoring of application is to keep track of metrics of all child processes of each application and aggregate them in order to obtain the metrics for the applications. These application metrics will support the decision center.

The project requirements according to monitoring of applications are mainly determined by needs of the simulation and validation scenarios, as inputs, and the decision center as a way to obtain accurate information on application state to take informed decisions. Various user interface and visualization tools have needs comparable to the decision center. As for monitoring of infrastructure, the simulation and validation scenarios impose mainly metrics, while decision center and interaction systems add functional requirements. The possibility to put in place alarms is important, in enables the determination of crashes in real time and the revaluation of the system state, for instance.

Required metrics include various system values at different level. Depending on the system, virtual machines might be used leading to a two level hierarchy: Applications runs on virtual machines, themselves running on hardware (host). Metrics can be of two types, direct or indirect. Direct metrics are directly obtained from the operating system (such as memory consumption). Indirect metrics are computed using several other metrics (such as power consumption of an application).

Applications and Virtual Machines

- An Application is a set of coherent processes achieving a particular task. An application can spread on several hosts. In this case we consider that each part of an application will communicate with others in a connected graph. The processes related to an application can run directly on nodes or in virtual machines.
- Virtual machines can execute processes related to one or several applications. The host can execute at the same time several virtual machines.

Required functions

- Metric's related functions
  - List of available metrics: all available metrics.
  - Last metric: get last [all/selected] metric values for the whole application/VM.

- Average metric: get average in given period of [all/selected] metric values for the whole application/VM.
- Metric's history: get historical in given period of [all/selected] metric values for the whole application/VM.

- Application's related functions
  - List of applications: all applications running on the HPC/Cloud environment.
  - List of applications by location: all applications running in a host or VM.
  - List of application's dependency: data, communication, dependency style (producer/consumer) and other applications dependencies.
  - List of application's requirements: library, architecture, number of hosts requirements.
  - List of locations of an application: list of the hosts/vm that runs a particular application
  - State of application: get the state of an application (running, finished, crashed, VM/nodes list).
  - Starting/ending time: get the starting/ending time of an application.

- Virtual machine's related functions
  - List of virtual machines: all VMs running on the Cloud environment.
  - List of virtual machines by location: all VMs running on a specific host.
  - Virtual machine's static characteristics: get VM's characteristics (limits on CPU/memory, technology, host location).
  - State of virtual machines: get the state of a VM (running, finished, crashed, in migration).

Nonfunctional requirements

- Nonfunctional requirements for application monitoring are identical to the one of infrastructure monitoring.

Required Metrics

- Application/VM load on resources: CPU load, Memory allocated, Network [inbound/outbound] bandwidth [bytes/packets], I/O [read/write] [byte/number of operation]

- Application/VM performance counters: all the default performance counters available on the architecture (9 hardware counters available on Linux, like Cache misses or number of floating point operation, the list is available in the kernel tree in *tools/perf/design.txt*)
- Application/VM communication: bandwidth, [send/receive] destination.
- Application/VM power consumption: value, model.
- Generic numerical metrics from sensors available on the host (using the standard Ganglia or Nagios interface)

3.3 Running workloads

An important feature of the Module Operation Platform is the possibility of execution of different workloads of applications according to validation scenarios that will be defined by WP6. These workloads would be specific for each scenario, as the CoolEmAll is targeting two main classes of scenarios, namely HPC and Cloud which are the two mostly used and growing data center usage scenarios [DC Investments]. HPC workloads usually consist of long lived and CPU bounded applications involving commonly data-intensive computations. They are usually batch multi-processor jobs submitted to queuing systems and executed using Message Passing Interface (MPI). On the other hand, cloud workloads refer mostly to the service requests that need to be handled within Service Level Agreement. Hence, the Module Operation Platform has to provide a set of tools that are able to run and manage both types of workloads. Since the workloads affect the physical layer they will be specified, together with the controlling infrastructure tools, for each validation test.

To distinguish between these two major classes of scenarios we will refer to workloads of batch jobs used in HPC as *HPC workloads*, or *computing workloads*, whereas to workloads used in cloud scenarios, as *cloud workloads*, or *service workloads*.

3.3.1 Cloud workloads

Cloud environment is one of the scenarios to be tested within the scope of the project. Hence, cloud services, defined as a set of virtual machines that work together to perform Cloud applications (for more detailed explanation see Section 4.4), will be executed in the CoolEmAll testbed. At this level, the MOP should provide the tools for deploying and executing services. The cloud workload to be executed will be described by the OVF file [OVF]. This file describes all the virtual architecture of the service plus the elasticity rules that are used to spin up or shut down a virtual image according to the defined policies and rules.

The virtual system manager needed for running the services is the layer which directly talks to the host machine and places the virtual images according to the given policy. Hence, the MOP should enable configuration of energy aware scheduling policies in order to tell the virtual system manager to consolidate or disperse the virtual machines among the ComputeBox infrastructure. The policies should take into account consumed energy or produced heat and minimize these factors by placement of virtual machines and by management of pool of physical machines.

3.3.2 HPC workloads

MOP also has to provide a system for HPC workload execution. However, allowing ordinary job submission is not the only requirement. Conducting full-scale experiments on real hardware allows evaluating the proposed models and adjusting the simulation environment. However, they are usually expensive and time-consuming. Hence, it is important to be able to reconstruct and repeat experiments from simulation environments on real hardware in a reliable and effortless manner. Thereby, the Module Operation Platform should provide a framework that will enable replaying workloads running on the testbed as well as those studied within the SVD Toolkit. Migration of experiments between simulator and real system will allow performing a comprehensive analysis including all the complexities of the investigated environments. Moreover, it will enable evaluation and adjustment of the proposed workload management policies in an efficient way.

Thus, another requirement related to HPC workloads refers to the possibility of executing synthetic workloads on the evaluated ComputeBox infrastructure. To this end, the Module Operation Platform should allow generating a set of jobs, basing on a given model and then submitting them to the actual system. This functionality could be achieved twofold: workload can be generated with respect to the characteristics provided by user or it may be based on the traces used in the workload simulation tool (also those delivered from various production environments). To meet the former objective the MOP tool should introduce the configuration file structure that will enable defining all necessary parameters needed to create the synthetic workload. The latter goal requires delivering of parser that will be able to read a trace file in one of the given formats and recreate the appropriate set of jobs. Moreover, it is desirable to enable gathering traces from the system in order to ensure feedback with simulation environment.

As regards HPC workloads formats from large-scale systems, the most commonly used and supported by various workload simulation tools file format is Standard Workload Format (SWF) [SWF]. SWF was defined in order to facilitate the use of workload logs and models. It provides a standardized format for independent jobs and chains from various large-scale parallel systems. Each workload is stored in a single ASCII file that is relatively simple and easy to parse. SWF logs provide basic information concerning job executions, such as:

job number, submit time, wait time, run time, requested time, number of allocated processors and used memory among others.

3.3.3 Summary of workload management functionality

The following list contains the required functionality with respect to the execution of workloads both for HPC and cloud scenarios:

- Possibility to run certain tasks on a specific host (according to use case UC1.1). Users should be able to submit and manage software applications with ability to point compute node or nodes where the applications should be started. Nodes should be selectable by their type or location.
- Possibility to migrate certain tasks from and to a specific host – migrate running application to different node or set of nodes. Nodes should be selectable by their type or location,
- Collecting and archiving trace logs,
- Defining the HPC workload characteristics (in particular number of jobs, job length, job submission time, number of requested resources such as processors and memory),
- Extracting main characteristics of HPC workloads from SWF files,
- Synthetic HPC workload generation based on the given workload characteristics
- Managing services, in particular defining the elasticity rules for services (and loading them from the OVF [OVF] file),
- Cloud service management: The MOP has to provide tools to manage the life-cycle of a service running on a cloud environment over a virtual platform manager system,
- Availability to set cloud workload levels to run the application with various load levels (e.g. very low, low, medium, high, very high),
- Capability to configure the virtual platform manager system to apply energy and thermal aware policies.

3.4 Requirements for Module Operation Platform GUI

Module Operation Platform should include graphical user interfaces in order to facilitate access to the project testbed. This access should be remote so that all partners could easily use the testbed and administrators monitor if everything is correct. The main goal of the MOP GUI is on one hand to display state of the testbed and, on the other hand, to visualize results of experiments. For this reason the MOP graphical user interfaces should include web access enriched with advanced visualization.

Functional requirements

Graphical User Interface for Module Operation Platform has to provide the functionality listed below.

MOP GUI should provide remote access to the testbed monitoring and control. This interface should be enhanced with advanced 3D visualization for compute box and server room. In this way, it will be easier for a user to understand processes taking place within the testbed and to compare results from real infrastructure with those from simulation tools. MOP GUI should be capable of displaying the following general information:

- 3D view of the compute box and server room (based on descriptions obtained from WP2 - described in [D3.1]). This functionality is needed in use cases UC2.1-3 in order to meet Infrastructure Monitoring and Control and Application Monitoring requirements.
- All monitoring information concerning infrastructure including 2D charts showing each kind of metrics obtained from TIMaCS. This functionality is needed in use cases UC2.1-3 in order to meet Monitoring Infrastructure and Applications requirements.
- Simulation results in 3D view with air temperatures and velocity markups. This functionality is needed in use case UC2.3 in order to meet Infrastructure Monitoring requirements.
- Interface for compute box management and control allowing to forward commands to TIMaCS (via RPC or REST service). This functionality is needed in use case UC2.1 (virtual thermal camera) in order to meet Infrastructure Control requirements.

The requirements concerning detailed data that should be visualized in the cases listed above are as follows:

- Displaying current testbed state,
- Displaying historical testbed state,
- Displaying comparison between measured states and simulated one,
- Choosing which infrastructure element is visualized,
- Choosing actual metric (from the list of supported metrics - see Section 3.1),
- Displaying charts with current data (in real time) and historical state for given set of metrics,
- Overlaying charts for different metrics and sources.

Nonfunctional requirements

The nonfunctional requirements include the requirements concerning technologies, interfaces, and data formats that should be applied for MOP GUI:

- The MOP GUI should seamlessly integrate web interfaces with advanced 3D visualization. The former will enable users to see text information, charts and to insert data in an easy and standard-based way using web browsers. The latter will provide intuitive visualization of the server room and experiment results.
- All GUI tools should be at least inter-operable, possibly accessible via common interface (for better user experience).
- GUI components provided by WP4 should be reusable by other workpackages (if feasible) to avoid redundant implementation works (e.g. formats to modules should be identical to those used in the SVD Toolkit).

Out of scope

Since there are no requirements for exposition of monitoring of applications data via GUI, only limited set of information about applications like e.g. name will be presented to the end user (exposition of other application metrics via GUI is not mandatory), The same applies to interactions, so functionalities like application submit, cancel, migrate etc. will not be implemented in the GUI. The same applies to running workloads - there is no GUI requirement for this functionality and there will be no support provided for this case. The main reason is a variety of scenarios and systems to be potentially used in experiments. As development of GUI for all of them would require large effort, command line tools will be used instead, which should be sufficient for experienced project partners. Efforts will be concentrated on GUI for analysis and comparison of experimental results as well as basic tools common for most scenarios and systems that can be identified within the project lifetime.

4 Architecture

The Module Operation Platform (MOP) is organized in several layers, each addressing requirements identified in previous sections, as shown in the figure below.

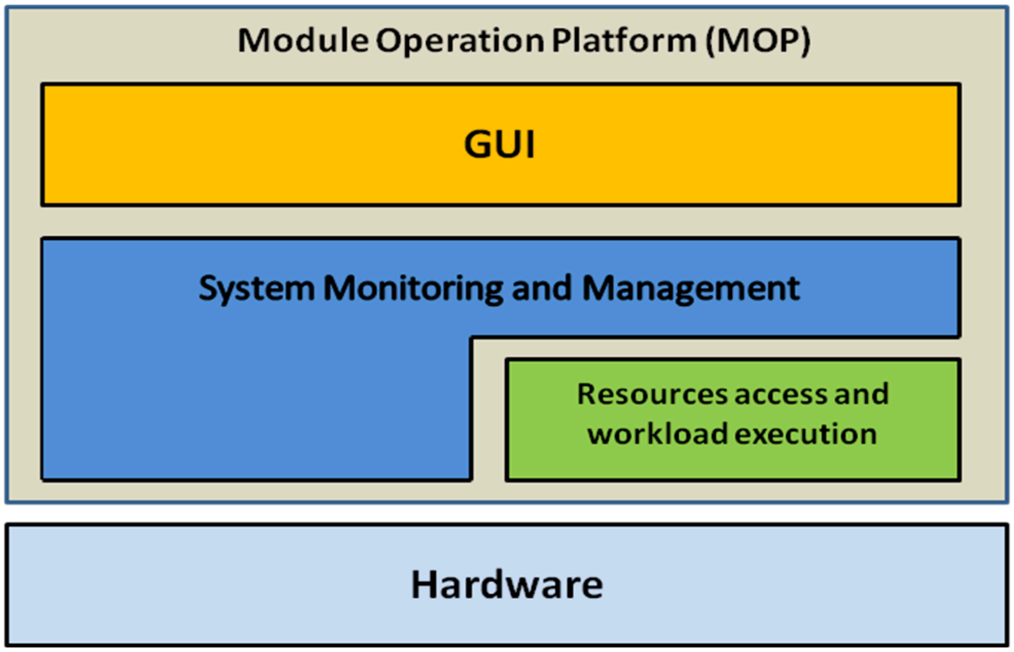
Module Operation Platform (MOP)
GUI
System Monitoring and Management
Resources access and workload execution
Hardware

Figure 5 Module Operation Platform - Layers

The hardware layer represents computing resources that are monitored, controlled and managed by module operation platform. Computing resources as used in CoolEmAll, provide integrated monitoring and control interfaces, ready to be used by “System Monitoring and Management” and "Resources access and workload execution" layers of the MOP. The “System Monitoring and Management” layer is responsible for monitoring and management of the hardware layer, capable to collect and aggregate monitoring data not only from the physical or virtualized computing resources, but also from the applications and workloads. The "Resources access and workload execution" layer is in charge of the workload policies and the management tools for the hardware layer. It is capable not only to set or change operational parameters of the physical hardware, but also provides mechanisms to control and manage the validation application and the workloads associated. The GUI layer delivers graphical user interfaces to the functionality of MOP, in particular monitoring and control of the testbed, and comparing of real data with models used in simulations.

The following subsections provide detailed description and architecture of the Module Operation Platform layers, describing in detail the hardware layers of the compute-box in Section 4.1, monitoring tools in Section 4.2, the TIMaCS system monitoring and management architecture in Section 4.3. Section 4.4 describes resource access and workload execution environment, allowing defining, simulating and executing workload policies. Finally, Section 4.5 provides description of GUI, allowing retrieving and showing historical data stored in TIMaCS database using graphical user interface.

4.1 Hardware layer (RECS)

The RECS Cluster System is an 18 node computer system that has an monitoring and controlling mechanism integrated as described in CoolEmAll D3.1. Through the integrated novel monitoring approach of the RECS Cluster System the network load can be reduced, the dependency of polling every single compute node at operation system layer can be avoided. Furthermore this concept build up a basis on which new monitoring- and controlling-concepts, like in this Deliverable described, can be developed. Therefore, each compute node of the RECS Cluster Server is connected to an Operation System independent microcontroller that collects the most important sensor data like temperature, power consumption and the status (on/off) from every single node. Theoretically it is also possible to gather information from the Operation System layer by sending these information over the SM-Bus to the microcontrollers, but this approach has not yet been fully developed. The main advantages of this new monitoring and controlling approach are on the one hand that it enables us to collect various metrics that are usually not available (e.g. power consumption of each single cluster node) without inducing high load on the network and the monitoring machine. On the other hand we can now control the system on a direct hardware layer by switching nodes on/off via a central microcontroller.

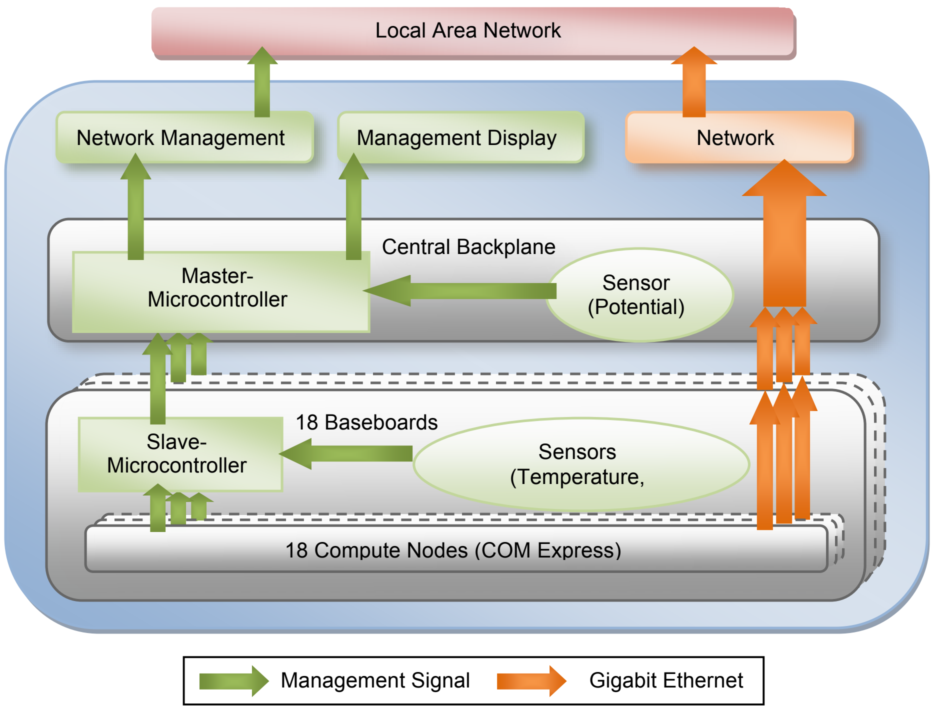

Figure 6 Architecture of the Master-Slave Microcontroller Monitoring System

This microcontroller-based monitoring architecture is accessible to the user by a dedicated network port and has to be read out only once to get all information about the installed computing nodes. If a user monitors e.g. 10 metrics on all 18 nodes, he would have to perform 180 pulls which can now be reduced to only one. This example shows the immense capabilities of a dedicated, aggregating monitoring architecture. Since the master microcontroller collects all data before sending it to the user, it is able to do a sensible pre-processing of the gained data. In the current testbed, only a simple pre-processing is executed, namely the potentials and currents used by the mainboards are multiplied to get the power consumption. Theoretically much more pre-processing is possible and it has to be evaluated which type of pre-processing or even controlling is useful. Another benefit is that collecting the IT resource usage is independent from the operating system and thus causes no payload on the compute nodes itself which means a maximization of the available computing power for the user.

It can’t be neglected that the microcontrollers also consume additional energy which has to be measured and compared to the advantages that are gained through the microcontrollers.

Input Data	Data Source	Unit
Status of the Mainboard	RECS MicroController	On/Off
Network Link Present	RECS MicroController	Yes/No
Network Speed	RECS MicroController	10/100/1000 MBit/s
Network Link Active	RECS MicroController	Yes/No
Fan Rotational Speed	Mainboard-Sensor	Rotations Per Second (rpm)
Temperature of the Mainboard	Sensor CMFB4000104JNT, RECS MicroController	°C
Temperature of the CPU	RECS MicroController	°C
Current used by the Mainboard	Sensor ACS715ELCTR-20A-T ,RECS MicroController	Ampere
Voltage of the Power Supply Unit	ATMEGA169P-16AU ADU, RECS MicroController	Volt
Power consumption of the Mainboard	RECS MicroController	Watt
Potential on the Mainboard	Mainboard-Sensors	Volt

Table 2 Sensor data that can theoretically be quantified by the RECS Cluster Server

The sensor data that can theoretically be quantified by the RECS Cluster System is described in **Error! Reference source not found.**. Which of these information can be measured on a real system depends on implementation details of the microcontrollers and support of the mainboards and will be evaluated and specified. As described above, besides the monitoring capabilities of the RECS it is also possible to control the compute nodes in two ways. The first and high-level way is to manage the nodes on the Operation System layer. Here we have the possibility to bring the system to different performance states (so called P-States), to hibernate or to shut down the node. To wake up the system we have a second, low-level way via the microcontrollers. The following energy states are generally possible for every single compute node, where again it has to be clarified whether they work with the specific configuration:

- On, Maximum Performance

  Maximum CPU frequency, no CPU throttling, Operation System scheduler at maximum power state
- On, Low Performance

  Minimum CPU frequency, CPU throttling, Operation System scheduler at energy saving state
- Sleeping/Hibernate

  CPU off, RAM in low power state
- Off

  Completely switched off, turn on via the MicroController

4.2 Monitoring tools

There are a wide variety of tools to monitor a data centre on the market available. Many of them are proprietary and only usable within a pre-defined, often vendor-specific (server-) environment and not extendable. On the other side there are also a lot of open and extendable monitoring tools, some of them are even available with their source-code which enables the user not only to extend them via plugins but also to adjust the tool to its specific needs.

We analyzed our needs and compared several tools, to check which of them meets our requirements best. Here is a short overview of needs that were necessary for us:

- Extendable through the use of self-developed plugins
- Open source to be able to customize and integrate only parts of the tool into our CoolEmAll project
- Scalable to thousands of nodes
- Standardized database

- Data aggregation possibilities
- External request API for different granularity levels (to get e.g. a dataset for an overview about the data centre with a single request)
- Analyzing framework that can be adjusted to our needs
- Alert mechanisms

To determine which monitoring tools fits best these requirements, we reviewed existing comparisons that list the available features of the monitoring tools and extended them by own expertise, see [GAMES D2.1] and [Wiki]. We found out that a layered monitoring infrastructure would be needed because none of the tools fulfilled all requirements. The first layer will provide an efficient collection of metrics whereas the second layer will care of the high level functions like aggregation, interpretation and providing standardized access to the data set. A further reason to use a second layer of monitoring is scalability and flexible/simple aggregation of nodes from various systems levels (RECS, Computebox1, ComputeBox2) using public/subscribe systems.

For the low-level collection of metrics we chose Nagios (see [Nagios]) as it provides a very good basis to set upon. The main reasons to use Nagios is that it is well known which means that many of the measurable metrics can be already measured via existing plugins from previous projects or from the lively Nagios community. Furthermore it provides a well-tested and efficient way of gathering metrics at a low level for up to 1000 systems. Nagios is even able to do a low-level analyzing of the measured data and to report/alert if thresholds have been exceeded, but these analyzing and reporting mechanisms are not sophisticated.

Therefore we decided to build up an architecture with multiple aggregation paths as described in the section "TiMaCS" below. At the bottom of this aggregation path we will find either physical sensors that measure metrics like power consumption or virtual sensors that measure application metrics like "requests per second".

4.3 System Monitoring and Management Layer

The “System Monitoring and Management” layer responsible for monitoring, data-history building and controlling of the rack level compute-box, is based on the architecture of the TIMaCS framework developed in scope of national project [TIMaCS], described in [Volk]. The TIMaCS framework is designed as a policy based monitoring and management framework with an open architecture and a hierarchical structure. The hierarchy of the TIMaCS framework is formed by management layers acting on different levels of information aggregation and abstraction. This is achieved by generating state information or aggregating metrics for groups of different granularity. Within CoolEmAll these granularities are: Node Unit, Node Group, ComputeBox1, ComputeBox2, as defined in CoolEmAll DEBB concept. These granularities form abstraction layers. A possible realization of the hierarchies can be achieved in a tree-like structure, as shown in the figure bellow.

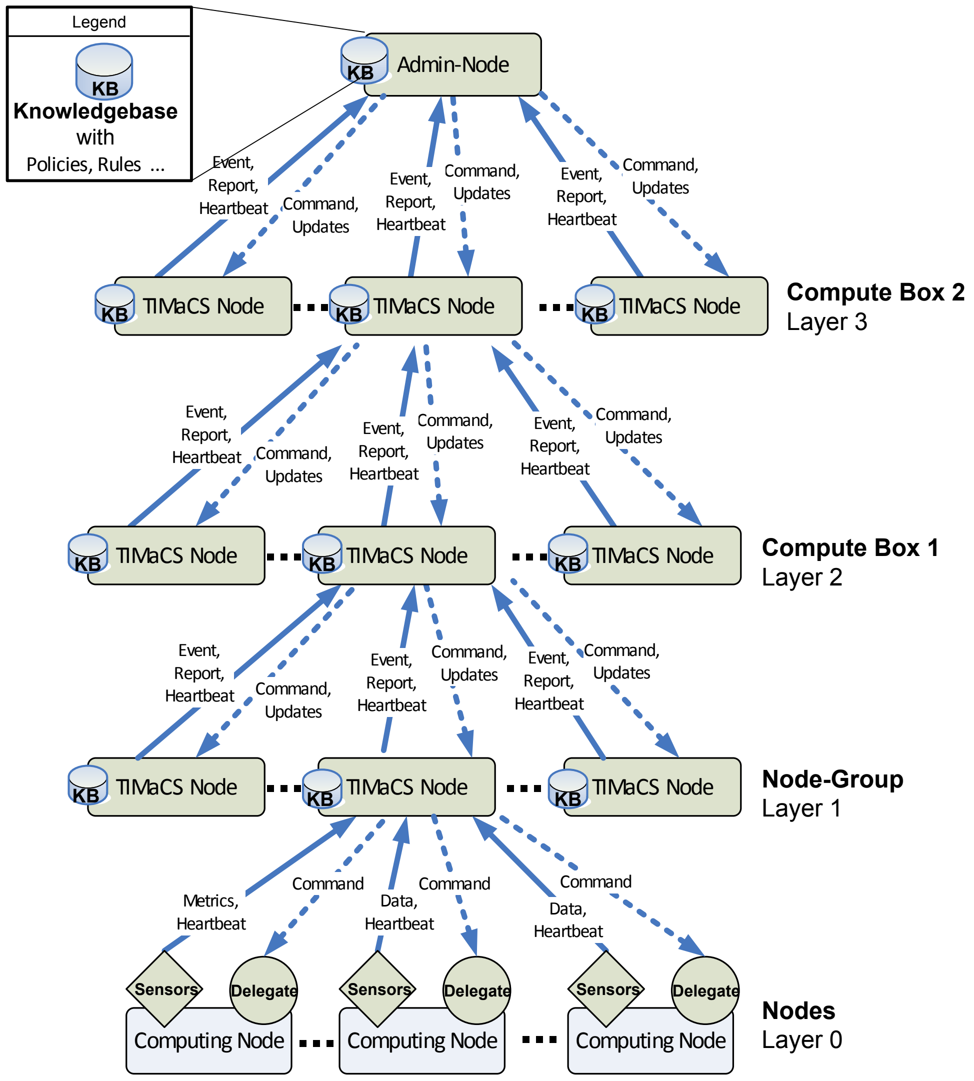

Figure 7 CoolEmAll/TIMaCS Hierarchies

The bottom layer of the CoolEmAll/TIMaCS hierarchy, called resource or Node layer, contains resources or compute nodes with integrated sensors, which provide monitoring information about resources or services running on them. Additionally, each managed resource has integrated Delegates. These are interfaces, which allow to execute commands on managed resources. Furthermore, there exist indirect Delegates which are not directly integrated in resources or Nodes but have an indirect possibility to influence resources or nodes. For example, a RECS microcontroller has the possibility to switch a node on or off.

Each management layer, settled on upper layers of the CooEmAll/TIMaCS hierarchy (Node-Group, Compute-Box1, Compute-Box2), consists of dedicated nodes, called TIMaCS nodes, with monitoring and management capabilities organized in two logical blocks. The monitoring block collects the information from the nodes of the underlying layer. It aggregates the information to calculate new aggregated metrics and makes conclusions by pre-analysing information, creating group states of certain granularity and triggering events, indicating request for reaction. The management block analyses triggered events applying intelligent escalation strategies, and determines which of them need decisions. Decisions are made in accordance with predefined policies and rules, which are stored in a knowledge base filled up by system administrators when configuring the framework. Decisions result in commands, which are submitted to Delegates and executed on managed resources (compute nodes) or other components influencing managed resources, like a job-scheduler, capable to remove defective nodes from the batch queue. The hierarchic structure of the TIMaCS framework allows reacting on errors locally with very low latency. Each decision is reported to the upper layer to inform about detected events and selected decisions to handle the event. The upper layer, which generally has more information, is now able to intervene on received reports by making new decisions resulting in new commands, or an update of the knowledge base of the lower layer. Only escalated reports/events require the effort of an administrator for deeper analysis.

On top of the framework an Admin-Node is settled, which allows administrators to configure the framework, the infrastructure monitoring, to maintain the Knowledge Base and to execute other administrative actions. All nodes are connected by message based communication infrastructure with fault tolerance capabilities and mechanisms ensuring the delivery of messages following the AMQP standard.

A possible deployment of TIMaCS within a rack level based compute-box (ComputeBox1) is shown in Figure 8. For simplification and resource saving purposes, there is only one TIMaCS node deployed, responsible for monitoring and management of all nodes within the ComputeBox1. The monitoring tools are collecting data from the hardware with integrated monitoring, realized by RECS, and from the applications. Monitoring data is collected from all compute nodes of all RECS, as well as from all RECS microcontrollers (µC) within a rack. Collected data is aggregated and stored in a database of the TIMaCS Node, ready to processed and retrieved by GUI or SVD toolkit. After the data has been analyzed, events might be triggered, indicating request for decision. As a result of decision process, commands or actions are selected according to predefined policies or rules, and are submitted to compute nodes, resources, or components influencing resources (such as workload execution component), where they are executed as a reaction on detected events. The resource access and workload execution components are capable to control the physical hardware by setting or changing parameters, and provide mechanisms to control and manage workloads that are executed on the hardware layer.

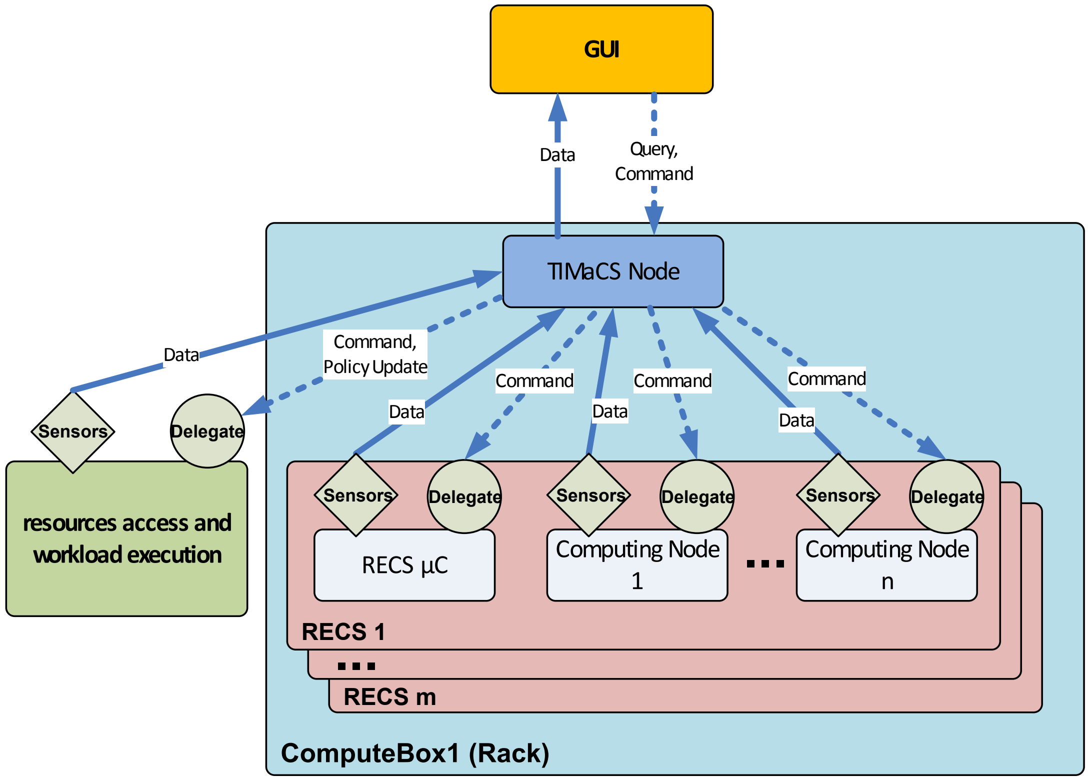

Figure 8: Monitoring and Management in CoolEmAll

The following subsections describe the architecture of the TIMaCS components in detail, explaining monitoring and management building blocks of the framework, shown in Figure 9.

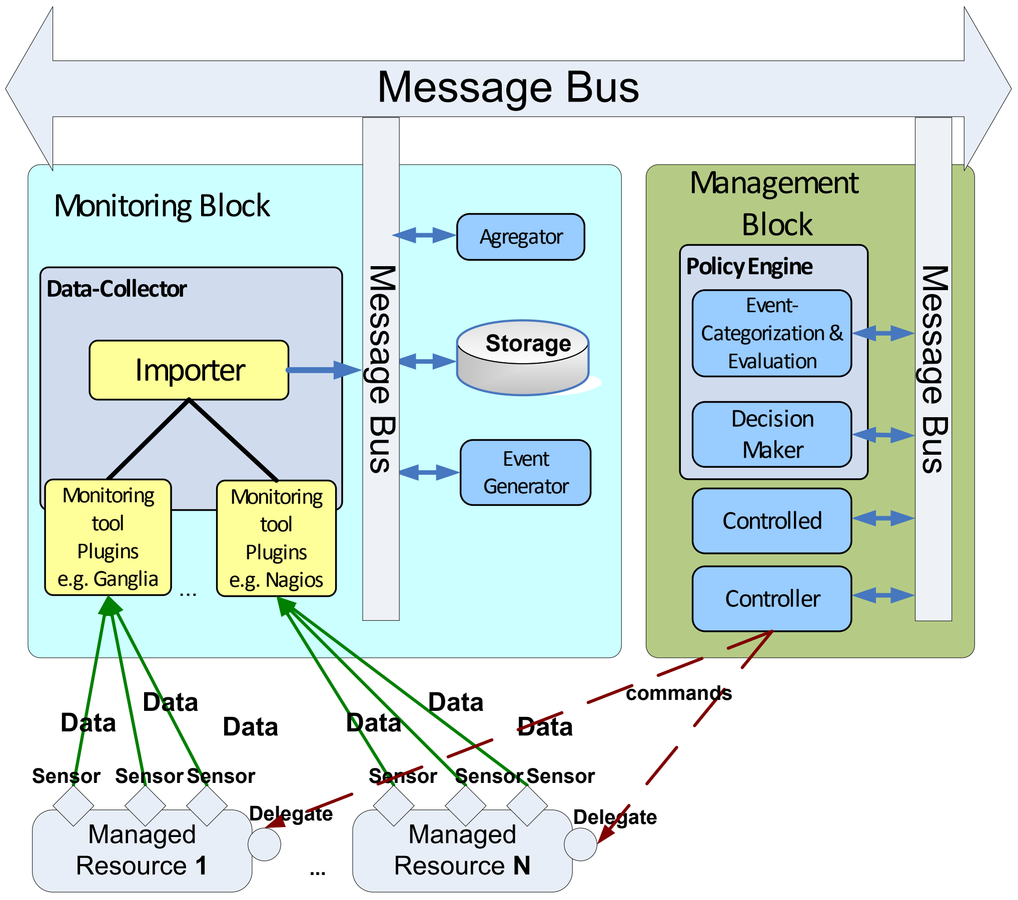

Figure 9 TIMaCS Components

4.3.1 Monitoring

The monitoring capability of the TIMaCS node provided in the monitoring block, consists of Data-Collector, Storage, Aggregator and the Filter & Event Generator. The components within the monitoring block are connected by messaging middleware, enabling flexible publishing and consumption of data according to topics. These components are explained in the subsequent sections.

Data-Collector

The Data-Collector collects metric data and information about monitored infrastructure from different sources, including compute nodes, switches, sensors or other sources of information. The collection of monitoring data can be done synchronous or asynchronous, in pull or push manner, depending on the configuration of the component. In order to allow integration of various existing monitoring tools (such as Ganglia or Nagios) or other external data-sources, TIMaCS uses a plug-in based concept, which allows the design of customized plugins, capable to collect information from any data-source. Collected monitoring data consist of metric values, and are semantically annotated with additional information, describing source location, the time, when the data were received, and other relevant information for data processing. Finally, annotated monitoring data are published according to topics, using AMQP based messaging middleware, ready to be consumed and processed by other components.

Storage

The Storage subscribes to the topics published by the Data-Collector, and saves the monitoring data in the local round robin database. Stored monitored data can be retrieved by system administrators and by components analyzing data history, such as Aggregator or the CoolEmAll GUI. For this purposes TIMaCS monitoring framework provides an API allowing retrieving stored monitoring data using RPC based clients.

Aggregator

The Aggregator subscribes to topics produced by the Data-Collector, and aggregates the monitoring data, i. e. by calculating average, mininimum, or maximum values, or the state of certain granularity (services, nodes, node-groups, cluster etc.). The window-scope used for the aggregation can be adjusted in configuration settings of the TIMaCS. The aggregated information is published in new topics, to be consumed by other components of the same node (i. e. by the Filter & Event Generator), or those of the upper layer.

Filter & Event Generator

The Filter & Event Generator subscribes to particular topics produced by the Data-Collector or Aggregators. It evaluates received data by comparing it with predefined values. In case those values exceed permissible ranges, it generates an event, indicating a potential error. The event is published according to a topic and sent to that components of the management block, subscribed to that topic. In addition to generated event, there might be also actions triggered, depending on event. The evaluation of data is done according to predefined rules, defining permissible data ranges. These data ranges may differ depending on the location, where these events and messages are published. Furthermore, the possible kinds of messages and ways to treat them may vary strongly from site to site and in addition it depends on the layer the node belongs to. The flexibility obviously needed can only be achieved by providing the possibility of explicitly formulating the rules by which all the messages are handled. TIMaCS provides a graphical interface for this purpose, based on eclipse Graphical Modeling Framework

4.3.2 Management

The management block is responsible for making decisions in order to handle predefined events or error events. It consists of the following components: Event Handler, Decision Maker, Knowledge Base, Controlled and Controller, as shown in Figure 2. Subsequent sections describe these components in detail.

Event Handler

The Event Handler analyses received reports and events applying escalation strategies, to identify those which require error handling decisions. The analysis comprises methods evaluating the severity of events/reports and reducing the amount of related events/reports to a complex event. The evaluation of severity of events/reports is based on their frequency of occurrence and impact on health of affected granularity, as service, compute node, group of nodes, cluster etc. The identification of related events/reports is based on their spatial and temporal occurrence, predefined event relationship patterns, or models describing the topology of the system and dependencies between services, hardware and sensors. After the event has been classified as ”requiring decision”, it is handed over to the Decision Maker.

Decision Maker

The Decision Maker is responsible for planning and selecting error correcting actions, made in accordance with predefined policies and rules, stored in the Knowledge Base. The local decision is based on an integrated information view, reflected in a state of affected granularity (compute node, node group, etc.). Using the topology of the system and dependencies between granularities and sub-granularities, the Decision Maker identifies the most probable origin of the error. Following predefined rules and policies, it selects decisions to handle identified errors. Selected decisions are mapped by the Controller to commands, and are submitted to nodes of the lower layer, or to Delegates of managed resources.

Knowledge Base

The Knowledge Base is filled up by the system administrators when configuring the framework. It contains policies and rules as well as information about the topology of the system and the infrastructure itself. Policies stored in the Knowledge Base are expressed by a set of objective statements prescribing the behavior of the system on a high level, or by a set of (event, condition, action) rules defining actions to be executed in case of error or particular event detection, thus prescribing the behavior of the system on the lower level.

The specification of (event condition action) rules requires specification of "*eca*" predicate:

eca *(Resource_Kind, ResourceName, State, Conditions, Target, Actions)*

with

- *Resource_Kind*, type of the ressource that triggered the event
- *ResourceName*, name of the ressource that triggered the event
- *State*, state of the resource at which the particular action should be executed
- *Conditions*, a list of conditions which are evaluated on received event and must be true for the execution of
- *Target*, target-resource, on which commands shall be executed
- *Actions*, a list of commands sent to the resource where they shall be executed

Controller, Controlled and Delegate

The Controller component maps decisions to commands and submits these to Controlled components of the lower layers, or to Delegates of the managed resources. The Controlled component receives commands or updates from the Controller of the management block of the upper layer and forwards these, after authentication and authorization, to addressed components. For example, received updates containing new rules or information, are forwarded to the Knowledge Base to update it. The Delegate provides interfaces enabling the receipt and execution of commands on managed resources. It consists of Controlled and Execution components. The Controlled component receives commands or updates from the channels to which it is subscribed and maps these to device specific instructions, which are executed by the Execution component. In addition to Delegates which control managed resources directly, there are other Delegates which can influence the behavior of the managed resource indirectly.

4.4 Resources Access and Workload Execution Layer

"Resource access and workload execution layer" is a set of tools and services providing means to run and control both single tasks and complex workloads on the CoolEmAll hardware layer. The system has to be designed and implemented to support two main paradigms addressed in the project: HPC and Cloud. The former case assumes running computational batch tasks on physical machines while the latter – managing virtual machines hosting either tasks or services on the testbed. The main aim of these tests is to gather information about responses of the system to various combinations and configurations of hardware and applications running on it. The measured responses in the form of a set of predefined metrics values will be used to determine applications and hardware profiles that will be subsequently modeled in the SVD Toolkit. The models of applications and hardware profiles modeled in SVD Toolkit will be tuned until they are consistent with the measured states obtained as results of empirical experiments for corresponding allocations of applications and hardware configurations (called system snapshots). The final validation of correctness of determined models will be performed by running complex workloads consisting of many tasks of different types (applications) and durations (or frequency of requests in case of services). If an aggregated difference between measured and simulated metrics values is below the defined threshold, it will be treated as a positive verification of correctness of the assumed models. On this basis, they will be used in simulations along with scheduling policies and infrastructure configurations exceeding limits of the CoolEmAll testbed. To perform repeatable experiments in an easy way, the system has to be able to run both synthetic and real workloads stored in commonly accepted formats supported by tools responsible for operating on the testbed as well as the SVD Toolkit. To ensure the repeatability of experiments and comparability of results between workpackages both single tasks together with their input data and parameters and also whole workloads will be stored in project repository. The set of applications will be chosen jointly with other workpackages (mostly with WP5 and WP6) in such a way that it will cover the whole spectrum of typical application categories. The detailed motivation and requirements concerning the functionality of Module Operational Platform in context of running workloads were presented in prior sections.

4.4.1 High Performance Computing Paradigm

As it was stated above one of the two main scenarios, the CoolEmAll project is focused on, is the use of popular and widely utilized High Performance Computing paradigm connected with computational demanding experiments and applications running on clusters. To have the testbed as much similar as possible to real solutions utilized in modern HPC Centers we will choose and deploy on the hardware layer one of the popular queuing systems in order to manage tasks and resources. An additional advantage of having the queuing system is twofold. Most of popular queuing systems offer variety of easily configurable scheduling policies and what is also important the many popular and standard scheduling policies offered by queuing systems are already implemented and supported by the GSSIM [GSSIM] simulation environment being part of the SVD Toolkit. As it was mentioned, the Module Operational Platform will be used internally in the project to work on applications and hardware profiles. The main aim of the tool is to make plenty of experiments and measurements easy to conduct and repeat. The role of Module Operational Platform is to facilitate the phase of gathering real data being an input for building models in SVD Toolkit. The typical scenario will consist of two phases: running the workload either on physical resources or in the SVD Toolkit and then repeating the same experiment in the second environment to verify results. The workload to be considered can be a real one taken from production HPC infrastructure or a synthetic one (generated by Workload Generator tool). For popular scheduling policies supported by both queuing system and GSSIM such workloads could be used without any modification. For policies unsupported by GSSIM, because of their complexity or long time needed to implement them, the workloads will have to be extended by information taken from queuing system describing the assignment of tasks to computational nodes. The mentioned information will allow forcing the simulator to repeat the given allocation of tasks without necessity to implement the whole scheduling algorithm. Based on the original workload and information taken from the Module Operational Platform (queuing system logs, TIMaCS) it will be possible to perform the same experiments in GSSIM environment and to synchronizing and compare values of empirically measured metrics with ones calculated by GSSIM. The process of submission of workload tasks to the hardware layer managed by queuing system will be performed by the Workload Executor System. The system will be in charge of reading the HPC format workload files and submission of tasks to the queuing system in accordance with the information defined in workloads. In case of verification of scheduling policy developed in the SVD Toolkit (GSSIM) environment on the testbed, instead of modifying queuing system scheduler to support policy to be verified, we will apply once again the “force” approach. Based on the schedule generated by GSSIM, the Workload Executor System will reproduce it on the testbed “forcing” tasks the same assignment as well as changes of states and/or configurations of nodes. During the experiment the monitoring infrastructure will publish and collect values of metrics defined in CoolEmAll project. Both current and past states of the testbed will be visualized by Visualization Tools (including Virtual Thermal Camera). Additionally, Visualization Tools will allow graphical comparison of system states obtained during empirical experiments with ones being results of the SVD Toolkit simulations.

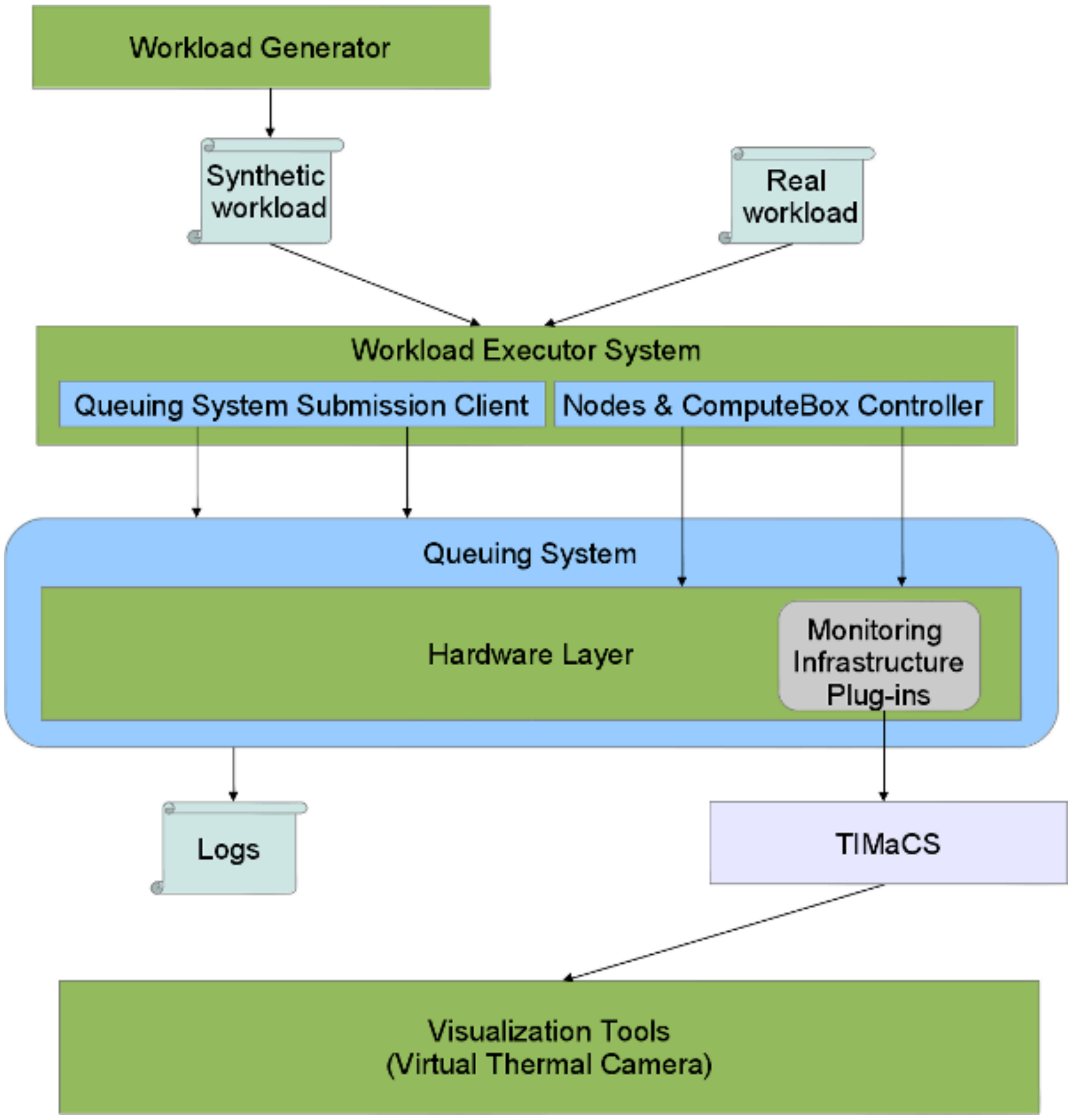
Workload Generator
Synthetic workload
Real workload
Workload Executor System
Queuing System Submission Client
Nodes & ComputeBox Controller
Queuing System
Hardware Layer
Monitoring Infrastructure Plug-ins
Logs
TIMaCS
Visualization Tools (Virtual Thermal Camera)

Figure 10 HPC Verification Environment Architecture

4.4.2 Cloud Paradigm

Cloud computing is a grown up market and computing paradigm is nowadays widely used by service providers and users utilizing any of the three cloud [CLOUD] known models: IaaS, PaaS and SaaS. As the aim of CoolEmAll is not the development services, but the study of the behavior of the infrastructure concerning energy and consumption aware, the simulations and tests will be run using the IaaS model. In an IaaS environment a service provider generates a set of virtual machines where the software (batch task, web servers, databases, etc.) runs. This set of virtual machines together with a description of the virtual infrastructure (the VMS) on the OVF [OVF] standard.

The OVF descriptor is a XML file that follows the dsp8023_1.1.0.xsd [dsp8023] XML schema. It defines the content and requirements of the packaged virtual appliance.

The main element of the schema is the Envelope. The main elements of the envelope are:

List of References: The list of the rest of files that are part of the OVF package. Typically they are virtual disk files, ISO images, and internationalization resources.

Section Elements provides meta-data information for the virtual appliance components in the OVF package. It includes information such as:

- DiskSection, describing meta-information about all virtual disks in the OVF package (e.g., capacity).
- NetworkSection, describing meta-information about all logical network used in the OVF package. Only names and descriptions can be specified.
- ResourceAllocationSection, specifying resource reservations (e.g., CPU, memory, virtualization technology, etc.) to perform at deployment time for virtual machines.
- ProductSection, specifying product information for software packages in virtual machines. Apart from information regarding the software package itself (e.g., vendor, product name, version, etc.) it also allows specifying customization parameters to help configuring the application automatically at deployment time (e.g., DNS servers, gateways, etc.).
- StartupSection, allowing specifying the powering on sequence order for virtual machines that compose the application.

- DeploymentOptionSection, allowing the specification of different configuration options, each one corresponding to different sets of virtual hardware.
- OperatingSystemSection, specifying the virtual machine operating system.
- InstallSection, specifying that the virtual machine has to be booted as part of the deployment process to install and configure software

Content: A description of the content. It can be either a single virtual machine (VirtualSystem element) or a collection of multiple virtual machines

(VirtualSystemCollection element).

OVF allows tohave a portable Virtual Appliance format among multiple hypervisors. In order to execute the virtual appliance defined in a package for each hypervisor it is required to convert this portable format to the specific hypervisor's file format. Currently there are available conversion tools for the following hypervisors: VMWare, XenServer and VirtualBox. The OVF format has the possibility to be extended which will be used in CoolEmAll and specifically in the MOP for defining the elasticity rules and the workloads of the applications used in the validation cloud scenarios. As the workloads are embedded in the OVF file each application has one specific workload based on the elasticity rules and hence the application will be stress in different ways, from the concurrent connections to the amount of CPU used or the memory used.

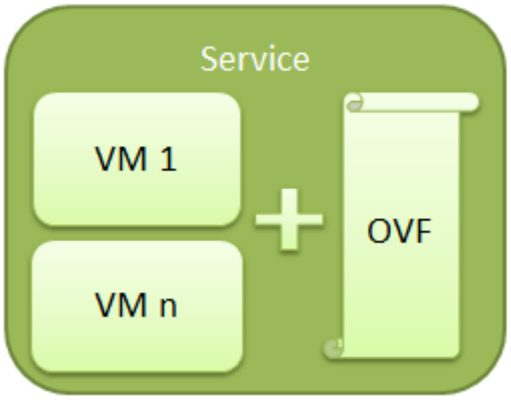

Figure 11 Service Description

The MOD provides two layers of management for the cloud infrastructure one on top of the other which manage two different levels. On the bottom level there is the virtual machine manager OpenNebula [ONE] which is an open source software used for the management of the virtual machines placement and the deployment of them on different hypervisors. However on top of this vm management level and for simulating what a real cloud infrastructure provider would do the MOP will use a second layer of management for what we defined as a service prior in this section. This second layer goes one step above in the abstraction level of a cloud infrastructure and it does not talk to the hypervisors directly but that utilizes the OpenNebula interface for such issue. The aim behind the separation and the need of the second management layer is the provisioning of elasticity at the service level based on the elasticity rules that the OVF document has defined. So as to do so this level is using two components, on one hand is the elasticity engine and on the other hand is the service life-cycle manager.

The elasticity engine saves per service the elasticity rules associated and uses the value specified on the OVF and it also collects information from the monitoring system, for both virtual and physical information. In the case one of this values passes the level up or down from the elasticity rule this component will notify the second component at this level, the Service Life-cycle Manager. This management layer works at service level and takes as input the service in the format described above with the set of VMs and the OVF doc. The component is in charge of manage the whole life-cycle of a service and it provides capabilities to deploy, undeploy, start, stop and through the elasticity rules scale up and down depending on the current workload of the service talking directly with the OpenNebula Interface.

The application workload is a direct consequence of the elasticity rules and hence for each service these have to be specified inside the OVF document. JMeter [JMeter] is used to simulate user concurrence connections will be used in order to stress the virtual appliances to force the elasticity rules and scale up or down the service due to the demand of more resources when the concurrent connections raise or drop.

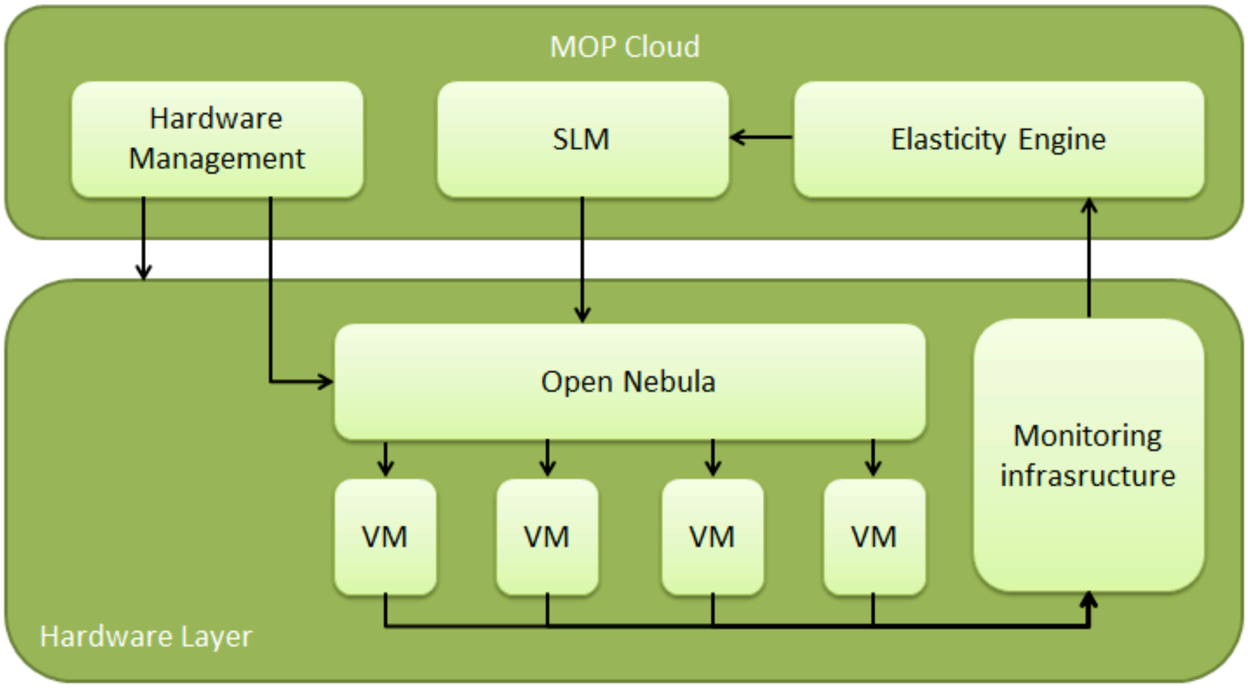

Figure 12 MOP Cloud Architecture

4.5 MOP Graphical User Interface

The Module Operation Platform GUI will deliver graphical user interfaces to support users in monitoring the project testbed, watching results of experiments and comparing them with simulations. These interfaces will include web access enriched with advanced visualization as it was specified within requirements for Module Operation Platform GUI described in requirements section (Section 3.5).

The design of general GUI functionality and architecture was done on the basis of use cases UC2.1-2.3 defined in Section 2. Although there are common elements in an interaction with GUI for all use cases, each use case has some additional elements allowing to access functionalities specific for the use case. For UC2.1 it is capability of sending commands to entities of the CoolEmAll testbed, this will be realized with utilization of controls accessible in the side panel. For UC2.2 and UC2.3 selection of point in time is a must, functionality will be possible both from form/slider in a side panel as well as from the Chart Generator panel, so selection of a point in time on the 2D chart will result in picking of the same point in time in the visualization "windows". For UC2.3 sharing of the point in time, point of view, scale, scene and selected testbed entity between visualization “windows” is necessary. MOP GUI architectures for all these use cases are presented in Section 4.5.2 and illustrated in Figures 11-13.

All of the use cases will be implemented in the form of 3D interfaces, so interpretation of data by the end user will be natural. Full functionality will be accessible from web browser (and additionally mobile client enhancing remote access and virtual thermal camera steering options). This will enable end users with possibility of choosing of the way of interaction with the 3D scene, so the solution fits their needs. Scene will consist of previously prepared 3D models of entities in the CoolEmAll testbed extracted from DEBBs that will be prepared in the scope of the WP2 (concept of DEBB was defined in deliverable [D3.1]). DEBB will contain additional meta information allowing to bound some particular 3D model or its part with entity and metrics to be inserted into the 3D scene. Depending on the use case, sources of enriching data will be results of simulations or infrastructure monitoring (for historical and real-time data).

Aside from 3D view that will present all metrics that can be visualized, there will also be a side panel containing textual data as well as 2D graphics (e.g. charts illustrating values of selected metrics in time). Due to its source of content, the side panel will be referenced in the remaining part of the document as the Chart Generator panel. Regardless of number of visualization windows, just one Chart Generator panel is foreseen in any of the use cases. For selection / interaction purposes, a side panel with various UI controls (HTML5 controls) will be accessible. End user will be able to customize GUI or take various actions by selecting options from the lists, pressing buttons etc.

Temperatures of solids (e.g. CPUs) will be illustrated in 3D with different colors. 3D visualization of air flow will reflect both direction and velocity of the air (e.g. by utilization of vectors). Load of the computing nodes will be also visible (e.g. in form of character string displayed in the scene or color of the entity). As mentioned, additional data like list of possible actions or longer text will be illustrated outside of the main window of the visualization (mostly in the Chart Generator panel), in order not to obfuscate the view.

In the following section there are described specific solutions on which base CoolEmAll will provide MOP GUI that will allow fulfilling requirements.

4.5.1 Module Operation Platform GUI selected solutions

4.5.1.1 Vitrall

The tool which enables efficient and easy integration web interfaces and 3D visualization is Vitrall [Vitrall] - a distributed web based visualization system designed to utilize efficiently multi-GPU server installations. In a nutshell, it allows highly efficient parallel rendering and high-resolution visualization of well-known data exchange formats among various scientific and engineering applications. Vitrall can be easily installed and maintained on a small GPU server as well as large scale GPU clusters thanks to its distributed and scalable architecture. It can be configured to display virtual scenes in real-time on different visualization panels, caves or geometric prisms to meet specific users requirements. Moreover, Vitrall can be used in collaborative environments, where many users can simultaneously interact in real-time, share or modify virtual 3D models over the Internet, as the processing part is performed completely on the remote side close to the data sources. The Vitrall application is a very modular solution based on the notion of exchangeable plugins.

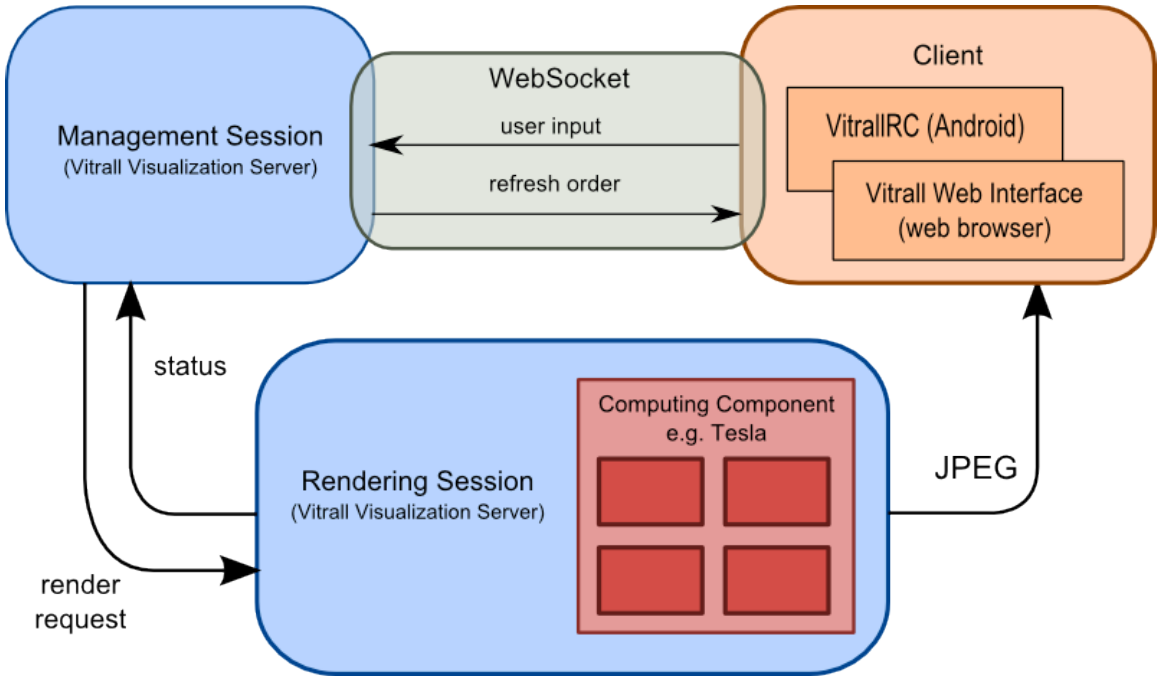
Management Session
(Vitrall Visualization Server)
WebSocket
user input
refresh order
Client
VitrallRC (Android)
Vitrall Web Interface (web browser)
status
Computing Component e.g. Tesla
Rendering Session
(Vitrall Visualization Server)
JPEG
render request

Figure 13 Logic architecture of the Vitrall system

The Vitrall rendering abilities are accessible through already existing web protocols: HTTP and WebSockets [WebSockets]. HTTP is used for sending compressed images, while user input (information read from sensors and from user forms) is sent with a WebSockets connection. WebSockets technology is a part of the HTML5 specification and is already implemented in most popular web browsers, like Firefox, Chrome, etc. Currently, two implementations of the Vitrall client are available: one for the web browser and a second for Android [Android] based mobile devices. While the web browser client is very easy to use (in principle it consists of a single web page with some JavaScript code), it has limited input capabilities because of constraints concerning a direct access to sensors from the browser environment. Still, thanks to Firefox's ability to expose the Windows multi-touch interface, Vitrall can provide a multi-touch user experience through a thin web browser client used on a tablet or other Windows based touch-enabled device. On the other hand, as a part of the Vitrall project, a dedicated Smartphone client was developed for the Android OS platform. This client has full support for sensors like gyroscope, multi-touch, accelerometers and magnetic field sensors. Though it is targeted specifically for work with the Vitrall system, it is a general solution in terms of the different scenarios that can be covered using the Vitrall platform – all information particular to a given scenario like user forms and supported user input methods is sent to the client using an XML based protocol. The Android based client [VitrallRC] is available in the Android Market and once installed can be used to connect to different Vitrall instances running different visualization scenarios. It is also worth mentioning that some web browsers available on mobile devices, Safari on iPad for example, already implement the WebSockets specification, allowing the usage of the Vitrall system just out-of-the-box.

One of the outstanding features of the Vitrall system is its flexibility with regard to defining the way how user input should be interpreted. All sensor information is sent in raw form: points of contact for multi-touch enabled devices (Smartphones, tablets), orientation in space for magnetic field sensors and gyroscopes, etc. That information is then interpreted by the Vitrall server, producing signals of a more semantic nature (rotate, translate, click) that are then routed to elements of the 3D scene in a way specific to the visualization scenario that can be changed dynamically.

In the scope of the CoolEmAll project it is important to note that Vitrall uses well known solutions not only regarding the choice of Internet protocols and data formats, but also for the underlying rendering engines. Vitrall uses the OpenSceneGraph [OpenSceneGraph] library for management of the 3D scene, and an integration of the VTK library is planned for the near future. Thanks to the Vitrall plugin system it is possible to load data for visualization using a dedicated library (if needed), or to implement other communication protocols (for example a virtual file system). This capability allows for easy integration with various data sources like TIMaCS or results of CoolEmAll simulations (SVD Toolkit), which make it possible to easily compare data from the real infrastructure and simulations. Vitrall is under active development at PSNC. One of the main advantages of Vitrall is that it is an Open Source solution.

4.5.1.2 Chart Generator

Although the 3D visualizations presented in the main GUI scene are very useful for displaying different monitoring information in given point of time, they are not suitable to show in one quick view simple metrics, especially including historical data as needed in use cases UC2.2 and UC2.3. Standard way of graphical representation of simple metrics takes an advantage of 2D chart. The Chart Generator software will present given metrics in form of 2D chart allowing simultaneously to interact with the source 3D scene. It means that user will be able not only to choose a metric to be displayed but also to select values on charts and see in 3D view e.g monitoring information for given object or point in time, dependably on metric type. The final solution will be able to present more than one series in one chart. It will allow to make easy and quick comparisons of metrics.

The natural choice for the technology that will be used to prepare an actual solution is limited to commonly used Adobe Flash (Flex) and more and more popular HTML5. Adobe Flex based solutions has been already adopted by many web based projects, have native support for basic charts drawing and reach library of open source components, however the developer has to be aware that

Flash is Adobe proprietary solution and e.g. Flash plugins may be changed without developer control causing software failures. The advantage of using HTML5 based solution for preparing Chart Generator is not only it's standard based implementations but also that Vitral solution is HTML5 based which should make the required integration easier. Additional advantage of HTML5 is global tendency to support in all modern web browsers and mobile devices, comparing it with declining market share of Flash based solutions. Additionally, choosing the technology, we have to take into consideration large size of ready to use Flash components and not final stage of HTML5 standard which can be disadvantages for given solutions. Before the final choice of technology to use will be made, all advantages and disadvantages of both will be taken into consideration in order to prepare one most suitable for Module Operation Platform solution. Additional factor that will be taken into consideration choosing the technology will be time and effort needed to meet functionality requirements for both options.

4.5.2 Web Client and Mobile Access Client

Described solution will be mostly based on the Vitrall software (3D part as well as side panel). Most of MOP GUI requirements defined in requirements section can be easily fulfilled by this software. In addition Vitrall offers both web clients (for ordinary HTML5 enabled web browser also for some mobile devices) and dedicated mobile client VitrallRC (for Android based devices). Output of the Chart Generator will be aggregated to the web page generated by the Vitrall software, so as a result one coherent solution will be presented to the end user. Chart Generator and Vitrall will not be tightly integrated, and communication between these 2 components will take place on the client-side (e.g. with utilization of JavaScript).

VitrallRC supports full functionality offered in the Vitrall Web Client thus enabling end users with most functionalities which will be implemented for MOP Web Client with almost no extra effort. Additional functionalities being not the part of the core of Vitrall - functionalities offered by Chart Generator are foreseen to be added to the VitrallRC as an extension. Achieving this is likely to require minor software development. As a result, for UC2.1-UC2.3, the same functionality will be offered in MOP Web Client, as well as in MOP Mobile Client, with only one difference caused by limited capabilities of mobile devices (screen size and computing power) for UC2.3, where views to "windows" for comparisons and outcome of the Chart Generator will not be presented side by side on the screen, but rather one of them will be visible at a time.

While support for web client will allow to access MOP GUI from, as many platforms as it is only possible, dedicated mobile client will be limited to Android OS platform, but in return it will enable end user with new quality of human computer interaction – natural user interface based on sensors built into modern mobile devices (gyroscope, accelerometers and magnetic field sensors). Orientation of the mobile device used for interaction will be reflected by orientation of viewpoint (direction of view), while interaction with the multi-touch screen will enable end user with capability of changing localization of the viewpoint in the 3D scene. This will give an impression, that the mobile device held in hand by the end user is a “virtual thermal camera”.

Architecture of system for mentioned use cases - UC2.1, UC2.2 and UC2.3 covering Vitrall and Chart Generator is illustrated respectively in figure 11, figure 12 and figure 13. Differences are limited to sources of data on the backend (for UC2.1 and UC2.2 only TIMaCS is supposed to be data source, for UC2.3 data for comparison will be also obtained from simulations) and nature of data (while for UC2.1 data is real-time, for UC2.2. and UC2.3 it is historical).

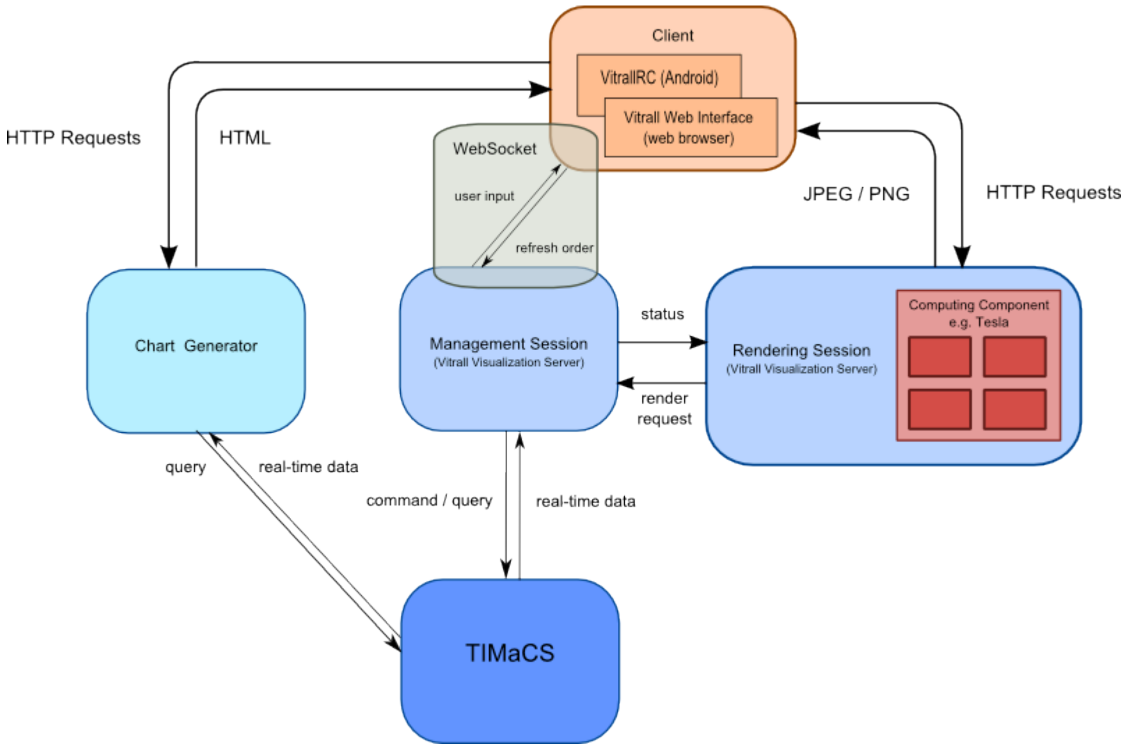

Figure 14 Architecture of MOP GUI for UC2.1.

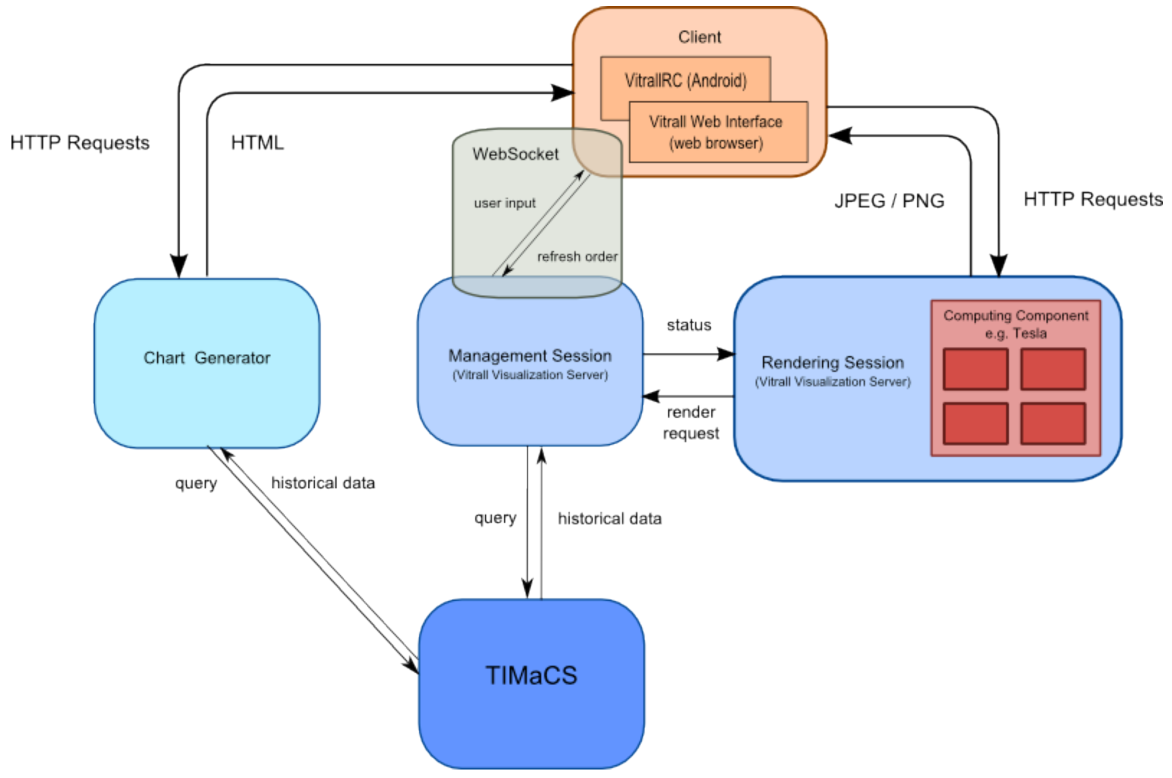

Figure 15 Architecture of MOP GUI for UC2.2.

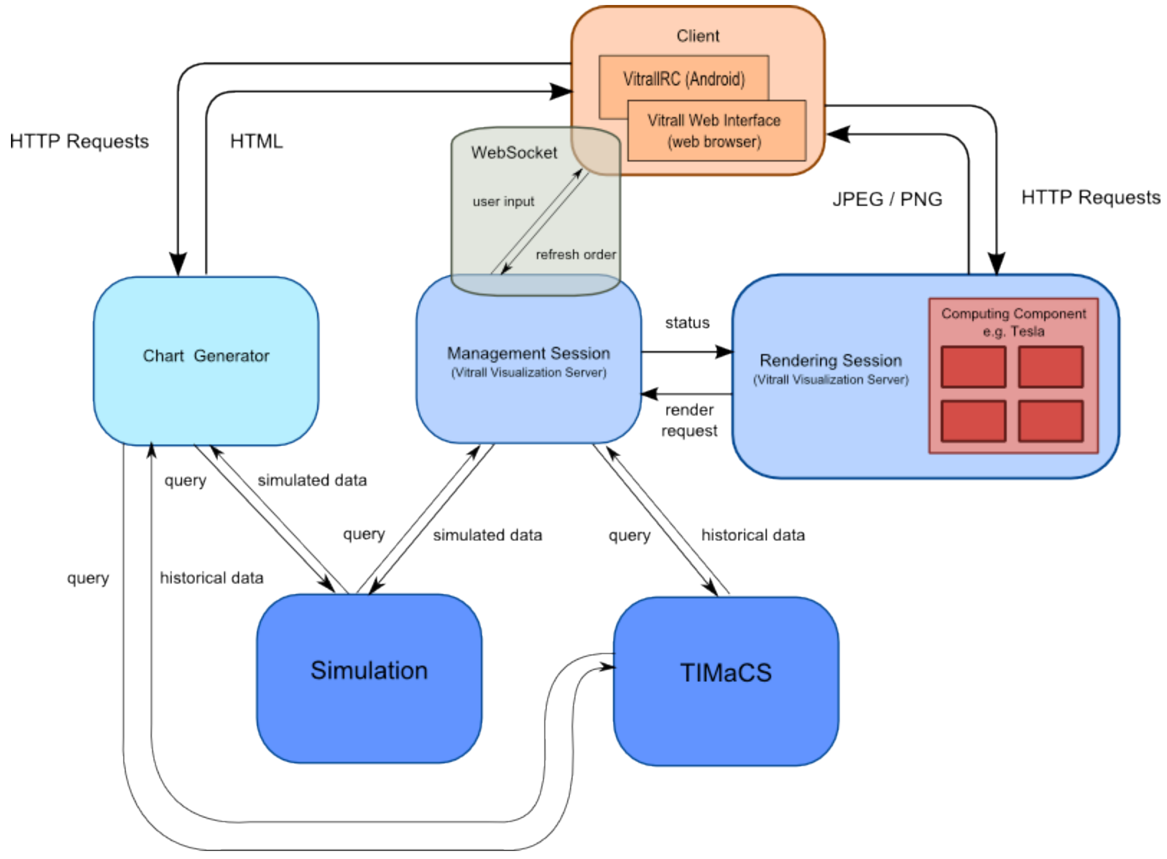

Figure 16 Architecture of MOP GUI for UC2.3.

The detailed design of MOP GUI will take into account the specification of DEBB and ComputeBox to be defined by WP2 and WP3, WP2 decisions concerning file formats and interfaces (to be compatible with simulation results), final set of metrics provided by WP5, and a set of scenarios defined within WP6. The detailed design along with a first release of the MOP GUI will be delivered in M18 in deliverable D4.4.

5 Conclusions

This deliverable describes a general architecture of the Module Operation Platform (MOP), a set of tools that will be used to enable access to the CoolEmAll testbed and to facilitate conducting experiments concerning energy efficiency and heat transfer. MOP will be essential to allow project partners to develop models of applications, hardware and their impact on energy consumption and temperatures. It will be also crucial to verify CoolEmAll simulation tools as well as energy- and thermal-aware management policies.

Deliverable D4.1 provides general requirements and architecture for MOP, which will be further enhanced with additional details and specific configurations based on verification scenarios defined by WP6. The next steps will include design of more detailed architecture, specific interfaces and formats to enable integration of software and hardware tools, and graphical user interfaces. The first release of the MOP is scheduled on M18, however, first basic environment will be deployed on top of the CoolEmAll RECS testbed before M12 to enable tests and experiments.

Deliverable D4.1 should be taken as an input to other deliverables that refer to tests on real computing infrastructure, in particular D6.1 (to define feasible verification scenarios), D5.2, D5.3, D5.5 (to integrate application monitoring tools and benchmarks with MOP) and obviously subsequent deliverables within WP4: D4.2, D4.3, D4.4.

6 References

[Android]	Google, “Android,” 2012. [Online]. Available: http://www.android.com/. [Accessed 2012].
[CLOUD]	http://en.wikipedia.org/wiki/Cloud_computing
[D2.1]	CoolEmAll Deliverable 2.1 - Report about state-of-the-art simulation and visualisation environments
[D2.2]	CoolEmAll Deliverable 2.2 - Design of the CoolEmAll simulation and visualisation environment
[D3.1]	CoolEmAll Deliverable 3.1 - First definition of the flexible rack-level ComputeBox with integrated cooling
[D5.1rel]	CoolEmAll Deliverable 5.1 - White paper on Energy - and Heat-aware metrics for computing modules - Related Work. *(Unofficial and for the project internal use version of D5.1 introducing primary set of metrics to be addresses in the early stage of the project)*
[D5.1]	CoolEmAll Deliverable 5.1 - White paper on Energy - and Heat-aware metrics for computing modules.
[D6.1]	CoolEmAll Deliverable 6.1 - Validation Scenarios Methodology and Metrics
[DC Investments]	http://www.tmcnet.com/voip/ip-communications/articles/218333-data-center-investments-comprise-significant-share-it-budgets.htm
[DEBBDef]	DEBB Definition Glossary, available at: http://coolemall.eu/web/guest/documents/-/document_library/view/11873/432?_20_redirect=http%3A%2F%2Fcoolemall.eu%2Fweb%2Fgues t%2Fdocuments%2F-%2Fdocument_library%2Fview%2F11873
[dsp8023]	http://schemas.dmtf.org/ovf/envelope/1/dsp8023_1.1.0.xsd

[GAMES D2.1]	Jiang, T., Kipp, A., Cappiello, C., Fugini, M., Gangadharan, G.R., Ferreira, A.M., Pernici, B., Plebani, P., Salomie, I., Cioara, T., Anghel, I., Christmann W., Henis, E.A., Kat, R.I., Lazzaro, M., Ciuca, A., Hatiegan, D. (2010). Layered Green Performance Indicators Definition – GAMES Deliverable 2.1. Retrieved March 28, 2012, from http://www.green-datacenters.eu/
[GSSIM]	S. Bąk, M. Krystek, K. Kurowski, A. Oleksiak, W. Piątek and J. Węglarz, "GSSIM - a Tool for Distributed Computing Experiments", Scientific Programming Journal, vol. 19, no. 4, pp. 231-251, 2011.
[JMeter]	http://jmeter.apache.org/
[Nagios]	Nagios Website. Retrieved March 28, 2012, from http://www.nagios.org/
[ONE]	OpenNebula, www.opennebula.org
[OpenScene Graph]	OSG Community, "Open Scene Graph Official Website," 2011. [Online]. Available: http://www.openscenegraph.org/. [Accessed 2012].
[OVF]	Open Virtualization Format, http://www.dmtf.org/standards/ovf
[SWF]	Parallel Workload Archive, http://www.cs.huji.ac.il/labs/parallel/workload/.
[TIMaCS]	Tools for Intelligent Management of very large Computing Systems, BMBF project, www.timacs.de
[VitrallRC]	G. Grzelachowski, T. Kuczyński and P. Śniegowski, "VitrallRC," 2011. [Online]. Available: https://market.android.com/details?id=org.vitrall.client. [Accessed 2012].
[Vitrall]	P. Śniegowski, M. Błażewicz, G. Grzelachowski, T. Kuczyński, K. Kurowski and B. Ludwiczak, "Vitrall: web-based distributed visualization system for creation of collaborative working environments.," in Parallel Processing and Applied Mathematics, Toruń, 2011.
[Volk]	Eugen Volk, Jochen Buchholz, Stefan Wesner et al. (2011) Towards Intelligent Management of Very Large Computing Systems . In Proceedings of the CiHPC (Competence in High Performance Computing) Conference 2010.

[WebSockets] W3C, “The WebSocket API,” 2012. [Online]. Available: http://dev.w3.org/html5/websockets/. [Accessed 2012].