D3.1: Interactions between system modules and messages format

Author: R. Roche

Version: 1.0

Date: 22/12/2017
Deliverable information

<table>
<thead>
<tr>
<th>Deliverable number</th>
<th>D3.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contractual date of delivery</td>
<td>31/12/2017 (M27)</td>
</tr>
<tr>
<td>Actual date of delivery</td>
<td>22/12/2017</td>
</tr>
<tr>
<td>Title of deliverable</td>
<td>D3.1: Interactions between system modules and messages format</td>
</tr>
<tr>
<td>Dissemination level</td>
<td>Restricted to other program participants (including the ANR Services)</td>
</tr>
<tr>
<td>WP contributing to the deliverable</td>
<td>WP3</td>
</tr>
<tr>
<td>Author</td>
<td>R. Roche</td>
</tr>
<tr>
<td>Co-authors</td>
<td>A. Sayah, D. Hissel, P. Stolf, S. Caux, J.-M. Pierson, G. DaCosta, B. Celik, J.M. Nicod</td>
</tr>
</tbody>
</table>

Revisions

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>22/12/2017</td>
<td>R. Roche</td>
<td>Final version</td>
</tr>
</tbody>
</table>

Abstract

The aims of this deliverable are to:
- Specify the links and interactions between the different modules in the system, i.e., how the various modules coordinate their actions.
- Specify the format of data exchanged between the different modules.

As a whole, this document provides an overview of the operation of the system. The detailed algorithms of the modules will be further studied in WP4, the implementation in WP5.

Keywords

module, interaction, message, algorithm, communication, format, JSON
# Table of contents

1. Global system structure 5

2. Modules roles and interactions 6
   2.1. Power system (PS) 6
      2.1.1. Description 6
      2.1.2. Interactions with other modules 9
   2.2. Power decision module (PDM) 10
      2.2.1. Roles 10
      2.2.2. Interactions with other modules 12
   2.3. IT decision module (ITDM) 14
      2.3.1. Roles 14
      2.3.2. Interactions with other modules 15
   2.4. Negotiation module (NM) 17
      2.4.1. Roles 17
      2.4.2. Interactions with other modules 18
   2.5. IT system (ITS) 19
      2.5.1. Description 19
      2.5.2. Interactions with other modules 19

3. Messages format 20
   3.1. Middleware 20
   3.2. Messages structure 21
   3.3. List of messages 23

4. Next steps 28

Appendix 1: Description of messages 29
### Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>Alternating current</td>
</tr>
<tr>
<td>DC</td>
<td>Direct current</td>
</tr>
<tr>
<td>DC</td>
<td>Datacenter</td>
</tr>
<tr>
<td>H₂</td>
<td>(Di)hydrogen</td>
</tr>
<tr>
<td>IT</td>
<td>Information technology</td>
</tr>
<tr>
<td>ITDM</td>
<td>IT decision module</td>
</tr>
<tr>
<td>ITS</td>
<td>IT system</td>
</tr>
<tr>
<td>MPPT</td>
<td>Maximum power point tracking</td>
</tr>
<tr>
<td>NM</td>
<td>Negotiation module</td>
</tr>
<tr>
<td>PDM</td>
<td>Power decision module</td>
</tr>
<tr>
<td>PS</td>
<td>Power system</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of service</td>
</tr>
<tr>
<td>RUL</td>
<td>Remaining useful life</td>
</tr>
<tr>
<td>SOC</td>
<td>State-of-charge</td>
</tr>
<tr>
<td>SOH</td>
<td>State-of-health</td>
</tr>
</tbody>
</table>
1. Global system structure

The system includes several subsystems named modules in the following. Two main sides are distinguished: the IT side, and the (electric) power side.

The modules include (see Fig. 1.1):

- **IT system (ITS):** the IT equipment, which can be real, such as an OpenStack system, or simulated, as with DCWorms or SimGrid for instance.
- **IT Decision module (ITDM):** in charge of taking decisions regarding how the IT load is scheduled and of interacting with the NM.
- **Negotiation module (NM):** in charge of the negotiation between the IT and the power sides.
- **Power decision module (PDM):** in charge of taking decisions regarding how the electric power generation and storage units are controlled and of interacting with the NM.
- **Power System (PS):** the power equipment (such as the renewable energy sources and the storage units), which as for the IT side, can be real (using Power Hardware in the Loop-PHIL to emulate part of this equipment) or simulated (PSS).

As described in the following sections, these modules interact with each other by exchanging messages so decisions can be made.

![Fig. 1.1: Structure of the system, including the different decision modules](image-url)
2. Modules roles and interactions

In this section, we describe what are the roles of the modules listed above and how they interact with each other, i.e., what information they exchange, when, in what order, etc. This is also expected to serve a first step toward defining the algorithms running in each module. The contents will be further developed and completed in WP4.

2.1. Power system (PS)

The Power System (PS) is a set of physical electrical components equipped with sensors and actuators. It includes energy sources, storage units, switches, etc., with their local controllers and low-level control loops (time constants $\ll 1$ sec.). These sources can be real (as in the PHIL testbed) or simulated (as in the simulator).

2.1.1. Description

Microgrid structure and equipment

As shown in Fig. 2.1.1, the microgrid supplying the datacenter loads (IT and cooling) includes several components: PV panels, wind turbines, batteries, hydrogen storage (combining electrolyzers, tanks and fuel cells), as well as supercapacitors.
All components are connected to the same high voltage direct current (DC) bus through static converters. These power electronics-based converters are used to convert electric power from one form to another, e.g., from a given DC voltage to another one, or from AC to DC and vice-versa. Note that similar converters are used to supply the IT equipment operating in low voltage DC from 230V AC power.

These converters are thus required to connect the components to the microgrid, as well as to control their output. Although converter control is not the focus of this project, it is required to control the operation of the storage units (batteries and hydrogen storage). This control takes as input a current or power setpoint issued by the PDM, and transforms it into a pulse width modulated (PWM) signal fed into the converter. This PWM signal then drives the operation of converter switches, which in turn impact the current and voltage. The generation of the PWM signal is either done by a central controller or by a local controller attached to the component, on the basis of setpoints received from an energy management system.

For protecting the equipment, circuit breakers may also be used together with relays to enable automatically or manually disconnecting components that may suffer from a fault: if an abnormal situation is detected (e.g., a very high current is detected), the part of the microgrid causing the fault should be isolated as fast as possible to avoid further damage. This is also a safety measure for the (human, on site) datacenter operators.

**PS roles**

The main role of the PS is to make sure that the electric power necessary to supply the load of the datacenter (IT load and cooling) is sufficient. As discussed above, all components are assumed to be controlled by a local controller in charge of:

- Running measurements of various physical quantities (current, voltage, etc.),
- Receiving set points from the PDM (i.e., current or power setpoints),
- Operating the components so they reach their set points as fast as possible, if it is possible. This aspect will be further described in WP4 and WP5.

The table below summarizes the local controller roles, quantities to be measured (outputs) and input commands for the listed components. Note that in the PHIL and simulator implementations of the microgrid, for the sake of simplicity and due the limited duration of the project, several roles and inputs/outputs may be omitted. For example, the aging of the components will only be based on models, and not on actual diagnostics (e.g., from impedance spectroscopy). Such elements are written in italics in the table.

<table>
<thead>
<tr>
<th>Components</th>
<th>Local controller role</th>
<th>Outputs (sensors)</th>
<th>Inputs</th>
</tr>
</thead>
</table>
| Battery    | Implement current and voltage set point  
*Estimate SOC*  
*Estimate SOH*  
*Ensure pack safety (BMS)* | Voltage  
Current  
Alerts from controller  
*Breaker status*  
*Temperature*  
*Impedance* | Instantaneous power set point  
Breaker status |
<table>
<thead>
<tr>
<th>Component</th>
<th>Function</th>
<th>Measurements</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supercapacitor</td>
<td>Implement current and voltage setpoint</td>
<td>Voltage, Current, Alerts from controller, Breaker status, Temperature</td>
<td>Instantaneous power set point</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Breaker status</td>
</tr>
<tr>
<td>Electrolyzer</td>
<td>Implement set point in terms of voltage and current</td>
<td>Voltage, Current, H₂ production rate, Alerts from controller, Breaker status, Temperature, Impedance</td>
<td>Instantaneous power to convert in H₂ set point</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Breaker status</td>
</tr>
<tr>
<td>Fuel cell</td>
<td>Implement set point in terms of voltage and current</td>
<td>Voltage, Current, H₂ consumption rate, Alerts from controller, Breaker status, Temperature, Impedance</td>
<td>Instantaneous power set point</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Breaker status</td>
</tr>
<tr>
<td>Hydrogen tank</td>
<td>Estimate SOC</td>
<td>Pressure, Alerts from controller</td>
<td></td>
</tr>
<tr>
<td>PV</td>
<td>Optimize output power deliverable (MPPT)</td>
<td>Current, Voltage, Alerts from controller, Breaker status, Temperature</td>
<td>Instantaneous power (desired Voltage and current)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Solar radiation, Breaker status</td>
</tr>
<tr>
<td>Wind turbine</td>
<td>Optimize output possible power</td>
<td>Current, Voltage, Alerts from controller, Breaker status</td>
<td>Instantaneous power (desired voltage and current)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Wind speed, Breaker status</td>
</tr>
<tr>
<td>Load (IT and thermal)</td>
<td>Current, Alerts from controller</td>
<td></td>
<td>Breaker status</td>
</tr>
</tbody>
</table>

Table 2.1.1: Component characteristics

Practical implementation also requires defining sampling periods for the different measurements:

- An order of magnitude of 1 µs to 1 ms for electrical quantities seems appropriate for converter control, while 1 s to 1 minute would be sufficient for tasks such as dispatching, and 1 to 15 minutes for scheduling. These measurements can then be downsampled or averaged for applications with longer time horizons.
- For weather quantities, a resolution of 1 minute seems sufficient for solar radiation and wind speed, while 5 minutes would be enough for temperature.
2.1.2. Interactions with other modules

Based on the above, we observe that the PS interacts only with the PDM, and not with the other modules. Its actions with respect to the PDM are to (see Fig. 2.1.2):

- Receive set points and status change requests from the PDM, including current/power set points, and relays/switches and valves status change commands. An example is when a new power profile (as defined in deliverable D4.1) must be implemented by the microgrid.
- Receive requests from the PDM for a new batch of measurements, estimates (e.g., for SOC and SOH), and status updates (e.g., on breaker status). An example is when the PDM wants to generate new prospective power profiles for the negotiation.
- Send measurements, estimates and status updates when requested by the PDM.
- Send alerts (or events) generated by a component controller to the PDM, for example due to a fault detected on a component that will require a new power schedule to be generated.

![Fig. 2.1.2: Interactions of the PS with the PDM. The left side of the figure lists the inputs from the PDM to the PS, and the right side the outputs of the PS to the PDM](image)

Note that it is not necessary that all information issued by the PS must be sent on the middleware bus (this middleware is further described in Section 3). For example, only selected measurements are necessary for display on the user interface. As a consequence, the PS and the PDM have a dedicated, high speed (due to the involved time constants), direct communication channel.
2.2. Power decision module (PDM)

The PDM is a control and energy management system in charge of coordinating the operation of the PS components as well as the interaction with the IT side through the NM.

2.2.1. Roles

The PDM has to fulfill a rather large number of tasks. A first role is to determine and/or acquire data and information required for the correct operation of the PS. This can be further subdivided into several tasks:

- Estimate component SOC and SOH, if component controllers are unable to do it themselves. This can for example be done by counting the energy throughput of the component, or its number of hours of operation.
- Determine the microgrid dynamic security level. For example, if all storage units are empty, the microgrid security level would be lower than for a microgrid with storage units close to full capacity.
- Acquire the necessary external data, from providers such as Météo France, especially regarding current weather and forecasts, with solar radiation, wind speed, temperature, humidity, cloud coverage, etc. This data is then useful to estimate future conditions and enable scheduling resources accordingly.

The above tasks enable the PDM to have an accurate representation of the status of the PS, as well as of what its components can achieve.

Next, based on data and information mentioned in the previous paragraph and/or provided by the PS and negotiations with the NM (see the dedicated sections), the PDM must determine several sets of possible operation schedules. The duration and time resolution of these profiles (such as 5 min. over 1 hour, or 1 hour over 2 days) are variable, depending on the scheduling horizon and the expected events (e.g., sudden decrease in PV generation, or a sudden increase in IT load). The resulting process is as follows:

- First, an “ideal” (reference) operation schedule (in W) is first determined for each component (PV panels, battery, etc.), with different time resolutions and horizons (e.g., from \( t_1 \) to \( t_2 \)).
- Using this ideal schedule (or another profile suggested by the NM, after the first iteration of the negotiation), the PDM can then determine the margin of operation flexibility (in W), mainly from different load and storage use schedules. Flexibility mainly arises from different strategies regarding the use of storage units and the time shifting of loads. It can also be expected that the cost of a profile (defined as the sum of the cost of each component over the scheduling horizon, which includes replacement and maintenance costs) will increase with the required flexibility, e.g., because of the increased need for storage (for further details, see deliverable D2.4). This flexibility will also be required to account the significant power ramps (changes in power demand in W/s) resulting from sudden changes in set points at the end of each period, as well as to face unexpected events.
From these ideal profile and flexibility margins, a series of power profiles (also in W) can be generated (see Fig. 2.2.1). For example, depending on whether a source is started or stopped, different profiles may emerge, each with different characteristics. As the output of the the storage units can be anywhere between 0 and the maximum capacity, an infinite number of candidate schedules can in theory be determined. To avoid this, only a limited number of profiles will be generated.

For each generated profile, metrics are computed to help the NM evaluate and compare these profiles. These metrics include:

- A set of power values (in W) for each component over a given duration, and the sum of each element of these curves for one moment in time, corresponding to the net system load at that time.
- The cost (in €) resulting from the use of each component, as well as the impact of aging on RUL (in h) (and the resulting need for investments in equipment replacement). The cost resulting from losses, e.g., in converters, will be neglected to avoid overly complexifying the models.
- A metric related to the security level of the microgrid and how the considered profile will enable following longer term trends (e.g., with respect to the state of charge of storage at the end of the profile).
- An “expiration time” (in minutes), after which a new profile must be computed (a few minutes or hours, depending on the reliability of forecasts and changing load or generation conditions), after which a new one has to be computed.
- Depending on results of investigations in other work packages, additional metrics may be added, such as a quantification of the confidence in a proposed profile.

The generation of these profiles (also called scheduling or commitment) is based on optimization algorithms, and will be further described in WP4.

![Fig. 2.2.1: Sample (fictitious) profiles generated by the PDM.](image-url)
After the negotiation described in a subsequent section has reached an agreement between the IT and the power sides, another role of the PDM is to implement the operation schedule of the microgrid components so they can supply the load. To do so, the schedule selected during the negotiation is split into several parts (e.g., one part per second) where the power is assumed constant. The output of each component is then determined by a dispatching algorithm, which can rely on an optimization algorithm (the focus of WP4) or another method. The sum of the output of each component in the system should be equal to the scheduled power that serves the load.

Finally, in terms of interactions with the local and remote users, the PDM should also interact with the local microgrid operator, through a user interface (designed as part of WP5). This could be to inform of component aging status, premature aging, preventive or required maintenance, etc. It could also be to receive commands from the operator and have them implemented by the PS, such as reconfiguration commands, e.g., to temporarily disconnect a source for maintenance.

2.2.2 Interactions with other modules

The PDM tasks are expected to operate at various frequencies, depending on the tasks at hand. Some must be run several times an hour, other perhaps only every few days. A new run of the PDM may be triggered by any of the following conditions:

1. If an event (see examples below) is detected by the PDM or by the PS, and the system is unable to keep operating adequately with the current profile.
2. If the NM is requesting new candidate profiles.
3. If the currently used profile is outdated, and a new one needs to be computed. In other words, the “wait” period of the NM is over and new states need to be computed through a negotiation.

A large number of events and alerts can arise during operation, including:

- Unexpected demand (significantly different from values agreed with the NM).
- SOC and SOH bounds are reached.
- Component startup/shutdown process began / is completed.
- Abnormal measurements (values out of acceptable range, conflicting values, etc.).
- Abnormally / unexpectedly high rate of change of a quantity (e.g., temperature).
- Protections / relays status changed (e.g., fault or maintenance).
- Weather forecasts were updated.

If any of the above conditions is met, the PDM process is as described below:

1. While no agreement is found with the ITDM (through the NM):
   a. The PDM collects measurements and information from the PS and external sources.
   b. From these, the PDM computes a series of power profiles (or schedules) with different time horizons, and with the corresponding metrics.
   c. The profiles are sent to the NM, which then evaluates these profiles and interacts similarly with the ITDM.
d. If the NM manages to reach an agreement on a profile that fits both the PDM and the ITDM expectations, and the expiration date of the profile is not reached, the selected profile is accepted and the loop ends. The algorithm used by the NM will be further developed in WP4, and may either select one of the proposed profiles, or create a composite profile from several ones.

e. If no agreement is reached with the ITDM, the next iteration of the loop is run, and new profiles are generated. These profiles can have a higher cost and would degrade the components more. The NM should however return additional information to PDM so it can generate adequate profiles. For example, this could include the average, min. and max. deviation between the best curves. In other words, the NM should give the PDM some hints regarding how it should improve its profiles. This can for example be done by sending a full profile.

2. The agreed upon profile is implemented by the PDM and the PS.
   a. The profile or schedule is split into several parts.
   b. Each part is successively sent through a dispatching algorithm that determines the output of each component.
   c. The corresponding setpoints are then sent to the PS.

The above instructions are summarized in Fig. 2.2.2.

Fig. 2.2.2: Interactions of the PDM with the PS and the NM. The left side shows the inputs from the PS (top) and the NM (bottom) to the PDM, and the right side the outputs of the PDM to the PS and the NM. External data input is visible in the center top.
2.3. IT decision module (ITDM)

The ITDM is the equivalent of the PDM on the IT side. It is in charge of managing the IT load in interaction with the ITS and the NM.

2.3.1. Roles

A first role of the ITDM is to determine and/or acquire data and information required for the correct operation of the ITS. This not only requires measuring the current IT load (resulting from users requests), in terms of tasks and power, but also forecasting the IT load for the future.

Based on the current and future IT loads and negotiations with the PDM through the NM, the ITDM must determine a set of possible operation schedules, that will result from the schedule plan of the different tasks the servers have to fulfill.

Next, based on data and information mentioned in the previous paragraph and/or provided by the ITS and negotiations with the NM (see the dedicated sections), the ITDM must determine several sets of possible operation schedules. The duration and time resolution of these profiles (such as 5 min. over 1 hour, or 1 hour over 2 days) are variable, depending on the scheduling horizon and the expected events (e.g., sudden decrease in PV generation, or a sudden increase in IT load). The resulting process is as follows:

- A series of power profiles (in W) is generated from the IT tasks at hand and the forecast of arriving tasks. For example, depending on whether a task is stopped and postponed or not, different profiles may emerge, each with different characteristics. As the number of tasks can be large, the number of candidate profiles may also be large. To avoid this, only a limited number of profiles will be generated.
- For each generated profile, metrics are computed to help the NM evaluate and compare these profiles. These metrics include:
  - A measure of the quality of service (QoS) of the profile, for example in the form of a real value between 0 and 1.
  - The “confidence” (% of expected error) on the profile. In other words, a way of determining whether the confidence in the predicted load is high or not.
  - An “expiration time”, after which a new profile must be computed (a few minutes or hours), after which a new one has to be computed, for example because the tasks to be run will have significantly changed. This specifies the maximum time by which the NM should answer for the negotiation.
  - The cost (€) resulting from this schedule and the corresponding QoS, as the way the tasks are scheduled may result in penalties, e.g., if the maximum time for completing a task has been exceeded.
  - Other indicators representing the quality of the profile.

The generation of these profiles (also called scheduling or commitment) is based on optimization algorithms, and will be further described in WP4.
After the negotiation described in a subsequent section has reached an agreement between the IT and the power sides, another role of the ITDM is to manage the implementation of the operation schedule of the tasks. To do so, the schedule selected during the negotiation is split into several parts (e.g., one part per second) where the power is assumed constant. The schedule for each task is then implemented by the IT infrastructure following the commands of the ITDM, where the total power consumed by the servers is equal to the power supplied by the microgrid.

Finally, in terms of interactions with the local and remote users, the ITDM should also interact with the local datacenter operator, through a user interface (designed as part of WP5). This could be to inform of component failure, preventive or required maintenance, new tasks, etc. It could also be to receive commands from the operator and have them implemented by the PS, such as reconfiguration commands, e.g., to temporarily stop a rack for maintenance.

2.3.2. Interactions with other modules

The ITDM tasks are expected to operate at various (time) frequencies, depending on the tasks at hand. Some must be run several times an hour, other perhaps only every few days. A new run of the ITDM may be triggered by any of the following conditions:

1. If an event of type I or E (see below) is detected, and the system is unable to keep operating with a sufficient QoS for the current profile.
2. If the NM is requesting new profiles.
3. If the current profile is outdated, and a new one needs to be computed. In other words, the “wait” period of the NM is over and new states need to be computed through a negotiation.

Several types of events and alerts can arise during operation, including:

- Type E, from the environment to the ITDM:
  - task arrival
  - task started (taskID)
  - task completed (taskID)
  - host overload (hostID)
  - host underload (hostID)
  - host start started (hostID)
  - host start completed (hostID)
  - host halt started (hostID)
  - host halt completed (hostID)
  - rack start started (rackID)
  - rack start completed (rackID)
  - rack halt started (rackID)
  - rack halt completed (rackID)

- Type I:
  - from the IT infrastructure to the ITDM:
- migration started (taskID)
- migration completed (taskID)
- power disruption (set of hostID)

○ From the ITDM to the IT infrastructure:
  - launch task (task characteristics (vCPU, RAM), taskID, hostID)
  - suspend task (taskID, hostID)
  - migrate task (taskID, host1, host2)
  - start host (hostID) (by default, a machine starts in full speed)
  - setPState (hostID, state) (e.g., suspend to RAM, halt)
  - setCState (hostID, level) (frequency level)

● Type N from the NM, e.g., to start a new negotiation.

If any of the above conditions is met, the ITDM process is as described below.

1. While no agreement is found with the PDM:
   a. The ITDM updates its internal state, by collecting the task requirements and information from the ITS.
   b. From these, the ITDM computes a series of power profiles with different time horizons, with the corresponding metrics.
   c. The profiles are sent to the NM, which then evaluates these profiles and interacts similarly with the PDM. The NM may send the NM hints regarding what profile characteristics could help the negotiation process.
   d. If the NM manages to reach an agreement on a profile that fits both the PDM and the ITDM expectations, and the expiration date of the profile is not reached, the selected profile is accepted and the loop ends. The algorithm used by the NM will be further developed in WP4, and may either select one of the proposed profiles, or create a composite profile from several ones.
   e. If no agreement is reached with the PDM, the next iteration of the loop is run, and new profiles are generated. These profiles can have a higher cost and would degrade the QoS. The NM may need to return additional information to ITDM so it can generate adequate profiles. For example, this could include the average, min. and max. deviation between the best curves. In other words, the NM should give the ITDM some hints regarding how it should improve it profiles.

2. The agreed upon profile is implemented by the ITDM and the ITS.

The above instructions are summarized in Fig. 2.3.1.
2.4. Negotiation module (NM)

The NM is in charge of handling negotiations between the power and IT sides. Its goal is to reach a decision on what each side does, in terms of IT and power profiles.

2.4.1. Roles

The main role of the NM is to conduct the negotiation between the PDM and the ITDM. The final result of the negotiation should be a profile that both the ITDM and the PDM should implement. The profile will last for a given duration, and will be implemented until its end or until an event requires a new negotiation.

To conduct the negotiation, the NM must interact with the PDM and ITDM to collect their proposed profiles. From the proposed profiles, the negotiation should select an existing profile (or generate a composite one) if possible, or, if no profile satisfies the given set of criteria (perhaps to minimize the overall operation cost, while considering quality of service and other constraints), ask for new sets of profiles to the PDM and the ITDM. Several
aspects still need to be further investigated in the dedicated tasks of WP4, for example regarding the type of algorithm to use (auctions, game theory, optimization, etc.) and the metrics to consider for comparison (a single one, a composite metric, etc.).

2.4.2. Interactions with other modules

The main steps of the NM operation are as follows:

1. The NM receives a negotiation request from the ITDM or the PDM. This request can result from the fact that an existing profile is outdated, or that an alert has occurred.
2. While a satisfying solution is not found or the maximum number of iterations is not reached:
   a. The PDM and ITDM send a set of profiles to the NM.
   b. The negotiation is then run. The NM can send hints regarding profile properties it would like to be sent by the PDM and/or the ITDM.
   c. If a profile that enables reaching an agreement is found, the loop ends.
   d. Otherwise, the PDM and ITDM are informed that new profiles are required and a new loop iteration is run.
3. The selected profile is sent to the PDM and ITDM for implementation. In the worst case, if no agreement was found after the negotiation, a "last resort" profile is used to avoid a blackout. The definition of this last resort profile will be further investigated in WP4.

These interactions are summarized in Fig. 2.4.1.

Fig. 2.4.1: Interactions of the NM with the PDM and the ITDM. The left side of the figure shows the inputs from the PDM and the ITDM to the NM, and the right side the outputs of the NM to the PDM and the ITDM
2.5. IT system (ITS)

The IT system or infrastructure consists of racks of machines (or hosts) connected to the same network, and supplied by the PS. These machines process the tasks issued by the datacenter users.

2.5.1. Description

The tasks issued by the users must be processed with the machines through the use of processors. In order to modulate the IT load for the negotiation, several actions can be taken. First, machines and racks can be started or stopped. Tasks can be then be moved from one machine to another or temporarily suspended. The frequency of a machine can also be changed to impact the processing speed and power consumption. These actions are decided by the ITDM and implemented by the ITS. Events and alerts are generated by the ITS whenever the tasks and the host configuration changes (see a list in section 2.3.2).

The ITS can be either a real IT infrastructure run by OpenStack, or a simulated infrastructure, for example simulated in the DCWorms or simGrid simulator.

2.5.2. Interactions with other modules

Based on the above, the ITS interacts only with the ITDM, and not with the other modules. Its actions with respect to the ITDM are to (see Fig. 2.5.1):

- Receive orders from the ITDM, for example to start or stop a machine, change a machine frequency, or start a task. These orders result from the implementation of an IT profile decided through the negotiation.
- Send events and alerts notifications to the PDM, for example when a machine has started or when a new task has arrived. These alerts and events may be related to tasks, hosts and the infrastructure.

![Fig. 2.5.1: Interactions of the ITS and the ITDM. The left side of the figure shows the inputs from the ITDM to the ITS, and the right side the outputs from the ITDM to the ITS](image-url)
3. Messages format

In this part, we focus on designing the format of the messages that are exchanged between the different modules. Contrary to the previous part, we do not focus on the contents of the messages, but rather on how they are structured and processed.

3.1. Middleware

In order to enable modules to communicate with each other by exchanging messages, a communication channel must be created. As shown in Fig. 3.1, a large number of interactions exist in the system, not only between the modules but also with other subsystems such as the graphical user interface. These other subsystems will be further described in WP5.

Fig. 3.1: An overview of the message exchanges between the different system modules and other systems
To handle these messages, a message oriented middleware is used. This is done using ActiveMQ, an open source messaging server (or message broker), which supports a large number of clients, protocols and languages. ActiveMQ enables a large number of producers and consumers of information to communicate using topics. These topics allow each process to receive only the messages it needs with a low configuration cost. ActiveMQ also provides persistence and coherence services.

3.2. Messages structure

To maintain a neutral representation of the data, thus enforcing the openness of the platform, the format and structure of the messages exchanged through the middleware are based on the JSON format. JSON, for JavaScript Object Notation, is an open standard data interchange format. It uses human readable text to transmit data objects. Each object consists of attribute-value pairs, i.e., each object includes a series of attributes, and each attribute has a corresponding value. JSON is independent from the language.

As a consequence, messages formatted in JSON can be exchanged between different modules, even if these modules are written in different languages (e.g., Java and C++, as in Fig. 3.2). This ensure the interoperability of the different modules, as long as they are able to send and receive messages and process them to and from JSON through dedicated classes (toJson and fromJson, respectively).

![Fig. 3.2: Example of use of JSON to transmit data through messages over ActiveMQ](image)

For example, messages describing the activity of an (IT) machine have the JSON format shown below:

```
"machineActivityMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:MachineActivityMessageContents",
    "properties" : {
        "idMachine" : {
            "type" : "string"
        },
        "machineState" : {
            "type" : "string",
            "enum" : [ "MACHINE_OFF", "MACHINE_ON", "MACHINE_SHUTDOWN",
                       "MACHINE_START", "MACHINE_UNREACHABLE" ]
        }
    }
}
```
This format shows that the machine is described by an object, with several properties (identifier, state, frequency, power used, etc.). The first type of message is used to transmit information on the current status of the machine (i.e., is it on or not, what is its frequency of operation, etc.). The second type of message can be used to change the state of a machine. Similarly, the third type of message can be used to change the frequency of a machine.

The corresponding Java classes are as follows:

```java
package activity.it.resources;
public class MachineActivity extends DataZeroObject {
    MachineDescription managedMachine;
    JobDataZero [] currentJobs;
    int nbCores;
    int indCurrentFrequency;
}
```
Similar formats were built for datacenter resources (IT and power) description and datacenter activity (IT, power, negotiation, logs, etc.). Additionally, specific data structures were created, for example a generic DATAZERO object, a time stamped array, a profile, etc. These structures are further described in the next section.

Note that each message type is identified with a topic, which is closely related to the message name. For example, the topic for the above message example is “IT_MACHINE_STATE”. This topic is useful to identify which type of message is received and then decide how to process its contents. The topic attached to each type of message is also listed in the next section.

3.3. List of messages

The list of expected messages is available in the Table 3.3.1 below. For each message, a domain, a message type, a topic, an object (a description) and a producer are specified. This enables defining what each message is intended for and how to identify each type of message accurately.

<table>
<thead>
<tr>
<th>Domain</th>
<th>Message</th>
<th>Topic</th>
<th>Object</th>
<th>Producer</th>
</tr>
</thead>
<tbody>
<tr>
<td>description.it</td>
<td>DataCenterITDescriptionMessageContents</td>
<td>IT_DESC_DC</td>
<td>IT description IT of the DC</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td>MachineDescriptionMessageContents</td>
<td>IT_DESC_MACHINE</td>
<td>Description of a machine</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td>MessageContents</td>
<td>Key</td>
<td>Description</td>
<td>Related System</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----</td>
<td>-------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>RackDescriptionMessageContents</td>
<td>IT_DESC_RACK</td>
<td>Description of a machine rack</td>
<td>OpenStack / Simulator</td>
<td></td>
</tr>
<tr>
<td>DataCenterPowerDescriptionMessageContents</td>
<td>ELEC_DESC_DC</td>
<td>Electrical description of the DC</td>
<td>PDM</td>
<td></td>
</tr>
<tr>
<td>BatteryDescriptionMessageContents</td>
<td>ELEC_DESC_BATTERY</td>
<td>Description of a battery</td>
<td>PS: battery</td>
<td></td>
</tr>
<tr>
<td>WindDescriptionMessageContents</td>
<td>ELEC_DESC_WIND</td>
<td>Description of a wind turbine</td>
<td>PS: wind turbines</td>
<td></td>
</tr>
<tr>
<td>PhotoVoltaicDescriptionMessageContents</td>
<td>ELEC_DESC_VOLTAIC</td>
<td>Description photovoltaic generator</td>
<td>PS: solar panels</td>
<td></td>
</tr>
<tr>
<td>GridDescriptionMessageContents</td>
<td>ELEC_DESC_GRID</td>
<td>Description of the grid</td>
<td>PDM</td>
<td></td>
</tr>
<tr>
<td>DataCenterDescriptionMessageContents</td>
<td>DZ_DESC_DC</td>
<td>IT and electrical description of the DC</td>
<td>DZ Supervisor</td>
<td></td>
</tr>
<tr>
<td>MachineActivityMessageContents</td>
<td>IT_MACHINE_STATE</td>
<td>Information on the machine activity</td>
<td>ITDM</td>
<td></td>
</tr>
<tr>
<td>RackActivityMessageContents</td>
<td>IT_RACK_STATE</td>
<td>Information on the rack activity</td>
<td>ITDM</td>
<td></td>
</tr>
<tr>
<td>DataCenterITActivityMessageContents</td>
<td>IT_DC_STATE</td>
<td>Information on the IT activity of the DC</td>
<td>ITDM</td>
<td></td>
</tr>
<tr>
<td>MachineChangeFrequencyMessageContents</td>
<td>IT_MACHINE_CHANGE_FREQUENCY</td>
<td>Machine frequency change order</td>
<td>ITDM</td>
<td></td>
</tr>
<tr>
<td>MachineChangeStateMessageContents</td>
<td>IT_MACHINE_CHANGE_STATE</td>
<td>Machine start order</td>
<td>ITDM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Information that the machine is ON</td>
<td>OpenStack / Simulator</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Machine stop order</td>
<td>ITDM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Information that the machine is OFF</td>
<td>OpenStack / Simulator</td>
<td></td>
</tr>
<tr>
<td>RackChangeStateMessageContents</td>
<td>IT_RACK_CHANGE_STATE</td>
<td>Rack start order</td>
<td>ITDM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Information that the rack is ON</td>
<td>OpenStack / Simulator</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rack stop order</td>
<td>ITDM</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Information that the rack is off</td>
<td>OpenStack / Simulator</td>
<td></td>
</tr>
<tr>
<td>FlavorListMessageContents</td>
<td>IT_FLAV_LIST</td>
<td>Information on the list of existing flavors (compute, memory, and storage capacity of jobs)</td>
<td>OpenStack / Simulator</td>
<td></td>
</tr>
<tr>
<td>FlavorCreateMessageContents</td>
<td>IT_FLAV_CREATE</td>
<td>Flavor creation order</td>
<td>ITDM</td>
<td></td>
</tr>
</tbody>
</table>

A flavor defines the compute, memory, and storage capacity of a virtual machine. In Datazero, this concept is also used to represent the needs of a job in terms of CPU, memory and disk.
<table>
<thead>
<tr>
<th>Message Contents</th>
<th>IT_XX</th>
<th>Description</th>
<th>Source/Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlavorDeleteMessageContents</td>
<td>IT_FLAV_DEL</td>
<td>Flavor deletion order</td>
<td>ITDM</td>
</tr>
<tr>
<td>FlavorListRequestMessageContents</td>
<td>IT_FLAV_LIST_REQUEST</td>
<td>Request list of existing flavors</td>
<td>ITDM</td>
</tr>
<tr>
<td>JobDataZeroArrivalMessageContents</td>
<td>IT_JOB_ARRIVAL</td>
<td>Information on taking into account a new job by the DZ system</td>
<td>ITDM</td>
</tr>
<tr>
<td>JobPlacementMessageContents</td>
<td>IT_JOB_PLACEMENT</td>
<td>Job placement order (initial or migration)</td>
<td>ITDM</td>
</tr>
<tr>
<td>JobChangeStateMessageContents</td>
<td>IT_JOB_CHANGE_STATE</td>
<td>Stopped job activation order: RUNNING</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Information job outgoing: OUTGOING</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Information job incoming: INCOMING</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Job stop order: STOPPED</td>
<td>ITDM</td>
</tr>
<tr>
<td>JobConsumptionOrTerminationMessageContents</td>
<td>IT_JOB_TERMINATE_D</td>
<td>Information on job ending, including the last resource consumption</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td>IT_JOB_CONSUMPTION</td>
<td>Information on job resources consumption</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td>JobInfosAtTerminationMessageContents</td>
<td>IT_INFOS_AFTER_JOB_EXIT</td>
<td>Overall job execution information</td>
<td>ITDM</td>
</tr>
<tr>
<td>SourceEnvironmentMessageContents</td>
<td>ELEC_ENV_PREVISATION</td>
<td>Source environment forecast (wind, solar radiation, …)</td>
<td>Retrieved from external sources (Internet) or produced by the PDM</td>
</tr>
<tr>
<td>SourceProductionMessageContents</td>
<td>ELEC_SOURCE_PRODUCTION</td>
<td>Source output</td>
<td>PS: each source</td>
</tr>
<tr>
<td>SourceChangeStateMessageContents</td>
<td>ELEC_SOURCE_CHANGE_STATE</td>
<td>Source start order: SOURCE_START</td>
<td>PDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Information that a source is ON: SOURCE_ON</td>
<td>PS: each source</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Source stop order: SOURCE_SHUTDOWN</td>
<td>PDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Information that a source is OFF: SOURCE_OFF</td>
<td>PS: each source</td>
</tr>
<tr>
<td>SourceChangePowerMessageContents</td>
<td>ELEC_SOURCE_CHANGE_POWER</td>
<td>Source power change order</td>
<td>PDM</td>
</tr>
<tr>
<td>SourceGetDiagnosticMessageContents</td>
<td>ELEC_SOURCE_GET_DIAG</td>
<td>Source diagnostic send request</td>
<td>PDM</td>
</tr>
<tr>
<td>SourceDiagnosticMessageContents</td>
<td>ELEC_SOURCE_DIAG</td>
<td>A source sends its diagnostic</td>
<td>PS: each source</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------</td>
<td>-------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>activity.power</td>
<td>DataCenterPowerActivityMessageContents</td>
<td>ELEC_DC_STATE</td>
<td>Information on the electrical activity of the DC</td>
</tr>
<tr>
<td>activity.datacenter</td>
<td>DataCenterActivityMessageContents</td>
<td>DZ_DC_STATE</td>
<td>Information on the IT and electrical activity of the DC</td>
</tr>
<tr>
<td>activity.negotiation</td>
<td>ProfileSetMessageContents</td>
<td>PDM_TO_NEGO</td>
<td>Sending of power profiles to NEGO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ITDM_TO_NEGO</td>
<td>Sending of IT profiles to NEGO</td>
</tr>
<tr>
<td>NegotiationResponseToITDMMessageContents</td>
<td>NEG0_TO_ITDM</td>
<td>Sending of a response to ITDM</td>
<td>NEGO</td>
</tr>
<tr>
<td>NegotiationResponseToPDMMessageContents</td>
<td>NEG0_TO_PDM</td>
<td>Sending of a response to PDM</td>
<td>NEGO</td>
</tr>
<tr>
<td>ProfileRequestMessageContents</td>
<td>NEG0_REQUEST_TO_ITDM</td>
<td>NEGO requests IT profiles to ITDM</td>
<td>NEGO</td>
</tr>
<tr>
<td>ProfileSetMessageContents</td>
<td>NEG0_REQUEST_TO_PDM</td>
<td>NEGO requests power profiles to PDM</td>
<td>NEGO</td>
</tr>
<tr>
<td>log</td>
<td>LogRequestMessageContents</td>
<td>LOG_REQUEST</td>
<td>Log extraction request</td>
</tr>
<tr>
<td></td>
<td>LogResponseSetMessageContents</td>
<td>LOG_RESPONSE</td>
<td>Messages extracted corresponding to the request</td>
</tr>
<tr>
<td>debug</td>
<td></td>
<td>DZ_FINISH</td>
<td>A topic for clean application finish</td>
</tr>
<tr>
<td>stringMessageContents</td>
<td>DZ_STRING1</td>
<td>Topics to exchange messages of type String</td>
<td>All</td>
</tr>
<tr>
<td></td>
<td>DZ_STRING2</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DZ_STRING3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3.1: List of messages with attached topics

An overview of the messages exchanged between the PDM, the NM and the ITDM is shown in Fig. 3.2. The negotiation process described in section 2 would start with a request for profiles sent by the NM to the PDM and the ITDM, which would then respond with the required profiles. Depending on these profiles, the NM may then request other profiles, and receive them from the PDM and the ITDM. The process would continue until the end of the negotiation.
The messages listed in Table 3.3.1 result from the interactions between the modules described in section 2. Activity messages, in particular, correspond to most of these interactions.

Note that several types of messages appear in the table and were not discussed earlier. This includes messages for logging (e.g., saving a value for later use or display, or retrieving a past value), for debugging, and for describing the IT and electrical equipment used in the datacenter (required so that each IT or electrical component can signal its existence and inform other modules of its characteristics).

The detailed content of each message, with the JSON format and the corresponding Java class, is described in the Appendix 1. These formats are similar to the examples provided in section 3.2.
4. Next steps

This deliverable has described the structure of the DATAZERO system, with its different modules, their roles and their interactions. The format of the messages exchanged between these modules were also defined. This work is expected to serve as a basis for work packages 4 and 5, in which, respectively, the optimization and negotiation algorithms will be developed, and the operation of the proposed system will be tested in both simulation and experimental validation. Additionally, deliverable 3.2, due in 6 months, will focus on aggregation issues on both power and IT sides, and will complement this document as well as ongoing work in WP4 and WP5.

While this deliverable describes the current state of development of the proposed system, it may be necessary to adjust the above contents or reconsider specific aspects. These adjustments may appear as WP4 and WP5 progress, and such changes seem necessary.
Appendix 1: Description of messages
DataZero messages

Contents

1. Goals........................................................................................................................................3
   1.1. Modeling the DataCenter ................................................................................................3
   1.2. From classes to Json schemas ......................................................................................3
   1.3. Topics .................................................................................................................................3

2. Common data structures.........................................................................................................3
   2.1. DataZeroObject.java ......................................................................................................3
   2.2. DataZeroObjectSet.java ................................................................................................4
   2.3. TimeStampedArray.java ..................................................................................................4
   2.4. Profile.java .....................................................................................................................5

3. "Message" Data Structures ..................................................................................................5
   3.1. DataZeroMessageContents.java ...................................................................................5
   3.2. MessageContents.java ................................................................................................6

4. Datacenter Description ..........................................................................................................6
   4.1. IT resources ....................................................................................................................6
       4.1.1. Machine ..................................................................................................................6
           4.1.1.1. MachineState.java ........................................................................................6
       4.1.2. Rack .........................................................................................................................8
           4.1.2.1. RackState.java ...............................................................................................8
           4.1.2.2. RackDescription.java et RackDescriptionMessageContents.java ...................8
       4.1.3. Flavor.java ...............................................................................................................9
       4.1.4. DataCenterITDescription.java et DataCenterITDescriptionMessageContents.java ....10
5. DataCenter activity ............................................................................................................. 19

5.1. IT side ............................................................................................................................ 19

5.1.1. IT resources .............................................................................................................. 19

5.1.1.1. Machine .................................................................................................................. 19

5.1.1.2. Rack ......................................................................................................................... 20

5.1.1.3. Flavor ....................................................................................................................... 21

5.1.1.4. DataCenterIT ............................................................................................................ 22

5.1.2. Jobs .............................................................................................................................. 23

5.2. Electric side ................................................................................................................... 27

5.2.1. The environment of the electric sources .................................................................... 27

5.2.2. Source production .................................................................................................... 28

5.2.3. Electricity Management in the DataCenter ............................................................... 30

5.3. Datacenter ...................................................................................................................... 31

5.4. Negotiation .................................................................................................................... 31

5.4.1. Dedicated profiles .................................................................................................... 31

5.4.2. A negotiation process ............................................................................................. 33

5.5. The system log ............................................................................................................... 37

5.6. For tests, different topics and one message .................................................................. 39

5.7. Topics and messages .................................................................................................... 40
1. Goals

1.1. Modeling the DataCenter
From the elements provided in the WorkPackages, propose Java classes that describe the components of the DC and its activity.

1.2. From classes to Json schemas
From these Java classes, use the methods provided by the Jackson package to generate the Json schemas that formally describe the messages that will be exchanged.

This description of the messages could be used to deduce the types of data that would correspond in other languages for the consistency of analysis of exchanged messages and / or the definition of classes in other languages (C ++, ...).

Messages exchanged via activemq will be generated via toJson function and rebuilt upon arrival in objects via a fromJson function.

1.3. Topics
Topics are proposed from the supposed activity of the DataCenter.

2. Common data structures

2.1. DataZeroObject.java

Classes

```java
package common;

public class DataZeroObject {
    public void print();
    public String toJson();
    public static String toJson(DataZeroObject messageContentObj);
    public static DataZeroObject fromJson(String jsonString, Class<?> messageContentClass);
    public String objectId;
    public boolean isSame(String idOther);  
    public DataZeroObject findDataZeroObjectInArray(String searchedId,
                                                     DataZeroObject [] objectArray);
    public static void printDataZeroArrayObject(DataZeroObject [] objectArray);
}
```
Json Schema

"dataZeroObject" : {
  "type" : "any"
}

2.2. DataZeroObjectSet.java

Classes

```java
package common;

public class DataZeroObjectSet<DataZeroType extends DataZeroObject> extends DataZeroObject {
  ArrayList<DataZeroType> contents;
}
```

Json Schema

"dataZeroObjectSet" : {
  "type" : "object",
  "id" : "urn:jsonschema:common:DataZeroObjectSet",
  "properties" : {
    "contents" : {
      "type" : "array",
      "items" : {
        "type" : "any"
      }
    }
  }
}

2.3. TimeStampedArray.java

Classes

```java
package common;

public class TimeStampedArray extends DataZeroObject {
  protected long startTime;
  protected long byStep;
  protected float [] data;
}
```

Json Schema

"timeStampedArray" : {
  "type" : "object",
  "id" : "urn:jsonschema:common:TimeStampedArray",
  "properties" : {
    "startTime" : {
      "type" : "integer"
    },
    "byStep" : {
      "type" : "integer"
    },
    "data" : {
      "type" : "array",
      "items" : {
        "type" : "number"
      }
    }
  }
}
2.4. Profile.java

Classes

```java
package common;

public class Profile extends TimeStampedArray {
    protected float quality;
    protected float confidence;
    protected long outDatedAt;
}
```

Json Schema

```
"profile" : {
    "type" : "object",
    "id" : "urn:jsonschema:common:Profile",
    "properties" : {
        "startTime" : {
            "type" : "integer"
        },
        "byStep" : {
            "type" : "integer"
        },
        "data" : {
            "type" : "array",
            "items" : {
                "type" : "number"
            }
        },
        "quality" : {
            "type" : "number"
        },
        "confidence" : {
            "type" : "number"
        },
        "outDatedAt" : {
            "type" : "integer"
        }
    }
}
```

3. "Message" Data Structures

3.1. DataZeroMessageContents.java

Classes

```java
package message;

public class DataZeroMessage extends DataZeroObject {
    protected long timeStamp;
    protected DataZeroObject value;

    public DataZeroMessage(DataZeroObject oneDataZeroObject);
    public DataZeroMessage(long timeStamp, DataZeroObject oneDataZeroObject);
    public DataZeroMessage(String JsonMessageValue, Class<?> messageContentClass);
    public DataZeroMessage(String JsonMessageValue, String topicValue);
    public static long timeStampValue(String JsonMessageValue);
    public static String messageContents(String JsonMessageValue);
    public long getTimeStamp();
    public void setTimeStamp(long timeStamp);
    public DataZeroObject getValue();
    public void setValue(DataZeroObject value);
    public void print();
}
```
Json Schema

"dataZeroMessage" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:DataZeroMessage",
    "properties" : {
        "timeStamp" : {
            "type" : "integer"
        },
        "value" : {
            "type" : "any"
        }
    }
}

3.2. MessageContents.java

package message;
public class MessageContents extends DataZeroObject{
    public String buildJsonStringMessageContents(long timeStamp);
    public String buildJsonStringMessageContents();
}

4. Datacenter Description

4.1. IT resources

4.1.1. Machine

4.1.1.1. MachineState.java

Classes

package description.it;

public enum MachineState {
    MACHINE_OFF,
    MACHINE_ON,
    MACHINE_SHUTDOWN,
    MACHINE_START,
    MACHINE_UNREACHABLE
}

Json Schema

"machineState" : {
    "type" : "string",
    "enum" : ["MACHINE_OFF","MACHINE_ON","MACHINE_SHUTDOWN",
              "MACHINE_START","MACHINE_UNREACHABLE"]
}
4.1.1.2. MachineDescription.java – MachineDescriptionMessageContents.java

Classes

```java
package description.it;

public class MachineDescription extends DataZeroObject {
    private String idMachine;
    MachineState machineState;
    private int numberOfCores;
    private float memory;
    private float idealMinPower;
    private float coeffAlpha;
    private float[] frequencies;
    private float[] powerMax;
}
```

```java
package message;

public class MachineDescriptionMessageContents extends MessageContents {
    MachineDescription machineDescription;
}
```

Json Schema

```json
"machineDescription" : {
    "type" : "object",
    "id" : "urn:jsonschema:description:it:MachineDescription",
    "properties" : {
        "idMachine" : {
            "type" : "string"
        },
        "machineState" : {
            "type" : "string",
            "enum" : ["MACHINE_OFF", "MACHINE_ON", "MACHINE_SHUTDOWN",
                      "MACHINE_START", "MACHINE_UNREACHABLE"]
        },
        "numberOfCores" : {
            "type" : "integer"
        },
        "memory" : {
            "type" : "number"
        },
        "idealMinPower" : {
            "type" : "number"
        },
        "coeffAlpha" : {
            "type" : "number"
        },
        "frequencies" : {
            "type" : "array",
            "items" : {
                "type" : "number"
            }
        },
        "powerMax" : {
            "type" : "array",
            "items" : {
                "type" : "number"
            }
        }
    }
}
```
"machineDescriptionMessageContents" : {
    "type": "object",
    "id": "urn:jsonschema:message:MachineDescriptionMessageContents",
    "properties": {
        "machineDescription" : {
            "type": "object",
            "$ref": "urn:jsonschema:description:it:MachineDescription"
        }
    }
}

4.1.2. Rack

4.1.2.1. RackState.java

Classes

package description.it;
public enum RackState {
    RACK_OFF,
    RACK_ON,
    RACK_SHUTDOWN,
    RACK_START,
    RACK_UNREACHABLE
}

Json Schema

"rackState" : {
    "type": "string",
    "enum": ["RACK_OFF", "RACK_ON", "RACK_SHUTDOWN", "RACK_START", "RACK_UNREACHABLE"]
}

4.1.2.2. RackDescription.java et RackDescriptionMessageContents.java

Classes

package description.it;
public class RackDescription extends DataZeroObject {
    String idRack;
    RackState rackState;
    MachineDescription [] machines;
    float powerBase;
}

package message;
public class RackDescriptionMessageContents extends MessageContents {
    RackDescription rackDescription;
}

Json Schema

"rackDescription" : {
    "type": "object",
    "id": "urn:jsonschema:description:it:RackDescription",
    "properties": {
        "idRack" : {
            "type": "string"
        },
        "rackState" : {
            "type": "string",
            "enum": [ "RACK_OFF", "RACK_ON", "RACK_SHUTDOWN", "RACK_START", "RACK_UNREACHABLE" ]
        }
    }
}
"machines" : {
  "type" : "array",
  "items" : {
    "type" : "object",
    "$ref" : "urn:jsonschema:description:it:MachineDescription"
  }
},
"powerBase" : {
  "type" : "number"
}
}
"rackDescriptionMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:RackDescriptionMessageContents",
  "properties" : {
    "rackDescription" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:description:it:RackDescription"
    }
  }
}

4.1.3. Flavor.java

Classes
package description.it;
public class Flavor extends DataZeroObject {
    private String idFlavor;
    private String flavorName;
    private int vcpu;
    private int disk;
    private int ram;
}

Json Schema
"flavor" : {
  "type" : "object",
  "id" : "urn:jsonschema:description:it:Flavor",
  "properties" : {
    "idFlavor" : {
      "type" : "string"
    },
    "flavorName" : {
      "type" : "string"
    },
    "vcpu" : {
      "type" : "integer"
    },
    "disk" : {
      "type" : "integer"
    },
    "ram" : {
      "type" : "integer"
    }
  }
}
4.1.4. DataCenterITDescription.java et DataCenterITDescriptionMessageContents.java

Classes

```java
package description.it;

public class DataCenterITDescription extends DataZeroObject {
    String idDataCenter;
    RackDescription [] racks;
    Flavor [] flavorList;
}
```

```java
package message;

public class DataCenterITDescriptionMessageContents extends MessageContents {
    DataCenterITDescription dataCenterITDescription;
}
```

Json Schema

```
"dataCenterITDescription" : {
    "type" : "object",
    "id" : "urn:jsonschema:description:it:DataCenterITDescription",
    "properties" : {
        "idDataCenter" : {
            "type" : "string"
        },
        "racks" : {
            "type" : "array",
            "items" : {
                "type" : "object",
                "$ref" : "urn:jsonschema:description:it:RackDescription"
            }
        }
    }
}

"flavorList" : {
    "type" : "array",
    "items" : {
        "type" : "object",
        "$ref" : "urn:jsonschema:description:it:Flavor"
    }
}

"dataCenterITDescriptionMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:DataCenterITDescriptionMessageContents",
    "properties" : {
        "dataCenterITDescription" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:description:it:DataCenterITDescription"
        }
    }
}
```
4.2. Electric sources

4.2.1. Common data

4.2.1.1. SourceState.java

Classes

```java
package description.power;
public enum SourceState {
    SOURCE_OFF,
    SOURCE_ON,
    SOURCE_SHUTDOWN,
    SOURCE_START,
    SOURCE_UNREACHABLE
}
```

Json Schema

```json
"sourceState" : {
    "type" : "array",
    "items" : {
        "type" : "string",
        "enum" : ["SOURCE_OFF", "SOURCE_ON", "SOURCE_SHUTDOWN",
                    "SOURCE_START","SOURCE_UNREACHABLE"]
    }
}
```

4.2.1.2. SourceType.java

Classes

```java
package description.power;
public enum SourceType {
    BATTERY,
    SOLAR,
    WIND_TURBINE,
    SUPER_CAP,
    FUEL_CELL,
    GRID,
    ALL_RENEWABLE
}
```

Json Schema

```json
"sourceType" : {
    "type" : "string",
    "enum" : ["BATTERY", "SOLAR", "WIND_TURBINE",
              "SUPER_CAP", "FUEL_CELL", "GRID", "ALL_RENEWABLE"]
}
```

4.2.1.3. AbstractSourceDescription.java

Classes

```java
package description.power;
public class AbstractSourceDescription extends DataZeroObject {
    SourceType sourceType;
    String idSource;
    String model;
    String techno;
    int nbItems;
    SourceState [] sourceState;
    String curve;
}
```
Json Schema

"abstractSourceDescription" : {
  "type" : "object",
  "id" : "urn:jsonschema:description:power:AbstractSourceDescription",
  "properties" : {
    "sourceType" : {
      "type" : "string",
      "enum" : ["BATTERY", "SOLAR", "WIND_TURBINE", "SUPER_CAP",
                 "FUEL_CELL", "GRID", "ALL_RENEWABLE"]
    },
    "idSource" : {
      "type" : "string"
    },
    "model" : {
      "type" : "string"
    },
    "techno" : {
      "type" : "string"
    },
    "nbItems" : {
      "type" : "integer"
    },
    "sourceState" : {
      "type" : "array",
      "items" : {
        "type" : "string",
        "enum" : ["SOURCE_OFF", "SOURCE_ON", "SOURCE_SHUTDOWN",
                  "SOURCE_START", "SOURCE_UNREACHABLE"]
      }
    },
    "curve" : {
      "type" : "string"
    }
  }
}

4.2.2. BatteryDescription.java and BatteryDescriptionMessageContents.java

Classes

package description.power;
public class BatteryDescription extends AbstractSourceDescription {
  float capacity;
  float maxCurrent;
  float ratedVoltage;
}

package message;
public class BatteryDescriptionMessageContents extends MessageContents {
  BatteryDescription batterydescription;
}

Json Schema

"batteryDescription" : {
  "type" : "object",
  "id" : "urn:jsonschema:description:power:BatteryDescription",
  "properties" : {
    "sourceType" : {
      "type" : "string",
      "enum" : ["BATTERY", "SOLAR", "WIND_TURBINE", "SUPER_CAP",
                "FUEL_CELL", "GRID", "ALL_RENEWABLE"]
    },
    "idSource" : {
      "type" : "string"
    },
    "model" : {
      "type" : "string"
    },
    "curve" : {
      "type" : "string"
    }
  }
}
"model" : {
  "type" : "string"
},
"techno" : {
  "type" : "string"
},
"nbItems" : {
  "type" : "integer"
},
"sourceState" : {
  "type" : "array",
  "items" : {
    "type" : "string",
    "enum" : ["SOURCE_OFF", "SOURCE_ON", "SOURCE_SHUTDOWN", "SOURCE_START", "SOURCE_UNREACHABLE"]
  }
},
"curve" : {
  "type" : "string"
},
"capacity" : {
  "type" : "number"
},
"maxCurrent" : {
  "type" : "number"
},
"ratedVoltage" : {
  "type" : "number"
}

"batteryDescriptionMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:BatteryDescriptionMessageContents",
  "properties" : {
    "batteryescription" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:description:power:BatteryDescription"
    }
  }
}

4.2.3. WindDescription.java and WindDescriptionMessageContents.java

**Classes**

```java
package description.power;

public class WindDescription extends AbstractSourceDescription {
    private float height;
    private float minSpeed;
    private float maxSpeed;
}

package message;

public class WindDescriptionMessageContents extends MessageContents {
    WindDescription windDescription;
}
```
### Json Schema

```json
"windDescription": {  
  "type": "object",  
  "id": "urn:jsonschema:description:power:WindDescription",  
  "properties": {  
    "sourceType": {  
      "type": "string",  
      "enum": ["BATTERY", "SOLAR", "WIND_TURBINE", "SUPER_CAP",  
                "FUEL_CELL", "GRID", "ALL_RENEWABLE"]  
    },  
    "idSource": {  
      "type": "string"  
    },  
    "model": {  
      "type": "string"  
    },  
    "techno": {  
      "type": "string"  
    },  
    "nbItems": {  
      "type": "integer"  
    },  
    "sourceState": {  
      "type": "array",  
      "items": {  
        "type": "string",  
        "enum": ["SOURCE_OFF", "SOURCE_ON", "SOURCE_SHUTDOWN",  
                  "SOURCE_START", "SOURCE_UNREACHABLE"]  
      }  
    },  
    "curve": {  
      "type": "string"  
    },  
    "height": {  
      "type": "number"  
    },  
    "minSpeed": {  
      "type": "number"  
    },  
    "maxSpeed": {  
      "type": "number"  
    }  
  }  
},  
"windDescriptionMessageContents": {  
  "type": "object",  
  "id": "urn:jsonschema:message:WindDescriptionMessageContents",  
  "properties": {  
    "windDescription": {  
      "type": "object",  
      "$ref": "urn:jsonschema:description:power:WindDescription"  
    }  
  }  
}
```

### 4.2.4. PhotoVoltaicDescription.java and PhotoVoltaicDescriptionMessageContents.java

#### Classes

```java
package description.power;

public class PhotoVoltaicDescription extends AbstractSourceDescription {  
  float panelArea;  
  int numberOfPanels;  
  String efficiencyCurve;  
}
```

```java
package message;
```
public class PhotoVoltaicDescriptionMessageContents extends MessageContents {
    
    PhotoVoltaicDescription photoVoltaicDescription;
}

Json Schema

"photoVoltaicDescription" : {
    "type" : "object",
    "id" : "urn:jsonschema:description:power:PhotoVoltaicDescription",
    "properties" : {
        "sourceType" : {
            "type" : "string",
            "enum" : ["BATTERY", "SOLAR", "WIND_TURBINE", "SUPER_CAP",
                       "FUEL_CELL", "GRID", "ALL_RENEWABLE"]
        },
        "idSource" : {
            "type" : "string"
        },
        "model" : {
            "type" : "string"
        },
        "techno" : {
            "type" : "string"
        },
        "nbItems" : {
            "type" : "integer"
        },
        "sourceState" : {
            "type" : "array",
            "items" : {
                "type" : "string",
                "enum" : ["SOURCE_OFF", "SOURCE_ON", "SOURCE_SHUTDOWN",
                          "SOURCE_START", "SOURCE_UNREACHABLE"]
            }
        },
        "curve" : {
            "type" : "string"
        },
        "panelArea" : {
            "type" : "number"
        },
        "numberOfPanels" : {
            "type" : "integer"
        },
        "efficiencyCurve" : {
            "type" : "string"
        }
    }
}

"photoVoltaicDescriptionMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:PhotoVoltaicDescriptionMessageContents",
    "properties" : {
        "photoVoltaicDescription" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:description:power:PhotoVoltaicDescription"
        }
    }
}
4.2.5. GridDescription.java and GridDescriptionMessageContents.java

Classes

```java
package description.power;
public class GridDescription extends AbstractSourceDescription {
    float capacity;
}
package message;
public class GridDescriptionMessageContents extends MessageContents {
    GridDescription gridDescription;
}
```

Json Schema

```json
"gridDescription" : {
    "type" : "object",
    "id" : "urn:jsonschema:description:power:GridDescription",
    "properties" : {
        "sourceType" : {
            "type" : "string",
            "enum" : ["BATTERY", "SOLAR", "WIND_TURBINE", "SUPER_CAP",
                       "FUEL_CELL", "GRID", "ALL_RENEWABLE"]
        },
        "idSource" : {
            "type" : "string"
        },
        "model" : {
            "type" : "string"
        },
        "techno" : {
            "type" : "string"
        },
        "nbItems" : {
            "type" : "integer"
        },
        "sourceState" : {
            "type" : "array",
            "items" : {
                "type" : "string",
                "enum" : ["SOURCE_OFF", "SOURCE_ON", "SOURCE_SHUTDOWN",
                          "SOURCE_START", "SOURCE_UNREACHABLE"]
            }
        },
        "curve" : {
            "type" : "string"
        },
        "capacity" : {
            "type" : "number"
        }
    }
}
"gridDescriptionMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:GridDescriptionMessageContents",
    "properties" : {
        "gridDescription" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:description:power:GridDescription"
        }
    }
}
```
4.2.6. DataCenterPowerDescription.java and DataCenterPowerDescriptionMessageContents.java

Classes

```java
package description.power;

public class DataCenterPowerDescription extends DataZeroObject {
    String idDataCenter;
    BatteryDescription [] batterySources;
    PhotoVoltaicDescription [] photoVoltaicSources;
    WindDescription [] windSources;
    GridDescription [] gridSources;
}
```

```java
package message;

public class DataCenterPowerDescriptionMessageContents extends MessageContents {
    DataCenterPowerDescription dataCenterPowerDescription;
}
```

Json Schema

```json
"dataCenterPowerDescription" : {
    "type" : "object",
    "id" : "urn:jsonschema:description:power:DataCenterPowerDescription",
    "properties" : {
        "idDataCenter" : {
            "type" : "string"
        },
        "batterySources" : {
            "type" : "array",
            "items" : {
                "type" : "object",
                "$ref" : "urn:jsonschema:description:power:BatteryDescription"
            }
        },
        "photoVoltaicSources" : {
            "type" : "array",
            "items" : {
                "type" : "object",
                "$ref" : "urn:jsonschema:description:power:PhotoVoltaicDescription"
            }
        },
        "windSources" : {
            "type" : "array",
            "items" : {
                "type" : "object",
                "$ref" : "urn:jsonschema:description:power:WindDescription"
            }
        },
        "gridSources" : {
            "type" : "array",
            "items" : {
                "type" : "object",
                "$ref" : "urn:jsonschema:description:power:GridDescription"
            }
        }
    }
}
```
"dataCenterPowerDescriptionMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:DataCenterPowerDescriptionMessageContents",
    "properties" : {
        "dataCenterPowerDescription" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:description:power:DataCenterPowerDescription"
        }
    }
}

4.3. DataCenterDescription.java and DataCenterDescriptionMessageContents.java

Classes

package description.datacenter;

public class DataCenterDescription extends DataZeroObject {
    String idDataCenter;
    DataCenterITDescription itResources;
    DataCenterPowerDescription powerSources;
}

package message;

public class DataCenterDescriptionMessageContents extends MessageContents {
    DataCenterDescription dataCenterDescription;
}

Json Schema

"dataCenterDescription" : {
    "type" : "object",
    "id" : "urn:jsonschema:description:datacenter:DataCenterDescription",
    "properties" : {
        "idDataCenter" : {
            "type" : "string"
        },
        "itResources" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:description:it:DataCenterITDescription"
        },
        "powerSources" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:description:power:DataCenterPowerDescription"
        }
    }
}

"dataCenterDescriptionMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:DataCenterDescriptionMessageContents",
    "properties" : {
        "dataCenterDescription" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:description:datacenter:DataCenterDescription"
        }
    }
}
5. DataCenter activity

5.1. IT side

5.1.1. IT resources

5.1.1.1. Machine

Classes

```java
package activity.it.resources;
public class MachineActivity extends DataZeroObject {
    MachineDescription managedMachine;
    JobDataZero [] currentJobs;
    int nbCores;
    int indCurrentFrequency;
    MachineActivityMessageContents machineActivityMessageContents;
}
```

```java
package message;
public class MachineActivityMessageContents extends MessageContents {
    private String idMachine;
    private MachineState machineState;
    private float currentFrequency;
    private float powerUsed;
    private int [] jobsPerState = new int[JobState.values().length];
}
```

```java
public class MachineChangeStateMessageContents extends MessageContents {
    private String machineId;
    private MachineState newState;
}
```

```java
public class MachineChangeFrequencyMessageContents extends MessageContents {
    private String machineId;
    private float newFrequency;
}
```

Json Schema

```
"machineActivityMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:MachineActivityMessageContents",
    "properties" : {
        "idMachine" : {
            "type" : "string"
        },
        "machineState" : {
            "type" : "string",
            "enum" : [ "MACHINE_OFF", "MACHINE_ON", "MACHINE_SHUTDOWN",
                       "MACHINE_START", "MACHINE_UNREACHABLE"
                    ]
        },
        "currentFrequency" : {
            "type" : "number"
        },
        "powerUsed" : {
            "type" : "number"
        },
        "jobsPerState" : {
            "type" : "array",
            "items" : {
                "type" : "integer"
            }
        }
    }
}
```

```
"machineChangeStateMessageContents" : {

```
"type": "object",
"id": "urn:jsonschema:message:MachineChangeStateMessageContents",
"properties": {
    "machineId": {
        "type": "string"
    },
    "newState": {
        "type": "string",
        "enum": ["MACHINE_OFF", "MACHINE_ON", "MACHINE_SHUTDOWN", "MACHINE_START", "MACHINE_UNREACHABLE"]
    }
}
}

"machineChangeFrequencyMessageContents" : {
    "type": "object",
    "id": "urn:jsonschema:message:MachineChangeFrequencyMessageContents",
    "properties": {
        "machineId": {
            "type": "string"
        },
        "newFrequency": {
            "type": "number"
        }
    }
}

5.1.1.2. Rack

Classes

```java
package activity.it.resources;

public class RackActivity extends DataZeroObject {
    RackDescription managedRack;
    MachineActivity [] machinesActivity;
    int nbMachines;
    float powerBase;
    RackActivityMessageContents rackActivityMessageContents;
}
```

```java
package message;

public class RackActivityMessageContents extends MessageContents {
    String idRack;
    RackState rackState;
    float powerUsed;
    int [] jobsPerState = new int[JobState.values().length];
    int [] machinesPerState = new int[MachineState.values().length];
}
```

```java
public class RackChangeStateMessageContents extends MessageContents {
    private String rackId;
    private RackState newState;
}
```

Json Schema

```json
"rackActivityMessageContents" : {
    "type": "object",
    "id": "urn:jsonschema:message:RackActivityMessageContents",
    "properties": {
        "idRack": {
            "type": "string"
        }
    }
},
```
"rackState": {  
  "type": "string",  
  "enum": ["RACK_OFF", "RACK_ON", "RACK_SHUTDOWN", "RACK_START", "RACK_UNREACHABLE"]  
},  
"powerUsed": {  
  "type": "number"  
},  
"jobsPerState": {  
  "type": "array",  
  "items": {  
    "type": "integer"  
  }  
},  
"machinesPerState": {  
  "type": "array",  
  "items": {  
    "type": "integer"  
  }  
}  
"rackChangeStateMessageContents": {  
  "type": "object",  
  "id": "urn:jsonschema:message:RackChangeStateMessageContents",  
  "properties": {  
    "rackId": {  
      "type": "string"  
    },  
    "newState": {  
      "type": "string",  
      "enum": ["RACK_OFF", "RACK_ON", "RACK_SHUTDOWN", "RACK_START", "RACK_UNREACHABLE"]  
    }  
  }  
}  

5.1.1.3. Flavor

Classes

package message;  
public class FlavorCreateMessageContents extends MessageContents {  
  private Flavor flavor;  
}  
public class FlavorDeleteMessageContents extends MessageContents {  
  private String idFlavor;  
}  
public class FlavorListMessageContents extends MessageContents {  
  private Flavor [] flavorList;  
}  
public class FlavorListRequestMessageContents extends MessageContents {  
}  

Json Schema

"flavorCreateMessageContents": {  
  "type": "object",  
  "id": "urn:jsonschema:message:FlavorCreateMessageContents",  
  "properties": {  
    "flavor": {  
      "type": "object",  
      "$ref": "urn:jsonschema:description:it:Flavor"  
    }  
  }  
}
"flavorDeleteMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:FlavorDeleteMessageContents",
    "properties" : {
        "idFlavor" : {
            "type" : "string"
        }
    }
}

"flavorListRequestMessageContents" : {
    "type" : "any"
}

"flavorListMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:FlavorListMessageContents",
    "properties" : {
        "flavorList" : {
            "type" : "array",
            "items" : {
                "type" : "object",
                "$ref" : "urn:jsonschema:description:it:Flavor"
            }
        }
    }
}

5.1.1.4. DataCenterIT

Classes

package activity.it.resources;

public class DataCenterITActivity extends DataZeroObject {
    JobDataZeroScheduling jobScheduling;
    DataCenterITDescription managedDataCenter;
    RackActivity [] racksActivity;
    int nbRacks;
    DataCenterITActivityMessageContents dataCenterITActivityMessageContents;
}

package message;

public class DataCenterITActivityMessageContents extends MessageContents {
    String idDataCenter;
    float powerUsed;
    int [] jobsPerState = new int[JobState.values().length];
    int [] racksPerState = new int[RackState.values().length];
    int [] machinesPerState = new int[Machinestate.values().length];
}

Json Schema

"dataCenterITActivityMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:DataCenterITActivityMessageContents",
    "properties" : {
        "idDataCenter" : {
            "type" : "string"
        },
        "powerUsed" : {
            "type" : "number"
        }
    }
}
"jobsPerState" : {
    "type" : "array",
    "items" : {
        "type" : "integer"
    }
},
"racksPerState" : {
    "type" : "array",
    "items" : {
        "type" : "integer"
    }
},
"machinesPerState" : {
    "type" : "array",
    "items" : {
        "type" : "integer"
    }
}
}

5.1.2. Jobs

Classes

package activity.it.jobs;

public enum JobState {
    NEW,
    WAITING,
    RUNNING,
    COMPLETED,
    OUTGOING,
    INCOMING,
    STOPPED,
    UNKOWN
}

public enum JobTypeResource {
    CPU_PERCENTAGE,
    MEMORY_AMOUNT,
    WRITE_VOLUME,
    READ_VOLUME
}

public class JobResources extends DataZeroObject {
    float [] resourcesNeeded;
}

public class JobPhase extends DataZeroObject {
    int duration;
    JobResources resourcesNeeded;
}

public class JobDataZero extends DataZeroObject {
    // creation information
    private String idJob;
    private String pathImage;
    String idFlavor;
    private boolean serviceJob;
    private long arrivalDate;
    private long dueDate; // if batch only
    private boolean isCyclicJob;
    private JobPhase [] listPhase;
    private JobState jobState;
// Evaluated
private long endExecutionDate;
private ArrayList<JobResources> resourcesConsumption;
private float QoS;
private int currentPhase;
}

public class JobDataZeroScheduling extends DataZeroObject {
    private long nextJobId;
    private Hashtable<String, JobDataZero> managedJobs;
}

package message;
public class JobDataZeroArrivalMessageContents extends MessageContents {
    private String idJob;
    private String imageName;
    private String flavorName;
    private boolean serviceJob;
    private long arrivalDate;
    private long dueDate; // if batch only
    private boolean isCyclicJob;
    private JobPhase[] listPhase;
}

public class JobPlacementMessageContents extends MessageContents {
    String idConcernedJob;
    String idMachineFrom;
    String idMachineTo;
}

public class JobChangeStateMessageContents extends MessageContents {
    private String jobId;
    private JobState newState;
}

public class JobConsumptionOrTerminaisonMessageContents extends MessageContents {
    private String idJob;
    private JobResources latestConsumption;
}

public class JobInfosAtTerminaisonMessageContents extends MessageContents {
    private String idJob;
    private long arrivalDate;
    private long dueDate; // if batch only
    private long endExecutionDate;
    private JobResources resourcesObtained;
    private float QoS;
}

Json Schema

"jobState" : {
    "type" : "string",
    "enum" : ["NEW", "WAITING", "RUNNING", "COMPLETED", "OUTGOING",
              "INCOMING", "STOPPED", "UNKNOWN"]
}

"jobTypeResource" : {
    "type" : "string",
    "enum" : [  "CPU_PERCENTAGE",  "MEMORY_AMOUNT",  "WRITE_VOLUME",  "READ_VOLUME"  ]
}

"jobResources" : {
    "type" : "object",
    "id" : "urn:jsonschema:activity:it:jobs:JobResources",
}
"properties": {
  "resourcesNeeded": {
    "type": "array",
    "items": {
      "type": "number"
    }
  }
}

"jobPhase": {
  "type": "object",
  "id": "urn:jsonschema:activity:it:jobs:JobPhase",
  "properties": {
    "duration": {
      "type": "number"
    },
    "resourcesNeeded": {
      "type": "object",
      "$ref": "urn:jsonschema:activity:it:jobs:JobResources"
    }
  }
}

"jobDataZeroArrivalMessageContents": {
  "type": "object",
  "id": "urn:jsonschema:message:JobDataZeroArrivalMessageContents",
  "properties": {
    "idJob": {
      "type": "string"
    },
    "imageName": {
      "type": "string"
    },
    "flavorName": {
      "type": "string"
    },
    "serviceJob": {
      "type": "boolean"
    },
    "arrivalDate": {
      "type": "integer"
    },
    "dueDate": {
      "type": "integer"
    },
    "isCyclicJob": {
      "type": "boolean"
    },
    "listPhase": {
      "type": "array",
      "items": {
        "type": "object",
        "$ref": "urn:jsonschema:activity:it:jobs:JobPhase"
      }
    }
  }
}

"jobPlacementMessageContents": {
  "type": "object",
  "id": "urn:jsonschema:message:JobPlacementMessageContents",
  "properties": {
    "idConcernedJob": {
      "type": "string"
    },
    "idMachineFrom": {
      "type": "string"
    }
  }
}
"idMachineTo" : {
    "type" : "string"
}

"jobChangeStateMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:JobChangeStateMessageContents",
    "properties" : {
        "jobId" : {
            "type" : "string"
        },
        "newState" : {
            "type" : "string",
            "enum" : ["NEW", "WAITING", "RUNNING", "COMPLETED", "OUTGOING",
                      "INCOMING", "STOPPED", "UNKOWN"]
        }
    }
}

"jobConsumptionOrTerminaisonMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:JobConsumptionOrTerminaisonMessageContents",
    "properties" : {
        "idJob" : {
            "type" : "string"
        },
        "latestConsumption" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:activity:it:jobs:JobResources"
        }
    }
}

"jobInfosAtTerminaisonMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:JobInfosAtTerminaisonMessageContents",
    "properties" : {
        "idJob" : {
            "type" : "string"
        },
        "arrivalDate" : {
            "type" : "integer"
        },
        "dueDate" : {
            "type" : "integer"
        },
        "endExecutionDate" : {
            "type" : "integer"
        },
        "resourcesObtained" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:activity:it:jobs:JobResources"
        },
        "qoS" : {
            "type" : "number"
        }
    }
}
5.2. Electric side

5.2.1. The environment of the electric sources

Classes

```java
package activity.power.environment;

public class SourceEnvironment extends DataZeroObject {
    String idEnv;
    Profile expectedEnvironmentValues;
    TimeStampedArray expectedProduction,
        observedEnvironmentValues;
}

package message;

public class SourceEnvironmentMessageContents extends MessageContents {
    SourceEnvironment sourceEnvironment;
}

Json Schema

"sourceEnvironment" : {
    "type" : "object",
    "id" : "urn:jsonschema:activity:power:environment:SourceEnvironment",
    "properties" : {
        "idEnv" : {
            "type" : "string"
        },
        "expectedEnvironmentValues" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:common:Profile"
        },
        "expectedProduction" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:common:TimeStampedArray"
        },
        "observedEnvironmentValues" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:common:TimeStampedArray"
        }
    }
}

"sourceEnvironmentMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:SourceEnvironmentMessageContents",
    "properties" : {
        "sourceEnvironment" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:activity:power:environment:SourceEnvironment"
        }
    }
}
```
5.2.2. Source production

Classes

```java
package activity.power.sources;

public class SourceProduction extends DataZeroObject {
   SourceType sourceType;
    String idSource;
    SourceState currentState;
    TimeStampedArray observedProduction;
    TimeStampedArray lostProduction;
    TimeStampedArray costProduction;
}
```

```java
public class SourceDiagnostic extends DataZeroObject {
    String idSource;
    float [] characteristicValues;
}
```

```java
package message;
public class SourceProductionMessageContents extends MessageContents {
    SourceProduction sourceProduction;
}
```

```java
public class SourceChangeStateMessageContents extends MessageContents {
    private String sourceId;
    private SourceState newState;
}
```

```java
public class SourceChangePowerMessageContents extends MessageContents {
    private String sourceId;
    private float newPower;
}
```

```java
public class SourceGetDiagnosticMessageContents extends MessageContents {
    private String sourceId;
}
```

```java
public class SourceDiagnosticMessageContents extends MessageContents {
    SourceDiagnostic sourceDiagnostic;
}
```

Json Schema

```
"sourceProduction" : {
    "type" : "object",
    "id" : "urn:jsonschema:activity:power:sources:SourceProduction",
    "properties" : {
        "sourceType" : {
            "type" : "string",
            "enum" : [ "BATTERY", "SOLAR", "WIND_TURBINE", "SUPER_CAP",
                        "FUEL_CELL", "GRID", "ALL_RENEWABLE" ]
        },
        "idSource" : {
            "type" : "string"
        },
        "currentState" : {
            "type" : "string",
            "enum" : [ "SOURCE_OFF", "SOURCE_ON", "SOURCE_SHUTDOWN",
                        "SOURCE_START", "SOURCE_UNREACHABLE" ]
        },
        "observedProduction" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:common:TimeStampedArray"
        },
    }
}
```
"lostProduction" : {
  "type" : "object",
  "$ref" : "urn:jsonschema:common:TimeStampedArray"
},
"costProduction" : {
  "type" : "object",
  "$ref" : "urn:jsonschema:common:TimeStampedArray"
}

"sourceProductionMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:SourceProductionMessageContents",
  "properties" : {
    "sourceProduction" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:activity:power:sources:SourceProduction"
    }
  }
}

"sourceChangeStateMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:SourceChangeStateMessageContents",
  "properties" : {
    "sourceId" : {
      "type" : "string"
    },
    "newState" : {
      "type" : "string",
      "enum" : [ "SOURCE_OFF", "SOURCE_ON", "SOURCE_SHUTDOWN",
                 "SOURCE_START", "SOURCE_UNREACHABLE" ]
    }
  }
}

"sourceChangePowerMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:SourceChangePowerMessageContents",
  "properties" : {
    "sourceId" : {
      "type" : "string"
    },
    "newPower" : {
      "type" : "number"
    }
  }
}

"sourceDiagnostic" : {
  "type" : "object",
  "id" : "urn:jsonschema:activity:power:sources:SourceDiagnostic",
  "properties" : {
    "idSource" : {
      "type" : "string"
    },
    "characteristicValues" : {
      "type" : "array",
      "items" : {
        "type" : "number"
      }
    }
  }
}

"sourceGetDiagnosticMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:SourceGetDiagnosticMessageContents",
  "properties" : {
    "sourceId" : {
      "type" : "string"
    },
    "characteristicValues" : {
      "type" : "array",
      "items" : {
        "type" : "number"
      }
    }
  }
}
"properties" : {
  "sourceId" : {
    "type" : "string"
  }
}

"sourceDiagnosticMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:SourceDiagnosticMessageContents",
  "properties" : {
    "sourceDiagnostic" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:activity:power:sources:SourceDiagnostic"
    }
  }
}

5.2.3. Electricity Management in the DataCenter

Classes

package activity.power;
public class DataCenterPowerActivity extends DataZeroObject {
  DataCenterPowerDescription managedDataCenter;
  DataCenterPowerActivityMessageContents dataCenterPowerActivityMessageContents;
}

package message;
public class DataCenterPowerActivityMessageContents extends MessageContents {
  private String idDataCenter;
  private TimeStampedArray renewableObservedProduction;
  private TimeStampedArray renewableLostProduction;
  private TimeStampedArray renewableCostProduction;
  private TimeStampedArray gridConsumption;
  private TimeStampedArray gridCost;
}

Json Schema

"DataCenterPowerActivityMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:DataCenterPowerActivityMessageContents",
  "properties" : {
    "idDataCenter" : {
      "type" : "string"
    },
    "renewableObservedProduction" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:common:TimeStampedArray"
    },
    "renewableLostProduction" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:common:TimeStampedArray"
    },
    "renewableCostProduction" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:common:TimeStampedArray"
    },
    "gridConsumption" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:common:TimeStampedArray"
    },
    "gridCost" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:common:TimeStampedArray"
    }
  }
}
5.3. Datacenter

Classes

```java
package activity.datacenter;

public class DataCenterActivity extends DataZeroObject {
    DataCenterDescription managedDataCenter;
    float powerCooling;
    DataCenterITActivity dataCenterITActivity;
    DataCenterPowerActivity dataCenterPowerActivity;
    DataCenterActivityMessageContents dataCenterActivityMessageContents;
}

package message;

public class DataCenterActivityMessageContents extends MessageContents {
    private String idDataCenter;
    private float powerUsed;
    private DataCenterITActivityMessageContents dataCenterITActivityMessageContents;
    private DataCenterPowerActivityMessageContents dataCenterPowerActivityMessageContents;
}
```

Json Schema

```json
"dataCenterActivityMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:DataCenterActivityMessageContents",
    "properties" : {
        "idDataCenter" : {
            "type" : "string"
        },
        "powerUsed" : {
            "type" : "number"
        },
        "dataCenterITActivityMessageContents" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:message:DataCenterITActivityMessageContents"
        },
        "dataCenterPowerActivityMessageContents" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:message:DataCenterPowerActivityMessageContents"
        }
    }
}
```

5.4. Negotiation

5.4.1. Dedicated profiles

Reminder

```java
class Profile

package common;

public class Profile extends TimeStampedArray {
    protected float quality;
    protected float confidence;
    protected long outDatedAt;
}
```
**Json Schema**

```
"profile" : {
  "type" : "object",
  "id" : "urn:jsonschema:common:Profile",
  "properties" : {
    "startTime" : { 
      "type" : "integer"
    },
    "byStep" : { 
      "type" : "integer"
    },
    "data" : { 
      "type" : "array",
      "items" : { 
        "type" : "number"
      }
    },
    "quality" : { 
      "type" : "number"
    },
    "confidence" : { 
      "type" : "number"
    },
    "outDatedAt" : { 
      "type" : "integer"
    }
  }
}
```

**Classes**

package activity.negotiation;

public class ProfileResponseITDM extends Profile { ... }

public class ProfileResponsePDM extends Profile { ... }

**Json Schema**

```
"profileResponseITDM" : {
  "type" : "object",
  "id" : "urn:jsonschema:activity:negotiation:ProfileResponseITDM",
  "properties" : {
    "startTime" : { 
      "type" : "integer"
    },
    "byStep" : { 
      "type" : "integer"
    },
    "data" : { 
      "type" : "array",
      "items" : { 
        "type" : "number"
      }
    },
    "quality" : { 
      "type" : "number"
    },
    "confidence" : { 
      "type" : "number"
    },
    "outDatedAt" : { 
      "type" : "integer"
    }
  }
}
```
"profileResponsePDM" : {
    "type" : "object",
    "id" : "urn:jsonschema:activity:negotiation:ProfileResponsePDM",
    "properties" : {
        "startTime" : {
            "type" : "integer"
        },
        "byStep" : {
            "type" : "integer"
        },
        "data" : {
            "type" : "array",
            "items" : {
                "type" : "number"
            }
        },
        "quality" : {
            "type" : "number"
        },
        "confidence" : {
            "type" : "number"
        },
        "outDatedAt" : {
            "type" : "integer"
        }
    }
}

5.4.2. A negotiation process

Associated sets of profiles

Classes

```java
package activity.negotiation;

class ProfileSet<ProfileType extends Profile> extends DataZeroObjectSet<ProfileType> {
    // generic instantiation ProfileSet
    ProfileSet<ProfileResponseITDM> responseITDMContents;
    ProfileSet<ProfileResponsePDM> responsePDMContents;

class ProfileRequest extends DataZeroObject {
    protected String idNegotiation;
    protected float relaxationValue;
    protected long startTime;
    protected long byStep;
    protected long endTime;
}
```

Json Schema

```json
"profileSet" : {
    "type" : "object",
    "id" : "urn:jsonschema:activity:negotiation:ProfileSet<common:Profile>",
    "properties" : {
        "contents" : {
            "type" : "array",
            "items" : {
                "type" : "object",
                "$ref" : "urn:jsonschema:common:Profile"
            }
        }
    }
}
```
"responseITDMContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:activity:negotiation:ProfileSet<activity:negotiation:ProfileResponseITDM>",
  "properties" : {
    "contents" : {
      "type" : "array",
      "items" : {
        "type" : "object",
        "$ref" : "urn:jsonschema:activity:negotiation:ProfileResponseITDM"
      }
    }
  }
}

"responsePDMContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:activity:negotiation:ProfileSet<activity:negotiation:ProfileResponsePDM>",
  "properties" : {
    "contents" : {
      "type" : "array",
      "items" : {
        "type" : "object",
        "$ref" : "urn:jsonschema:activity:negotiation:ProfileResponsePDM"
      }
    }
  }
}

"profileRequest" : {
  "type" : "object",
  "id" : "urn:jsonschema:activity:negotiation:ProfileRequest",
  "properties" : {
    "idNegotiation" : {
      "type" : "string"
    },
    "relaxationValue" : {
      "type" : "number"
    },
    "startTime" : {
      "type" : "integer"
    },
    "byStep" : {
      "type" : "integer"
    },
    "endTime" : {
      "type" : "integer"
    }
  }
}

---

**Negotiation and message exchange**

**Classes**

```java
package activity.negotiation;

public class Negotiation extends DataZeroObject {

  String idNegotiation;
  ProfileSet<Profile> ITDMProfileSet;
  ProfileSet<Profile> PDMProfileSet;
}

public enum NegotiationResponseType {
  NEW_PROPOSAL,
  END_NEGO
}
```
public class NegotiationResponse<ProfileResponse extends Profile> extends DataZeroObject {
    String idNegotiation;
    NegotiationResponseType negotiationResponseType;
    float relaxationValue;
    ProfileSet<ProfileResponse> responseContents;
}  
NegotiationResponse<ProfileResponseITDM> negotiationITDMResponse;
NegotiationResponse<ProfileResponsePDM> negotiationPDMResponse;

package message;
public class ProfileSetMessageContents extends MessageContents {
    private String refExchange;
    private ProfileSet<Profile> profileSetValue;
}
public class NegotiationResponseToITDMMessageContents extends MessageContents {
    private NegotiationResponse<ProfileResponseITDM> negotiationResponseITDM;
}
public class NegotiationResponseToPDMMessageContents extends MessageContents {
    private NegotiationResponse<ProfileResponsePDM> negotiationResponsePDM;
}
public class ProfileRequestMessageContents extends MessageContents {
    private ProfileRequest profileRequest;
}

Json Schema

"negotiationResponseType" : {
    "type" : "string",
    "enum" : [ "NEW_PROPOSAL", "END_NEGO" ]
}

"profileSetMessageContents" : {
    "type" : "object",
    "id" : "urn:jsonschema:message:ProfileSetMessageContents",
    "properties" : {
        "refExchange" : {
            "type" : "string"
        },
        "profileSetValue" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:activity:negotiation:ProfileSet<common:Profile>"
        }
    }
}

"negotiationITDMResponse" : {
    "type" : "object",
    "properties" : {
        "idNegotiation" : {
            "type" : "string"
        },
        "negotiationResponseType" : {
            "type" : "string",
            "enum" : [ "NEW_PROPOSAL", "END_NEGO" ]
        },
        "relaxationValue" : {
            "type" : "number"
        },
        "responseContents" : {
            "type" : "object",
            "$ref" : "urn:jsonschema:activity:negotiation:ProfileSet<activity:negotiation:ProfileResponseITDM>"
        }
    }
}

"negotiationResponseToITDMMessageContents" : {

DataZero Messages definition
"type" : "object",
"id" : "urn:jsonschema:message:NegotiationResponseToITDMMessageContents",
"properties" : {
  "negotiationResponseITDM" : {
    "type" : "object",
    "$ref" : "urn:jsonschema:activity:negotiation:NegotiationResponse<activity:negotiation:ProfileResponseITDM>"
  }
}

"negotiationResponsePDM" : {
  "type" : "object",
  "properties" : {
    "idNegotiation" : {
      "type" : "string"
    },
    "negotiationResponseType" : {
      "type" : "string",
      "enum" : [ "NEW_PROPOSAL", "END_NEGO" ]
    },
    "relaxationValue" : {
      "type" : "number"
    },
    "responseContents" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:activity:negotiation:ProfileSet<activity:negotiation:ProfileResponsePDM>"
    }
  }
}

"negotiationResponseToPDMMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:NegotiationResponseToPDMMessageContents",
  "properties" : {
    "negotiationResponsePDM" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:activity:negotiation:NegotiationResponse<activity:negotiation:ProfileResponsePDM>"
    }
  }
}

"profileRequestMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:ProfileRequestMessageContents",
  "properties" : {
    "profileRequest" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:activity:negotiation:ProfileRequest"
    }
  }
}
Negotiation and extractions for the GUI

**Classes**

```java
package database;

public class ResponseGetNegotiationList extends DataZeroObject {
    ArrayList<String> contents;
}

public class ResponseGetNegotiationSelect extends DataZeroObject {
    static public class ItemNegotiationSelect extends DataZeroObject {
        private long timeStamp;
        private String idNego;
        private String topic;
        private String messageValue;
    }
    ArrayList<ItemNegotiationSelect> contents;
}

**Json Schema**

"responseGetNegotiationList" : {
    "type" : "object",
    "id" : "urn:jsonschema:database:ResponseGetNegotiationList",
    "properties" : {
        "contents" : {
            "type" : "array",
            "items" : {
                "type" : "string"
            }
        }
    }
}

"responseGetNegotiationSelect" : {
    "type" : "object",
    "id" : "urn:jsonschema:database:ResponseGetNegotiationSelect",
    "properties" : {
        "contents" : {
            "type" : "array",
            "items" : {
                "type" : "object",
                "id" : "urn:jsonschema:database:ResponseGetNegotiationSelect:ItemNegotiationSelect",
                "properties" : {
                    "timeStamp" : {
                        "type" : "integer"
                    },
                    "idNego" : {
                        "type" : "string"
                    },
                    "topic" : {
                        "type" : "string"
                    },
                    "messageValue" : {
                        "type" : "string"
                    }
                }
            }
        }
    }
}
```

5.5. **The system log**

The GUI needs information about the state of the system, taking into account time intervals or topics, so a log system has been integrated into the platform. Messages designed for this purpose can be used by other DataZero processes.
The log keeps in a database all the messages exchanged (timestamp, topic and message value). Messages are returned on request and the response includes all messages that meet the specified criteria (timestamp, topic et message value).

Classes

```java
package log;

public class LogRequest extends DataZeroObject {
    protected long timeStart;
    protected long timeEnd;
    protected String topic;
    protected boolean onlyLastMessage;
}

public class LogResponse extends DataZeroObject {
    protected long timeStamp;
    protected String topic;
    protected String messageValue;
}

public class LogResponseSet extends DataZeroObjectSet<LogResponse> {
}

package message;

public class LogRequestMessageContents extends MessageContents {
    LogRequest logRequest;
}

public class LogResponseSetMessageContents extends MessageContents {
    LogResponseSet logResponseSet;
}

Json Schema

``logRequest``: {
    "type": "object",
    "id": "urn:jsonschema:log:LogRequest",
    "properties": {
        "timeStart": {
            "type": "integer"
        },
        "timeEnd": {
            "type": "integer"
        },
        "topic": {
            "type": "string"
        },
        "onlyLastMessage": {
            "type": "boolean"
        }
    }
}
```
"logRequestMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:LogRequestMessageContents",
  "properties" : {
    "logRequest" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:log:LogRequest"
    }
  }
}

"logResponse" : {
  "type" : "object",
  "id" : "urn:jsonschema:log:LogResponse",
  "properties" : {
    "timeStamp" : {
      "type" : "integer"
    },
    "topic" : {
      "type" : "string"
    },
    "messageValue" : {
      "type" : "string"
    }
  }
}

"logResponseSet" : {
  "type" : "object",
  "id" : "urn:jsonschema:log:LogResponseSet",
  "properties" : {
    "contents" : {
      "type" : "array",
      "items" : {
        "type" : "object",
        "$ref" : "urn:jsonschema:log:LogResponse"
      }
    }
  }
}

"logResponseSetMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:LogResponseSetMessageContents",
  "properties" : {
    "logResponseSet" : {
      "type" : "object",
      "$ref" : "urn:jsonschema:log:LogResponseSet"
    }
  }
}

5.6. For tests, different topics and one message

To allow tests in message exchange situation, a String message is proposed with three topics. They allow the validation of the exchanges between the parties, without having to actually finalize the information exchanged.

Classes

```java
package message;

public class StringMessageContents extends MessageContents {

    String textMessage;
}
```
### Json Schema

```json
"stringMessageContents" : {
  "type" : "object",
  "id" : "urn:jsonschema:message:StringMessageContents",
  "properties" : {
    "textMessage" : {
      "type" : "string"
    }
  }
}
```

### 5.7. Topics and messages

<table>
<thead>
<tr>
<th>Domain</th>
<th>Message Content</th>
<th>Topic</th>
<th>Object</th>
<th>Producer</th>
</tr>
</thead>
<tbody>
<tr>
<td>description.it</td>
<td>DataCenterITDescriptionMessageContents</td>
<td>IT_DESC_DC</td>
<td>IT description IT of the DC</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td>MachineDescriptionMessageContents</td>
<td>IT_DESC_MACHINE</td>
<td>Description of a machine</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td>RackDescriptionMessageContents</td>
<td>IT_DESC_RACK</td>
<td>Description of a machine rack</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td>description.power</td>
<td>DataCenterPowerDescriptionMessageContents</td>
<td>ELEC_DESC_DC</td>
<td>Electrical description of the DC</td>
<td>PDM</td>
</tr>
<tr>
<td></td>
<td>BatteryDescriptionMessageContents</td>
<td>ELEC_DESC_BATTERY</td>
<td>Description of a battery</td>
<td>PS: battery</td>
</tr>
<tr>
<td></td>
<td>WindDescriptionMessageContents</td>
<td>ELEC_DESC_WIND</td>
<td>Description of a wind turbine</td>
<td>PS: wind turbines</td>
</tr>
<tr>
<td></td>
<td>PhotovoltaicDescriptionMessageContents</td>
<td>ELEC_DESC_VOLTAC</td>
<td>Description photovoltaic generator</td>
<td>PS: solar panels</td>
</tr>
<tr>
<td></td>
<td>GridDescriptionMessageContents</td>
<td>ELEC_DESC_GRID</td>
<td>Description of the grid</td>
<td>FDM</td>
</tr>
<tr>
<td>description.datacenter</td>
<td>DataCenterDescriptionMessageContents</td>
<td>DP_DESC_DC</td>
<td>IT and electrical description of the DC</td>
<td>DZ Supervisor</td>
</tr>
<tr>
<td></td>
<td>MachineActivityMessageContents</td>
<td>IT_MACHIN_STATE</td>
<td>Information on the machine activity</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>RackActivityMessageContents</td>
<td>IT_RACK_STATE</td>
<td>Information on the rack activity</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>DataCenterITActivityMessageContents</td>
<td>IT_DC_STATE</td>
<td>Information on the IT activity of the DC</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>MachineChangeFrequencyMessageContents</td>
<td>IT_MACHINE_CHANGE_FREQUENCY</td>
<td>Machine frequency change order</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>MachineChangeStateMessageContents</td>
<td>IT_MACHINE_CHANGE_STATE</td>
<td>Machine start order</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>MachineChangeStateMessageContents</td>
<td>IT_MACHINE_CHANGE_STATE</td>
<td>Information that the machine is ON</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td>MachineChangeStateMessageContents</td>
<td>IT_MACHINE_CHANGE_STATE</td>
<td>Machine stop order</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>RackChangeStateMessageContents</td>
<td>IT_RACK_CHANGE_STATE</td>
<td>Information that the rack is ON</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td>RackChangeStateMessageContents</td>
<td>IT_RACK_CHANGE_STATE</td>
<td>Rack stop order</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>RackChangeStateMessageContents</td>
<td>IT_RACK_CHANGE_STATE</td>
<td>Information that the rack is off</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td>activity.it.resources</td>
<td>FlavorListMessageContents</td>
<td>IT_FLAV_LIST</td>
<td>Information on the list of existing flavors (compute, memory, and storage capacity of jobs)</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td>FlavorCreateMessageContents</td>
<td>IT_FLAV_CREATE</td>
<td>Flavor creation order</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>FlavorDeleteMessageContents</td>
<td>IT_FLAV_DEL</td>
<td>Flavor deletion order</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>FlavorListRequestMessageContents</td>
<td>IT_FLAV_LIST_REQUEST</td>
<td>Request list of existing flavors</td>
<td>ITDM</td>
</tr>
</tbody>
</table>

A flavor defines the compute, memory, and storage capacity of a virtual machine. In datazero, this concept is also used to represent the needs of a job in terms of CPU, memory and disk.
<table>
<thead>
<tr>
<th>Domain</th>
<th>Message</th>
<th>Topic</th>
<th>Object</th>
<th>Producer</th>
</tr>
</thead>
<tbody>
<tr>
<td>activity.it_jobs</td>
<td>JobDataZeroArrivalMessageContents</td>
<td>IT_JOB_ARRIVAL</td>
<td>Information on taking into account a new job by the DZ system</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>JobPlacementMessageContents</td>
<td>IT_JOB_PLACEMENT</td>
<td>Job placement order (initial or migration)</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>JobChangeStateMessageContents</td>
<td>IT_JOB_CHANGE_STATE</td>
<td>Stopped job activation order: RUNNING</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Information job outgoing: OUTGOING</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Information job incoming: INCOMING</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Job stop order: STOPPED</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>JobConsumptionOrTerminationMessageContents</td>
<td>IT_JOB_TERMINATED</td>
<td>Information on job ending, including the last resource consumption</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td>JobInfoOnTerminationMessageContents</td>
<td>IT_JOB_CONSUMPTION</td>
<td>Information on job resources consumption</td>
<td>OpenStack / Simulator</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Overall job execution information</td>
<td>ITDM</td>
</tr>
<tr>
<td>activity.power.environment</td>
<td>SourceEnvironmentMessageContents</td>
<td>ELEC_ENV_PREVISION</td>
<td>Source environment forecast: (wind, solar radiation, ...)</td>
<td>Retrieved from external sources (Internet) or produced by the PDM</td>
</tr>
<tr>
<td></td>
<td>SourceProductionMessageContents</td>
<td>ELEC_SOURCE_PRODUCTION</td>
<td>Source output</td>
<td>PS: each source</td>
</tr>
<tr>
<td></td>
<td>SourceChangeStateMessageContents</td>
<td>ELEC_SOURCE_CHANGE_STATE</td>
<td>Source start order: SOURCE_START</td>
<td>PDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Information that a source is ON: SOURCE_ON</td>
<td>PS: each source</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Source stop order: SOURCE_SHUTDOWN</td>
<td>PDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Information that a source is OFF: SOURCE_OFF</td>
<td>PS: each source</td>
</tr>
<tr>
<td>activity.power.sources</td>
<td>SourceChangePowerMessageContents</td>
<td>ELEC_SOURCE_CHANGE_POWER</td>
<td>Source power change order</td>
<td>PDM</td>
</tr>
<tr>
<td></td>
<td>SourceGetDiagnosticMessageContents</td>
<td>ELEC_SOURCE_GET_DIAG</td>
<td>Source diagnostic send request</td>
<td>PDM</td>
</tr>
<tr>
<td></td>
<td>SourceDiagnosticMessageContents</td>
<td>ELEC_SOURCE_DIAG</td>
<td>A source sends its diagnostic</td>
<td>PS: each source</td>
</tr>
<tr>
<td>activity.power</td>
<td>DataCenterPowerActivityMessageContents</td>
<td>ELEC_DC_STATE</td>
<td>Information on the electrical activity of the DC</td>
<td>PDM</td>
</tr>
<tr>
<td>activity.datacenter</td>
<td>DataCenterActivityMessageContents</td>
<td>DZ_DC_STATE</td>
<td>Information on the IT and electrical activity of the DC</td>
<td>DZ Supervisor</td>
</tr>
<tr>
<td>activity.negotiation</td>
<td>ProfileSetMessageContents</td>
<td>PDM_TO_NEGO</td>
<td>Sending of power profiles to NEG0</td>
<td>PDM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ITDM_TO_NEGO</td>
<td>Sending of IT profiles to NEG0</td>
<td>ITDM</td>
</tr>
<tr>
<td></td>
<td>NegotiationResponseToTDMMessageContents</td>
<td>NEG0_TO_ITDM</td>
<td>Sending of a response to ITDM</td>
<td>NEG0</td>
</tr>
<tr>
<td></td>
<td>NegotiationResponseToPDMessageContents</td>
<td>NEG0_TO_PDM</td>
<td>Sending of a response to PDM</td>
<td>NEG0</td>
</tr>
<tr>
<td></td>
<td>ProfileRequestMessageContents</td>
<td>NEG0_REQUEST_TO_ITDM</td>
<td>NEG0 requests IT profiles to ITDM</td>
<td>PDM</td>
</tr>
<tr>
<td></td>
<td>ProfileSetMessageContents</td>
<td>NEG0_REQUEST_TO_PDM</td>
<td>NEG0 requests power profiles to PDM</td>
<td>NEG0</td>
</tr>
<tr>
<td>log</td>
<td>LogRequestMessageContents</td>
<td>LOG_REQUEST</td>
<td>Log extraction request</td>
<td>AI</td>
</tr>
<tr>
<td></td>
<td>LogResponseSetMessageContents</td>
<td>LOG_RESPONSE</td>
<td>Messages extracted corresponding to the request</td>
<td>LOG_DB</td>
</tr>
<tr>
<td>debug</td>
<td>stringMessageContents</td>
<td>DZ_Finish</td>
<td>A topic for clean application finish</td>
<td>A1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DZ_STRING1</td>
<td>Topics to exchange messages of type String</td>
<td>A1</td>
</tr>
</tbody>
</table>