FP7 Collaborative Project
no 608806

Project duration:
October 2013 – September 2016

Coodinator:
Svein Hallsteinsen
SINTEF
Norway

Strategic objective:
6.4 Optimizing Energy Systems for Smart Cities

Website:
www.cossmic.eu

D4.3 Report on validation results of the self-optimising multi agent framework

VERSION/STATUS	DUE DATE	DELIVERY DATE
1.0/External Approved	2016-12-31	2017-04-04

DELIVERABLE NATURE	DISSEMINATION LEVEL	NUMBER OF PAGES:
R=Report	PU=Public	77

EDITOR

Salvatore Venticinque/SUN

CONTRIBUTING PARTNERS

SUN, UIO, ISC,NTNU

ABSTRACT

This document describes both theoretical and technical achievements of research activities developed within the WP4 of CoSSMic project. In particular, it focuses on two main features of the CoSSMic multi agent system: learning of energy profiles and techniques for optimal schedule of consuming appliances. Theoretical results include an improved learning model for consuming devices, automatic task detection and management. The report also presents a model to predict solar potential and overshadowing effect of solar systems in urban areas. Moreover results of activities focusing on fulfilment of a distributed policy for the energy optimization are presented.
The practical contribution mainly deals with learning algorithms and their integration in the CoSSMic platform, development of learning tools for off-line data analysis, and the provisioning of a web GUI for monitoring the learning processes. Both the definition and development of the final version of the distributed optimization model are discussed.

INTERNAL REVEWER(S)	APPROVED BY
Shanshan Jiang/SINTEF	Svein Hallsteinsen

CITE AS **CoSSMic deliverable D4.3,** Report on validation results of the self-optimising multi agent framework.

Contents

1 Introduction

D4.3 is a report that describes final development, testing and validation results of the self-optimizing multi-agent system. It includes a description of advances by WP4 about learning and optimization capability over a distributed Neighborhood of consuming and producing devices, according to the research activities planned in Task 4.3 and Task 4.4 of the work-plan. Based on the analysis of results collected by running the first version of the CoSSMic software, the learning capability and the distributed negotiation, integrated into the self-optimizing multi-agent system, have been revised and advanced. This document provides a detailed overview of learning and optimization techniques, a description of developed software and a discussion of results.

1.1 Tasks from the DoW

According to the DoW (Description of Work) two Task 4.3 and Task 4.4 contribute to the development to this deliverable. The following two subsections are a copy of the content the DoW.

1.1.1 Task 4.3 Learning and self-optimisation

The first activity of this task is to identify the active decisions to be made by the players in neighbourhood energy distribution system, i.e. their strategies or actions. An obvious action for all players is to abstain from a play, i.e. withdraw temporarily by not consuming energy from the common pool. A house with solar panels can do this by reducing the energy consumption to match exactly the production, and an electric car can remain unplugged for the duration of the play. However, for players that are pure consumers, switching off all consumption for a period of time will probably not make sense. For these it would probably be more useful if the actions could be defined as "reduce power budget by 10%" or "increase power budget by 10%" or "consume as now". The identification and definition of the actions are the reasons for the task to start early in the project, and the findings will feed into the previous task on the design of the multi-agent framework. The player or agent will obviously chose the action that will maximise the payoff. Since the individual payoff depends on the actions by all players, it is a stochastic value for each player. Thus, the player should also select the action randomly, according to a given probability distribution. If the player modifies the action probability distribution as a result of the payoff, the player or agent learn the combination of actions that maximises the sum of all future payoffs. This is the definition of reinforcement learning, and CoSSMic will therefore model the players as reinforcement learning agents. Central research questions are: How to select the best algorithm for updating the action probability vectors? And more importantly, which algorithms scale well in the number of players and avoid infinite oscillations in the action probabilities? Furthermore, as indicated in Section 1.2 [of the DoW] on peer to peer optimisation, the actions might be dependent on the current state of the system, turning the learning problem into a Markov Decision Problem (MDP). This task will implement the algorithms described in Section 1.2 [of the DoW], and then conduct extensive multi-agent simulations to evaluate both the efficiency of the learning approach, as well as the scalability to systems and

neighbourhoods of a size well beyond what can be demonstrated by the trials in this project. The simulations and the trials of WP5 will be carried out in parallel, and feedback from both activities will be used to further improve the algorithms and the learning based approach. The scalability simulations may proceed in parallel with the trials since we can dimension the trial neighbourhoods such that scalability is not a problem. UIO will be the main partner, in this task supported by SUN. Results will be delivered by D4.2.

1.1.2 Task 4.4 Distributed policy fulfilment

Where the previous task took the view point of a single control agent, this task deals with the more complex situation where many control agents must collaborate to achieve a policy set by the user. One example is when the user first wants to use the power produced by her own solar panels, before buying power from the neighbourhood or the grid. In this case the fixed power budget set by the solar panel control agent must be distributed by the control agents of the consumers. A first approach is to view this as a smaller scale neighbourhood optimisation, and reuse the techniques of the "Power Game" described in Section 1.1 [of the DoW] to manage the situation. However, since all the control agents belong to the same user agent, distributed negotiation models will also be investigated for scheduling the allocation between power sources and energies storages, such as auction and reverse auction models , game theory, iterative bargaining, and many-to-many protocols. The work on this task is deliberately scheduled after the neighbourhood optimisation problem has been solved, since this work is more exploratory, and can be added late to the trials. SUN will contribute to this activity starting from, and advancing the agent based services and algorithms for negotiation and brokering of Cloud resources developed in the FP7 mOSAIC project, and UIO will investigate the suitability of a multi-agent learning based MDP approach. The policies will be designed to be integrated as independent pluggable modules that allows to extend the D4.2 prototype. D4.3 will deliver results of this task.

2 Improved Learning capabilities

Advanced learning capabilities have been designed and implemented in order to achieve the following main objectives:

- Automation of task management to foster the usage of the CoSSMiC scheduler.
- Support of a greater number of heterogeneous devices.
- Improvement of learning results in terms of energy profiles.

In this section we present learning models for different classes of supported devices. We will introduce development achievements and results of validation activities.

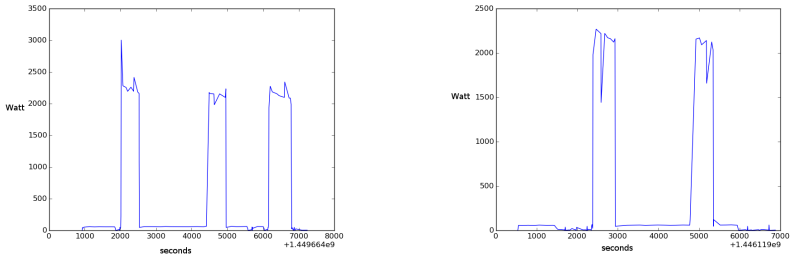
2.1 Learning models

2.1.1 Single run devices

Single run devices are characterized by one task per run, whose energy profile is represented by a single, usually not interruptible load.

The load profile can change because of a different configuration of the device for the specific run or different environmental conditions. For example a washing machine uses a different amount of energy to run a task, according to the programming parameters set by the user, and depending on the weather conditions. In fact on a winter day the energy needed to warm the water could be much more than in summer. Even the amount of clothes to be washed can affect consumption.

For these kind of devices, the energy consumption is continuously measured by CoSSMic, and it is used to learn the average profile and predict the energy requirements for the next run. In Figure 1 there are examples of washing machine profiles for two different programs, measured during the trials at Konstanz in the KN10 site. We show time-series of consumed power, rather than of energy, because the variations can be better understood.



(a) High consuming working mode (b) Low consuming working mode

Figure 1: Example of washing machine power profiles for two different washing programs. Data are collected from KN10 trials.

The learning model for these kind of devices uses a 3-steps approach, (that is an original result of the CoSSMic project) [2, 4]: *start detection*,*stop detection*, average profile reconstruction by a regression model, that uses B-spline

approximation.

The technique for detection of start of appliances has been defined by SUN and improved with the support of ISC. The improvements allows to better filter noise and peaks which was observed in some installations. The final version of the start detection model uses three parameters.

- *noise*. It is a power threshold (Watt). A device consuming less power than this threshold is probably off. On the other hand if it is consuming more than this threshold is probably on.
- *silence*. It is a time interval (seconds). A device that is consuming less than the noise threshold for more than this time interval must be off.
- *silence_start*. It is a time interval (seconds). If its value is 0 and the device is off, then the start of the device is detected at the first power sample greater than noise. If its value is greater than 0, and the device is off, then the start is detected if after the first power sample greater than *noise*, the next power sample is greater than the noise and it is received within *silence_start* seconds.

The improvement of the start detection model that uses the *silence_start* parameter has been suggested by ISC. It observed some spurious power peaks, which produced false detections of switch-on of appliances. The issue was caused by some power peaks over the noise threshold, that were detected for example in KN04, as false switch-on of a dishwasher.

By a deeper analysis about possible causes of this problem, we have concluded that the observed phenomena was associated to a combination of two simultaneous conditions:

a) the usage of some low cost smart-meters (smart-plugs) that communicate only the energy and not the sample time.

b) the high number of devices which communicate their energy at the same time.

Because of the a) condition the platform synchronizes the energy measures at the data collection point. On one side this allows to use the same time reference, that is the CPU clock for all the devices, on the other side the sample time differs from the collection time, because of the delay due to the transmission.

Because of the condition b), the sample E1 communicated by the smart-meter at time t1 can be delayed when the channel is busy. It happens if other meters try to communicate their measures at the same time. The time between the delayed sample and the next sample E2 (not delayed) could be shorter than the actual interval between samples. Thats why the power value computed as (E2-E1)/(t2-t1-delay) could be greater than the true value. In this case a software solution, that is the usage of *silence_start*, allowed us to address the problem without spending more effort. We did not need to change the installed equipments with more expensive ones, and without reducing the number of monitored devices. The solution worked because the installed smart plugs increase the communication frequency, when consumed power changes faster. *The lesson learned tells us that to avoid this kind of problems one needs to use*

smarter smart-meters, which allow for reading both energy and sample-time, or at least both energy and power.

On clicking a single run device such as the washing machine in the CoSSMic user interface, the form shown in Figure 2 appears. It includes static parameters that should be set once, at device installation time.

These parameters must be tuned for the specific devices and installations. During the project it was the responsible of installation and configuration, who collected and analysed the measures and the consuming profiles. In production these parameters will be part of the device templates, which will be pre-loaded into the software, and eventually updated from remote, selected by the user herself, who will apply the one corresponding to manufacturer and model of her own device.

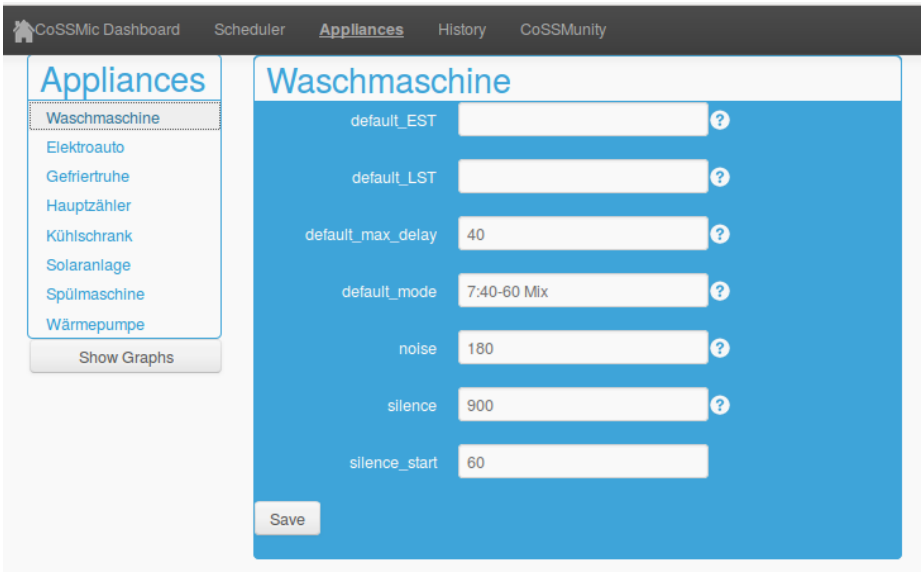


Figure 2: User interface to configure Single run devices

Stop detection

Suppose we are measuring the cumulative energy consumed by device at any sampling rate, that is not necessary uniform. We compute the power consumption from these values. The stop of energy consumption by a single run device is detected using two parameters:

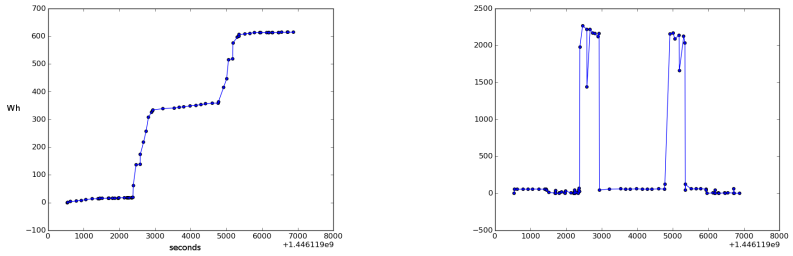
- the average duration of the load profile
- the *silence* parameter defined at the previous paragraph.

The stop of a device is detected when the power samples, whose values are less than the noise threshold, are received for a time interval that is longer than the number of seconds specified by silence. In addition, when the average duration, which has been learned from previous runs, is greater than 0, the system asynchronously checks the status of the device to speed up the detection. The stop time is the time value of the first sample below noise received during the silence interval.

Average profile learning

The last n executions detected by the systems, for the same working program, are used to learn an average profile and to predict the energy requirement for the next run. At the state of the art CoSSMic is not able to dynamically recognize the program set by the user. We suppose that the user sets a default working program and changes it, if it is necessary, before switching on the device. An off-line method, for supervised clustering of exection according to the related working program has been defined and developed by SUN. It will be described in Section 2.3. According to the methodology described in [4], we merged all the samples for the different runs to reconstruct a fine grained time series of cumulative energy samples.

In Figure 3 we have the power profile of a single load and its original representation as cumulative energy.



(a) Original cumulative time-series for measured samples

(b) Computed power profile

Figure 3: Comparison of original energy samples with computed power time-series

Every new run with a particular working mode, of a single-run device, adds new samples to the *observation* set of the ensemble of runs. The *load profile* for a given working mode of an appliance should represent the mean value of the cumulative energy consumption at a given time, based on the scattered observation set. It is therefore natural to think of the load profile as a regression function, *i.e.* the conditional expectation of the energy consumption given the time samples. In general, a *regression function* is a *model* of the relation between the independent time variable and the dependent energy consumption, $L(t|\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a vector of model parameters.

One model free solution would be to use *smoothing splines* that are polynomials of degree d between any pair of samples $\langle t_k, t_{k+1}\rangle$ and the resulting regression function is continuous in the first $(d+1)/2$ derivatives The main issue with this approach is that the resulting function has as many "pieces" as there are intervals between samples. The available measurements from various devices seem to indicate that a device will typically have periods of little consumption, or "off" periods intermixed with periods of continuous consumption or "on" periods. This indicates that it should be possible to make the interpolating curves span larger sections of the sample interval.

Our suggestion is to use *basis splines* (B-splines) for the regression [8]. Like smoothing splines the B-spline will consist of a set of continuous polynomials that are joined to a continuous regression function at a set of *knots* whose

cardinality may be much less than, and independent of, the cardinality of the observation set. Furthermore, the geometric properties of the function is determined by a set of *control points*. B-splines are generalisations of Bézier curves as the latter is a B-spline with no internal knots. The design parameters to be chosen are the number of control points, C, and the degree of the partial spline polynomials, d.

The parameter vector $\boldsymbol{\theta}$ will consist of the knot positions and the control points of the B-spline. The number of parameters will in general be less than the number of observation points, and the parameter vector will be found by solving the non-linear programming problem for n equal to the number of observations

$$\min_{\boldsymbol{\theta}} \frac{1}{n-1} \sum_{k=1}^{n} \left[L\left(t_k\right) - L\left(t_k|\boldsymbol{\theta}\right) \right]^2 \tag{1}$$

As $L(t|\boldsymbol{\theta})$ is the conditional expectation, the objective function in (1) is easily recognised as the *unbiased sample variance* of the observations. The resulting load profile will consequently be a minimum variance regression to the available observations.

The vector of regression errors follows directly from the minimisation problem as $\boldsymbol{e} - \boldsymbol{B}\left(\boldsymbol{t}|\boldsymbol{k}\right)\boldsymbol{c}_E$. Its sample average is $\left[\boldsymbol{e} - \boldsymbol{B}\left(\boldsymbol{t}|\boldsymbol{k}\right)\boldsymbol{c}_E\right]^T \mathbf{1}/n$ where $\mathbf{1}$ is a vector with all elements equal to unity. Its unbiased variance estimate is $\left\|\boldsymbol{e} - \boldsymbol{B}\left(\boldsymbol{t}|\boldsymbol{k}\right)\boldsymbol{c}_E\right\|^2/(n-1)$. Both quantities are readily available after solving the minimisation problems, and one can then use sample Chebyshev bounds [7] to establish a confidence interval around the load profile.

This process is illustrated in Figure 4, which shows the 62 observations of the same *mode* sampled twice from consecutive runs of a washing machine. The 95% Chebychev confidence interval using sample mean and variance is about ± 0.092 for these time series. It evident that scheduling the load based on the upper bound seems a safe choice.

Additional information has been published in [4].

The B-spline approximation provides an analytical representation of the profile that:

- minimize the error square root;
- allows to reduce the communication overhead during the negotiation between producers and consumers of different micro-grids;
- allows to choose a compromise between precision of the analytical representation and communication overhead.

The B-spline formula for representing such a profile is:
$f\left(t\right) = \sum p_i\left(t\right) B_{i,0}\left(t\right)$
An example of profile representation is expressed by the following list of parameters-values.

"k": 3, "y": [171.20715823281222, -3.7774669541951154, 429.3349908308822, 486.90619333257257, 993.0089657794821, 929.9440925276704], "t": [0.0, 2348.0, 3924.0, 4981.0]

(2)

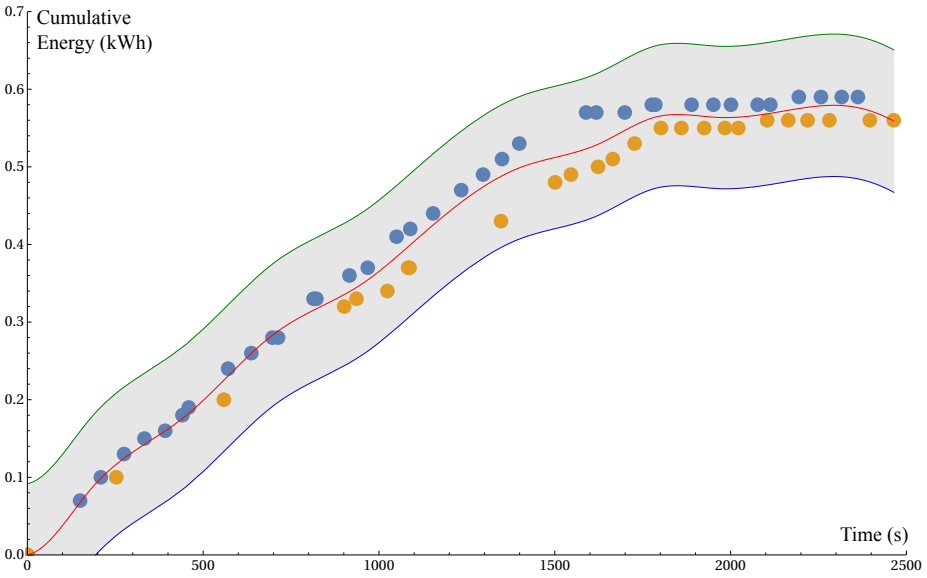


Figure 4: The two runs 9 and 13 of the same mode with the conditional mean load as a B-spline curve of degree $d = 3$ with 15 control points and its 95% Chebychev bound based on sample mean and variance.

The size of representation does not depend on the number of samples, which compose the time-series, but only on the polynomial degree and on the numbers of knots.

2.1.2 Continuously run devices

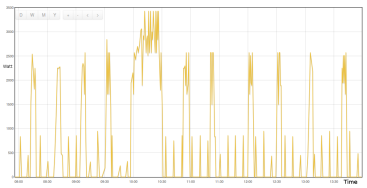
Continuously run devices are characterized by long executions. Examples of appliances are fridges, freezers, heat pumps, air conditioners. During their execution, they periodically generate new tasks, that is defined as an consuming-time (with a specific energy load) and a off-time. Start, stop, duration and energy consumption of tasks are managed by an internal controller and usually depend on environmental conditions.

The frequency of task generation and their energy profiles can change from season to season, from day to day and during the day. Moreover some random conditions can also occur. For example when the user frequently opens the door of the fridge or the window of a room the internal controller of devices can switch on the cooler and starts to consume.

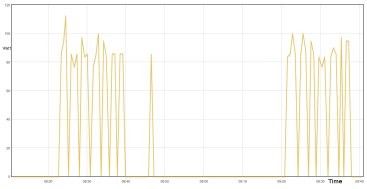
In Figure 5 there is an example of profile of an heat-pump and of a fridge installed in KN10.

Because of the heterogeneity of devices, and because of the lack of an open interface for monitoring and controlling this kind of devices currently installed in users households, we used smart-plugs both to monitor the energy consumption and to unplug the device. The smart-plug allows to shift the consumption only forward. In fact we switch-off a device if the scheduler suggests to delay the consumption.

The learning model designed and developed in CoSSMic represents each long



(a) Heat pump power profile from trial KN10



(b) Fridge power profile from trial KN10

Figure 5: Power profile of two different continuously running device

run of this kind of devices as a sequence of tasks. Each task is modeled by three parameters:

- *off-duration*. This is the duration (secs) of the last time interval from when the device has been switched off by the internal controller and until it has been switched on again.
- *on-duration*. This is the duration (secs) of the last time interval from when the device has been switched on by the internal controller and until it has been switched off.
- *consumed-energy*. This is the amount of energy used during the last run.

These parameters are used to predict the consumption at the next switch-on and the time when the next switch-on is expected. In particular when a switch-on is detected the off-duration is learned. When a switch-off has been detected, the consumed-energy and the on-duration are recorded. Moreover, when the switch-off has been detected, the next switch-on is estimated at the switch-off time plus the off-duration. Start-time and stop-time detections work according to the same approach defined for Single run devices. A continuously run profile will be represented as a simple time-series of cumulative energy:

```
[[0,0]
[on-duration,consumed-energy]]
```

A click on a continuously running device, like a heat-pump, in the CoSSMic GUI, opens the form shown in Figure 33. The form allows for setting static parameters, which must be initialized at installation time to configure the learning process.

2.1.3 E-cars

The *e-cars* are modeled as consuming devices, which can generate multiple and different loads, which can be separated or can overlap. Each e-car is modeled as a storage with static and dynamic parameters:

- *capacity*. It is the amount of energy that the battery can store, as declared by the manufacturer.
- *ChargingEfficiency*. It is the percentage of capacity the battery can effectively use at the current time because of its aging.

- *MinimumChargingLevel*. It is the minimum charging level that is needed to guarantee not to damage the battery.
- *MaxChargingPower*. This is the maximum power we can use to charge the battery.
- *MinChargingPower*. If power is below this value the battery cannot charge.
- *minimum_energy_target*. This is the minimum charging level the user wants to charge as soon as possible.
- *energy_target*. This is the charging should be charged within the target deadline.
- *target_deadline*. This is the time of the day before which the battery of the e-car must be charged at the energy target value.

In Figure 6 a common charging profile for e-car of KN10 trial is shown. It charges the e-car at the maximum charging power.

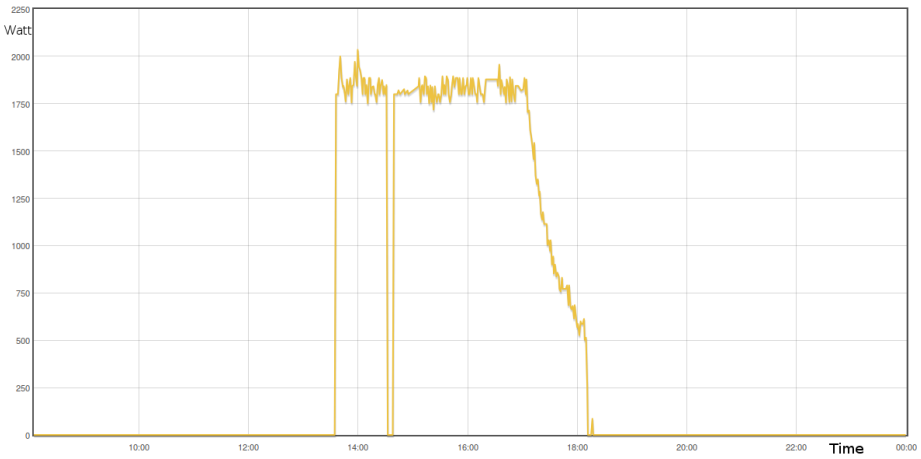


Figure 6: Example of power profile of e-car from trial KN10

Using a specific API, it is necessary that the driver alerts the CoSSMic learning process when the car is plugged. In fact the start detection of charging is not enough to plan the next loads. It needs to get the State of Charge (SoC) of the battery.

In the case of the e-car model of CoSSMic trials, the manufacturer provided a web service by which it was possible to get the SoC of the e-car. The service was invoked by the driver. However only when the e-car charger is plugged, the SoC value is used to plan the charging profile. The effort needed to develop the driver, and the software updates of the service interface by the manufacturer, limited the exploitation of the e-car model in the trials. This particular experience demonstrates once more the need of open and hopefully standard interface of smart devices, and above all of battery management systems. Moreover, above all in the case of e-cars, which are unplugged and move from a location to a different one, it is necessary to make this information available by an ubiquitous service.

When the driver notifies the SoC a first load is planned. It will charge the e-car at the maximum charging power until the minimum_energy_target is reached.

For this reason a first load will have:

$$on\_duration = \frac{minimum\_energy\_target - current\_energy\_target)}{MaxChargingPower}$$

$$consumed\_energy = minimum\_energy\_target - current\_energy\_level$$

It will be submitted with EST=LST=now only if current_energy_level is less than minimum_energy_target.

A second load is planned with an earliest start time right after the end of the first load. The learning process will first compute the average power needed to charge the battery from the end of the first load to the target_deadline. If such value is less than the MinimumChargingPower than the MinimumChargingPower is used.

Its profile will be:

$$on\_duration = \frac{energy\_target - max(current\_energy\_level, minimum\_energy\_target)}{average\_power}$$

$$consumed\_energy = energy\_target - max(current\_energy\_level, minimum\_energy\_target)$$

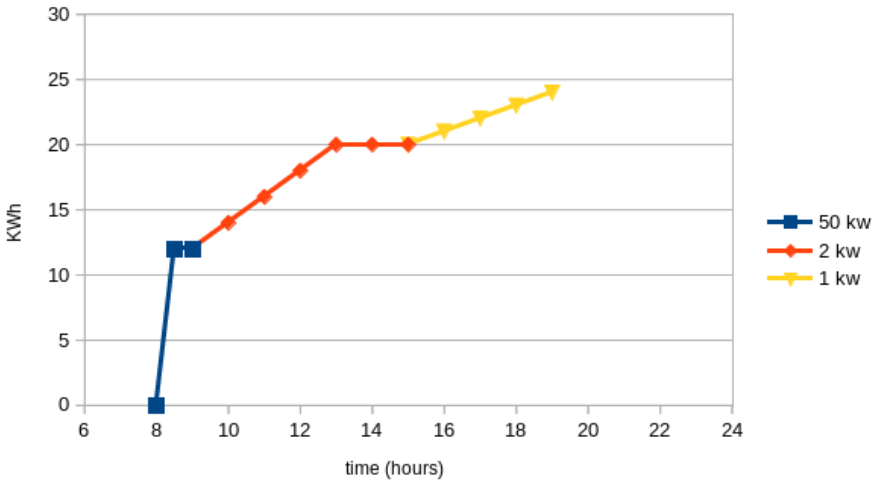


Figure 7: Planning example for e-car as a sequence of two loads.

After the fast charge to the minimum_energy_level the residual energy to achieve the energy_target can be also split into more loads. For example, using the parameters values in the following list, the charging profile can be split as it is shown in Figure 7.

- Capacity: 24 kWh
- ChargingEfficiency: 1
- MaxChargingPower: 50kW

- minimum_energy_target: 50%
- energy_targetl: 100%
- target_deadline: 20:00

A click on an e-car device, in the CoSSMic GUI, opens the form shown in Figure 8. It shows static parameters that must be set at device installation time.

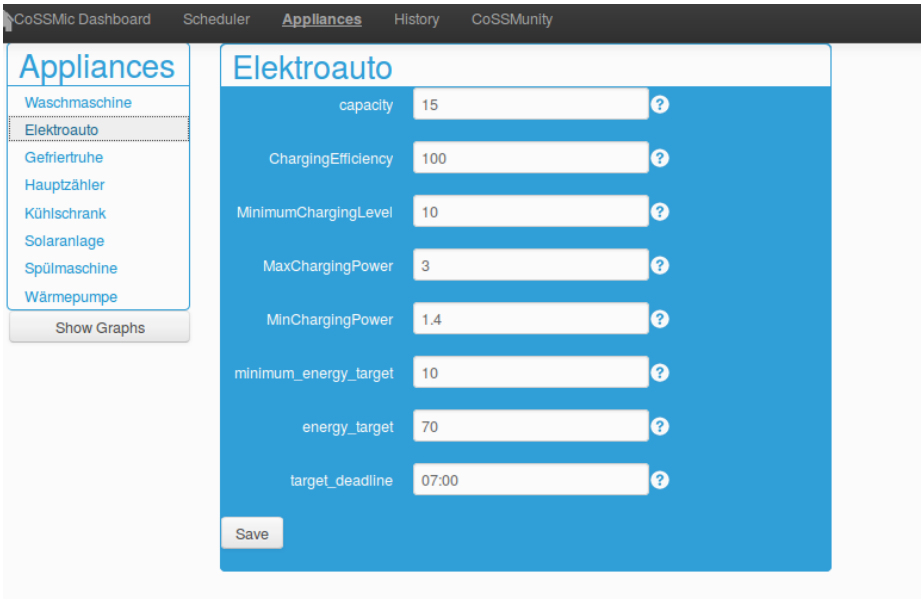


Figure 8: E-car static configuration.

2.1.4 Reactive management of background consumption

These kinds of loads belong to the background consumption of the micro-grid, which that is monitored using a global smart-meter. It is computed by subtracting the consumptions of devices which are individually controlled by CoSSMic from the total consumption of the household. This load cannot be shifted, however the awareness of this unmanaged consumption can be used by CoSSMic to better estimate the renewable energy actually available. Two learning approaches are available in CoSSMic to optimize the schedule:

- **Predictive.** The background load for the current day is predicted equal to the background consumption of the same day of the previous week.
- **Reactive.** The background load for the next time slot is predicted equal to the load of the last slot.

In the first case the daily background prediction is split in different chunks of a fixed duration and average power. Duration is set statically, average power is computed analytically. They are submitted at the beginning of the day without flexibility and are allocated by the producer to improve the estimation of the available renewable energy. In the second case, chunks are estimated and submitted one after the other. If the background load was greater than 0 during the last time slot, a load equals to the last observed one is submitted for the next time slot.

How background loads are estimated will be explained in the Section2.3.3.

2.1.5 Zero delay single-run devices

If the user does not want to delay the start of the device it can set a 0 delay and the energy profile will be submitted only to improve the estimation of available energy as it was explained for for background loads in the previous section.

2.1.6 Photovoltaic Panels

In order to improve the prediction of production, NTNU research unit investigated the overshadowing effect on solar systems, which have been installed in urban areas, exploiting the data collected by the Konstanz trials. The focus was to develop a PV-shadowing prediction model based on the combination between solar analyses conducted by using dynamic simulation software (i.e. DIVA for Rhino) and a python-based code developed in order to handle the outputs coming from the solar analyses on the surface.

The approach used for the case study in Konstanz faces the challenge to bridge the gap between the solar analyses which calculate the hourly values of solar radiation arriving on the surfaces and the estimation of the energy production evaluated using the yield-calculation tool. It is a tool able to calculate energy gains from PV installations on buildings. Annual values of solar radiation components comprising global, beam and diffused radiations are used as inputs for this tool. Moreover, technical specifications related to PV panels such as the orientation, the installation parameters etc. are taken into consideration by the tool as well. The annual energy gained from PV is provided by the tool. It along with the solar analysis provides an integrated algorithm used to obtain a more precise prediction of solar electricity for buildings. The prediction takes

in consideration both the solar radiation under the real weather conditions as well as the overshadowing effect created by the urban surrounding and by the design of the solar system itself. The entire process is composed of the following steps:

1. Modeling from the vector file (i.e. dwg), the surfaces of the urban surrounding and the profile of the terrain of the urban area where the case study is located;
2. Converting the weather data climate from a meteorological station into an .epw file format usable for running the solar simulations for the real cases.
3. Conducting solar dynamic simulation analyses for calculating the solar radiation on the building envelopes and/or PV surfaces such as louvres using DIVA for Rhino. The outputs of this stage are the hourly values of the direct, diffuse and global components of the solar irradiation for the entire year.
4. The outputs of the step 3 will be used as inputs for the energy production calculation conducted by the yield-calculation tool. At this stage the final outputs are the energy gains from PV installations.

After that, we started an analysis of the potential solar radiation obtained from an hypothetical weather data file. The methodology consisted of the following phases:

1. Use of the same case study model of the previous analysis.
2. Creation of a potential weather data climate in .epw file format from statistical available data to be used for running the solar simulations in the hypothetical scenario of maximum solar irradiation. It was developed individuating the maximum hourly value for each week and these results were applied to 7 days of a week. The process was carried on for the entire year.
3. Conducting solar dynamic simulation analyses for calculating the solar radiation on PV surfaces using DIVA for Rhino and the previously developed potential weather data file.

Simulations were run both for the isolated and obstructed contexts. The outputs of this stage are the hourly values of the direct, diffuse and global components of the solar irradiation for the entire year, obtained setting materials reflection and the parameter ab (maximum number of diffuse bounces computed by the indirect calculation) with values 0-1.

In Table 10 there is a description of the parameters used for the different analysis.

The process has been tested on the solar systems installed on the facade (PV/1) and on the roof (PV/2) of a commercial building in Konstanz (Germany). The two systems have the following features:

- The facade system PV/1 is constituted by five rows of 14 PV panels (dimensions: 64.5cm x 128cm) each. The distance between the rows is 1.5m. The panels are 30 tilted and 170 South-East oriented (Figure 11 (a).)

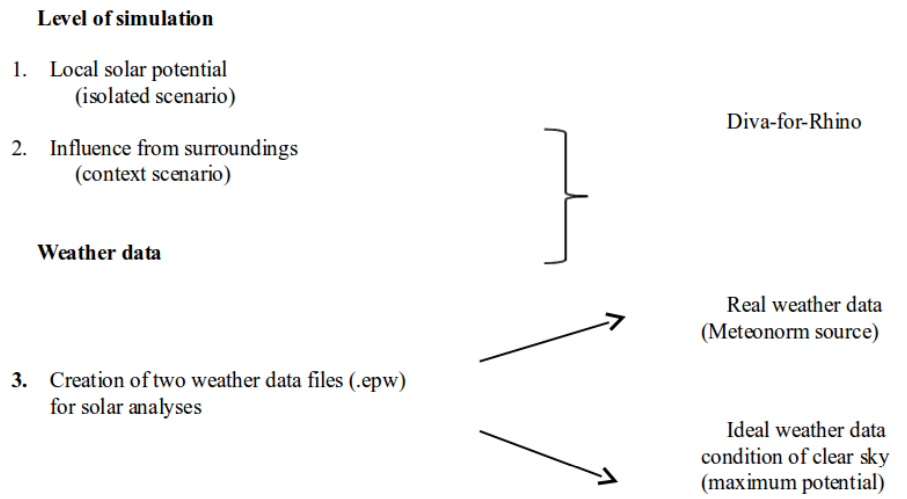


Figure 9: Scheme of the methodology

Type of context	*ab*	*Materials*	*Output*
Isolated scenario	0	Ground 20; Building 35	Direct Irradiation
	1	Ground 00; Building 00	Global Irradiation
Context scenario	0	Ground 20; Building 35	Direct Irradiation
	1	Ground 00; Building 00	Global Irradiation without reflections
	1	Ground 20; Building 35	Global irradiation including reflected irradiation

Figure 10: Input data for simulations: materials and parameters.

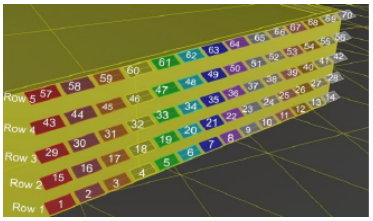
- The roof system PV/2 is composed by nine rows of six PV panels (dimensions: 99cm x 168cm) each. The distance between the rows is 34cm. The panels are 14 tilted and 170 orientation South-East oriented (Figure 11 (c)).

From the process, examples of different profiles have been extracted are:

1. *Profile 1* describes the solar potential radiation (ideal weather conditions with clear sky for an entire year). This profile will calculate the solar radiation on the analyzed surfaces without any obstruction and under ideal weather condition of clear sky. The analyses will be conducted in isolated scenario with the profile of the terrain that will give a contribution in terms of solar reflection from the ground.

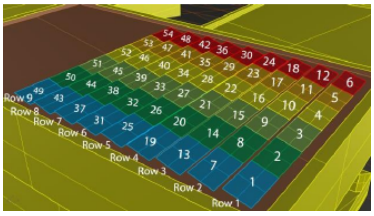
2. *Profile 2* describes typical solar radiation during an entire year (real weather conditions). This profile allows calculating the solar radiation on surfaces considering the real weather data climate provided every 4 hours from the meteorological data station. The profile 2 will be calculated in isolated scenario without any obstruction given by the urban surrounding. The simulations will be run considering the profile of the terrain that will

(a) The photo shows how the PV/1 system looks like.



(b) The composition of the PV/1 system on the facade.

(c) The photo shows how the PV/2 system looks like



(d) The composition of the PV/2 system on the facade.

Figure 11: Example of real systems and composition of facades.

give a contribution in terms of solar reflection. In this way is possible to deduct the differences between profile 1 and profile 2 without any affection given by the urban surrounding.

3. *Profile 3* describes solar radiation affected by mutual urban solar reflection. The analyses calculate the solar radiation in context scenario in order to take into account the solar mutual contributions (reflections among the building and the ground) and the overshadowing effect by urban surrounding.

4. *Profile 4* describes typical solar radiation during an entire year as done in Profile 2, but using maximum potential weather conditions. It was calculated in isolated scenario without any obstruction given by the urban surrounding. The simulations were run considering direct and global irradiations as output. Secondly, it was possible to deduct the diffuse irradiation as differences between the previous ones.

5. *Profile 5* describes solar radiation affected by mutual urban solar reflection. The analyses were used to evaluate the solar radiation in context scenario in order to take into account the solar mutual contributions (reflections among the building and the ground) and the overshadowing effect by urban surrounding. Simulations were run using the maximum potential weather data file, different ab parameters and material settings to obtain direct irradiation, global irradiation without reflections and reflected irradiation. Secondly, the diffuse irradiation was calculated starting from the previous achieved data.

For the case study in Konstanz the .epw weather data climate extracted from

Meteonorm [1] has been used for Profiles from 1 to 3, while for Profile 4 and 5 it was used the potential weather data file, created as described above. The most relevant results for the system PV/1 and PV/2 analyzed in Konstanz are shown below for both weather data files used.

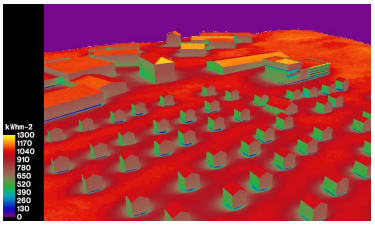
Results for the real weather data from Meteonorm are:

Isolated scenario:

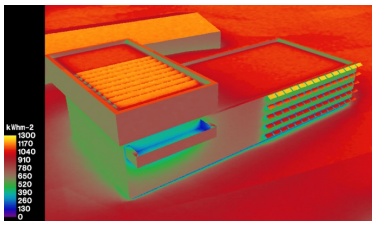
- The PV1 system is affected by overshadowing effect due to its reported shadowing. This aspect is responsible by 6% of reduction of direct radiation, and until 10% for the global radiation between the profile 1 and the profile 2;
- The PV2 system has a reduction caused by its design around 2% for direct radiation, and until 2.5% for the global radiation between the profile 1 and the profile 2;

Context scenario

- The PV1 system is affected by overshadowing effect due to the urban surrounding. This caused until 10% of reduction of direct radiation, and until 20% for the global radiation between the profile 2 and the profile 3;
- As expected the results have confirmed that for PV/2 is practically not affected by the overshadowing created by the surrounding.



(a) The solar mapping of the entire district in which the ISC building is located



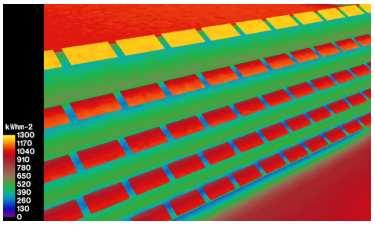
(b) The solar mapping analysis of the ISC building with the system PV/1 installed on the facade and PV/2 installed on the roof

Figure 12: Example of solar mapping

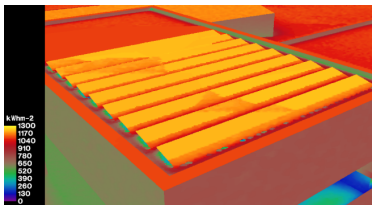
For the system PV/1 installed on the facade, the highest contribution in terms of solar reduction comes from the direct radiation: from 7.7% for the 2nd row and 10.5% for the 5th row on the bottom in the isolated scenario. While in the context scenario, the presence of the urban surrounding and in particular of the intervention of renovation constituted by a new horizontal block added on the top of the existing building and the balcony give a reduction in terms global of solar radiation (from 12.6% to 21.1%) (Figure 14). The difference is made by comparing profile 2 and profile 3 (Figure 15).

For the system PV/2 installed on the roof, the reduction in terms of solar radiation is around 2% in both scenarios, isolated and obstructed Figure 14.

[1] Meteonorm Software - Irradiation data for every place on Earth. (http://meteonorm.com/)



(a) The solar mapping analysis of the PV/1 system on the facade. The image shows the overshadowing effect from the top to the bottom of the system.



(b) The solar mapping analysis of the PV/2 system on the roof. The image shows the homogenous radiation on the system

Figure 13: Example of.

This means that the overshadowing effect is created only by the design of the system, while there is not any influence from the urban surrounding. Regarding the solar reflected component (from the ground reflection), it is resulted in a very low influence (<1%) for the PV/1in the context scenario, while is completely absent for PV/2.

PV/1 Row	Beam Irr. Isol. scenario [kWh/m²yr]	Δ [%]	Global Irr. Obstr. scenario [kWh/m²yr]	Δ [%]	Beam Irr. Obstr. Scenario [kWh/m²yr]	Δ [%]	Global Irr. Obstr. scenario [kWh/m²yr]	Δ [%]
5	653.4	/	1238	/	651.8	/	1229.6	/
4	614.4	-6.4	1140.3	-8.6	605.2	-7.7	1091.7	-12.6
3	613	-6.6	1134	-9.2	599.7	-8.7	1045.5	-17.6
2	613	-6.6	1134	-9.2	591.7	-10.2	1023.2	-20.2
1	613	-6.6	1118.3	-10.7	589.6	-10.5	1015.5	-21.1

Figure 14: Direct and global radiation on the system PV/1 in the isolated and context scenario.

The results of PV/2 were compared with the measured data on site conducted by Abdelraheem, Ahmed Khaled Farghaly during his master thesis conducted at ISC in Konstanz. The results are similar for direct radiation (Figure 22) and global (Figure 21), while there are some differences for the diffuse radiation (Figure 23). However, all the profiles of the components are similar: in both conducted analyses the periods of the year in which the system PV/2 has the highest reductions in terms of solar radiation are always the same. The profiles of the components are quite similar and the calculations demonstrated that the methodology used for achieving the goal of prediction solar potential and overshadowing effect caused by the design of the system and the urban surrounding gives reliable results and trends.

Results arising from simulations with the developed potential weather data are shown below. Specifically, they resulted from the isolated and context scenario for PV/1 and PV/2 systems.

For the system PV/1 installed on the facade, the highest gap between the row 1 and row 5 resulted in the periods of May-June and of July-August when the sun is higher and consequently the radiation is greater. This loss is due to an atypical cloudy condition during summer in the weather data file from Meteonorm. In addition, the loss between the first and the last row of the

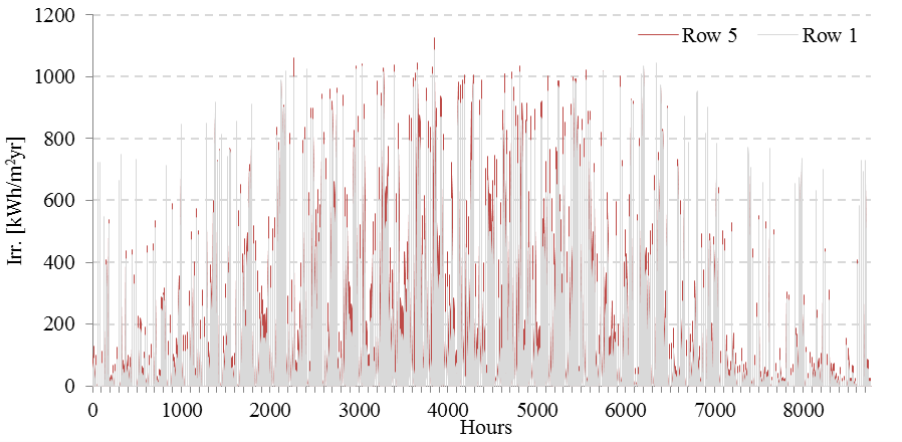


Figure 15: The profile of the global radiation on the row 1 and 5 of the PV/1 in the context scenario.

system in terms of radiation is due to shadowing effect caused by the geometry and the position of the entire panel facade. In order to avoid these consequences, the system should be improved taking into account the inclination of panels and the distance between them.

For the system PV/2 installed on the roof, the main difference between the first and the last row is identified during winter season, contrarily to the facade system. This is due to the inclination of the sun during that period of the year. Moreover, the reduction in terms of solar radiation is caused by panels design, including their geometry and the presence of a border in the roof. In fact, the overshadowing effect is created more by the design of the system, than the urban surrounding.

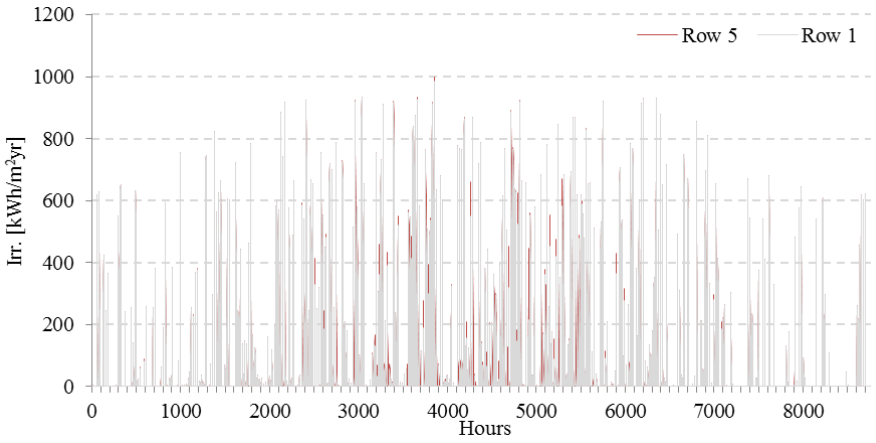


Figure 16: The profile of the direct radiation on the row 1 and 5 of the PV/1 in the context scenario

PV/1 Row	Beam Irr. Isol. scenario [kWh/m²yr]	Δ [%]	Global Irr. Obstr. scenario [kWh/m²yr]	Δ [%]	Beam Irr. Obstr. Scenario [kWh/m²yr]	Δ [%]	Global Irr. Obstr. scenario [kWh/m²yr]	Δ [%]
9	598.2	-1.9	1173.2	-1.7	598	-1.7	1167.7	-1.8
8	598.3	-1.9	1172.5	-1.8	598	-1.7	1171.8	-1.5
7	598	-2.0	1167.3	-2.2	597.8	-1.8	1167.8	-1.8
6	598	-2.0	1166	-2.3	597.8	-1.8	1167.5	-1.8
5	598.2	-1.9	1168	-2.2	598	-1.7	1165	-2.1
4	598.5	-1.9	1168.6	-2.1	598.3	-1.7	1166	-2.0
3	598.6	-1.9	1168.6	-2.1	598.3	-1.7	1166.8	-1.9
2	600.5	-1.5	1173	-1.7	598.3	-1.7	1164.7	-2.1
1	609.8	/	1193	/	608.3	/	1189	/

Figure 17: Direct and global radiation on the system PV2 in the isolated and context scenario.

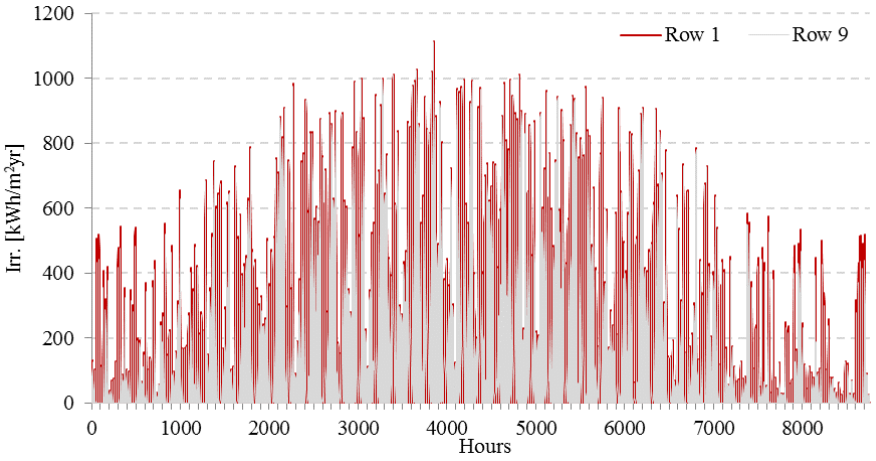


Figure 18: The profile of the global radiation on the row 1 and 9 of the PV/2 in the context scenario.

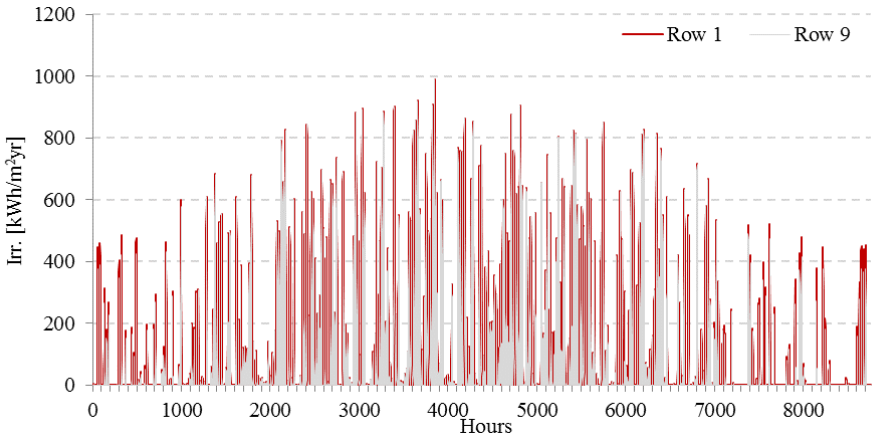


Figure 19: The profile of the direct radiation on the row 1 and 9 of the PV/2 in the context scenario

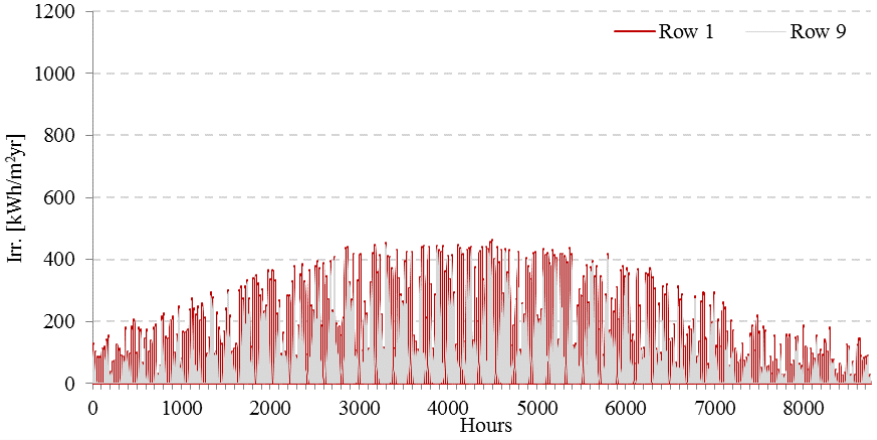


Figure 20: The profile of the diffuse radiation on the row 1 and 9 of the PV/2 in the context scenario

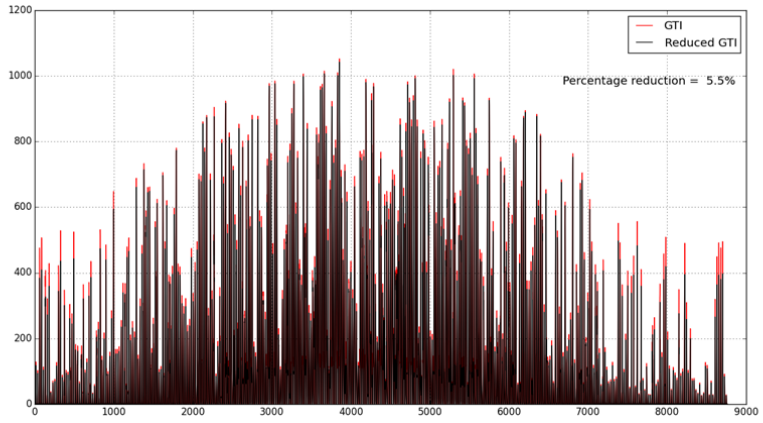


Figure 21: The profile global radiation of the system PV/2 carried out from the measured data on site. (Author: Abdelraheem, Ahmed Khaled Farghaly)

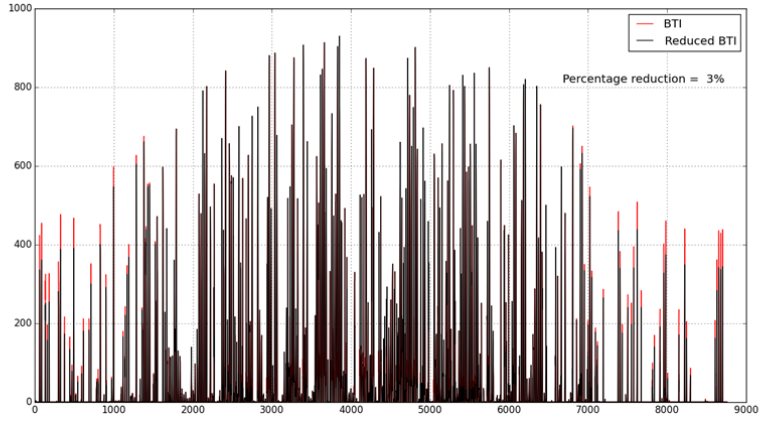


Figure 22: The profile direct radiation of the system PV/2 carried out from the measured data on site. (Author: Abdelraheem, Ahmed Khaled Farghaly)

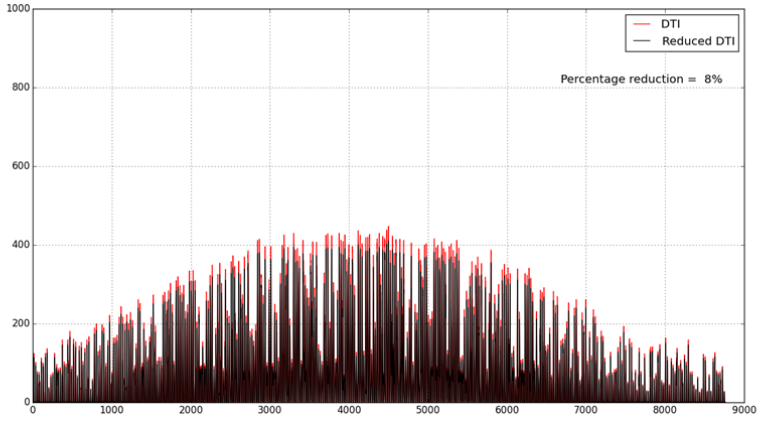


Figure 23: The profile diffuse radiation of the system PV/2 carried out from the measured data on site. (Author: Abdelraheem, Ahmed Khaled Farghaly)

PV/1 Row	Beam Irr. Isol. scenario [kWh/m²yr]	Δ [%]	Global Irr. Isol. scenario [kWh/m²yr]	Δ [%]	Beam Irr. Obstr. Scenario [kWh/m²yr]	Δ [%]	Global Irr. Obstr. scenario [kWh/m²yr]	Δ [%]
5	1634.6	/	2518.5	/	1629.2	/	2506.7	/
4	1544.8	-5.4	2342.2	-8.6	1518.1	-7.7	2250.0	-12.6
3	1530.3	-6.6	2308.4	-9.2	1495.3	-8.7	2156.2	-17.6
2	1526.6	-6.6	2303.7	-9.2	1477.9	-10.2	2115.9	-20.2
1	1528.6	-6.6	2282.3	-10.7	1473.3	-10.5	2105.4	-21.1

Figure 24: Direct and global radiation on the system PV/1 in the isolated and context scenario.

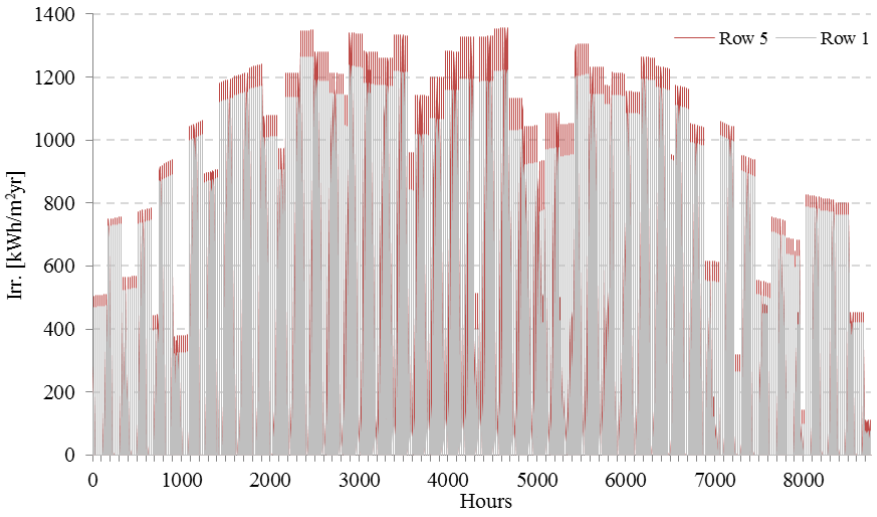


Figure 25: The profile of the global radiation on the row 1 and 5 of the PV/1 in the context scenario.

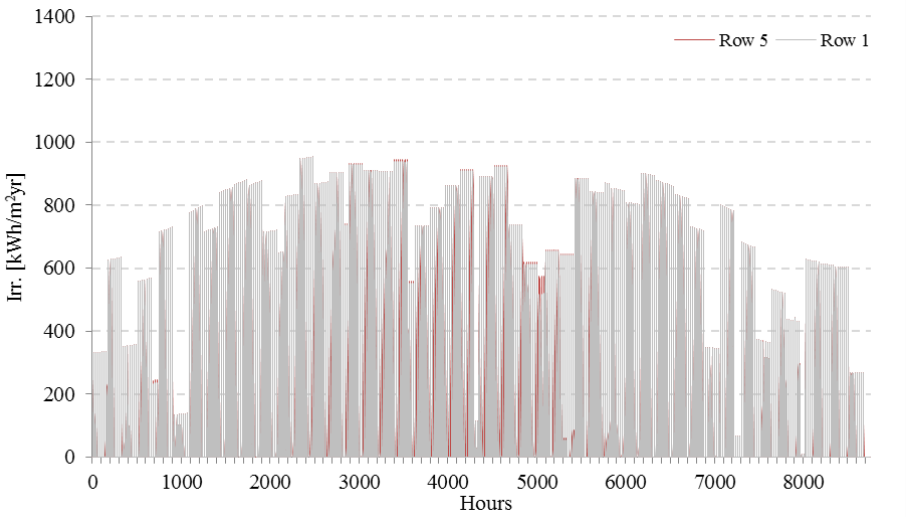


Figure 26: The profile of the direct radiation on the row 1 and 5 of the PV/1 in the context scenario

PV/1 Row	Beam Irr. Isol. scenario [kWh/m²yr]	Δ [%]	Global Irr. Isol. scenario [kWh/m²yr]	Δ [%]	Beam Irr. Obstr. Scenario [kWh/m²yr]	Δ [%]	Global Irr. Obstr. scenario [kWh/m²yr]	Δ [%]
9	1485.9	-1.9	2331.4	-1.7	1485.0	-1.7	2335.7	-1.8
8	1485.7	-1.9	2331.2	-1.8	1484.8	-1.7	2335.2	-1.5
7	1485.5	-2.0	2330.5	-2.2	1484.6	-1.8	2335.0	-1.8
6	1485.5	-2.0	2330.5	-2.3	1484.6	-1.8	2335.0	-1.8
5	1485.6	-1.9	2331.5	-2.2	1484.8	-1.7	2335.7	-2.1
4	1484.6	-1.9	2329.8	-2.1	1483.8	-1.7	2334.8	-2.0
3	1484.6	-1.9	2329.8	-2.1	1483.8	-1.7	2334.8	-1.9
2	1484.6	-1.5	2329.8	-1.7	1483.8	-1.7	2334.9	-2.1
1	1515.0	/	2389.4	/	1512.0	/	2382.2	/

Figure 27: . Direct and global radiation on the system PV2 in the isolated and context scenario..

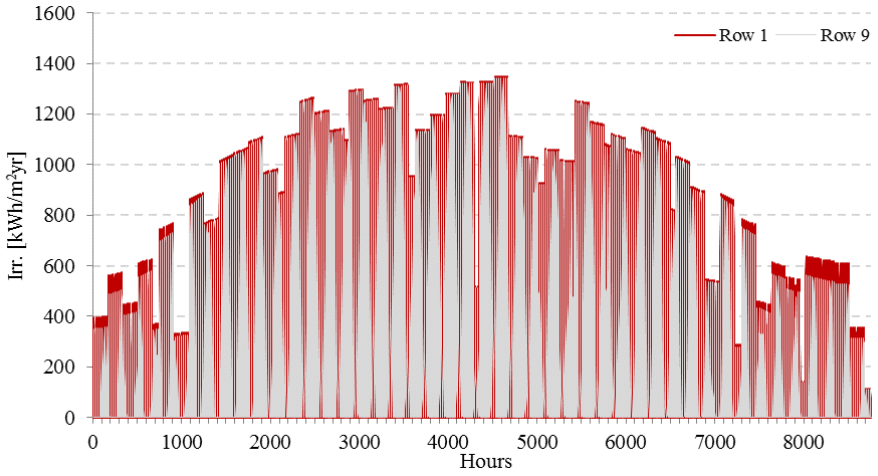


Figure 28: The profile of the global radiation on the row 1 and 9 of the PV/2 in the context scenario.

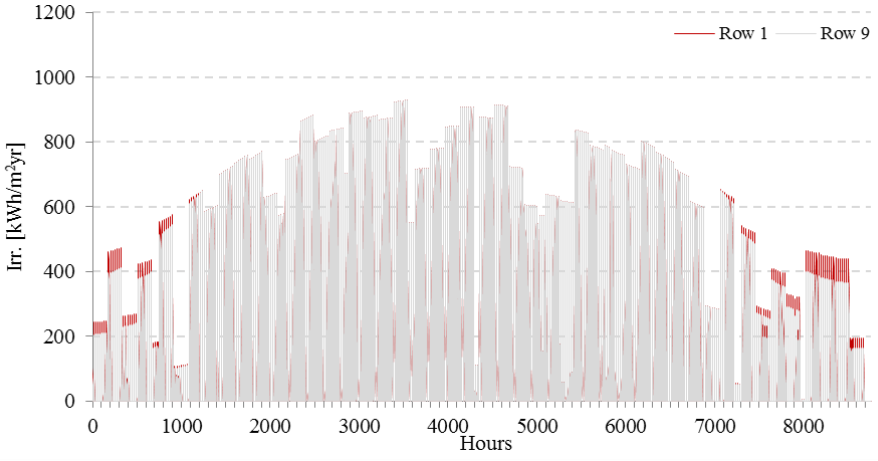


Figure 29: The profile of the direct radiation on the row 1 and 9 of the PV/2 in the context scenario

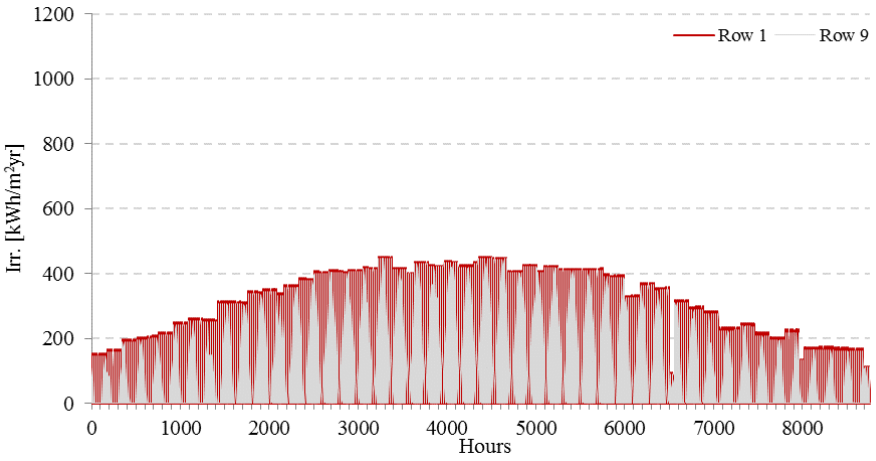


Figure 30: The profile of the diffuse radiation on the row 1 and 9 of the PV/2 in the context scenario

2.2 Automatic task planning

Automatic task planning is a feature that leverages the usage of CoSSMic platform by the user and provide advanced autonomic capabilities to agents. It is based on the start detection mechanism and on default preferences set by the user, who no longer needs to manually schedule a task each time he is switching on the devices. Below, we describe how it works with different kinds of devices.

2.2.1 Automatic task planning for single run devices

In order to use the automatic task planning of single run devices the user needs to set:

- *default program.* This is the default configuration set by the user when the appliance is used. It is necessary to define if it is preferred to delay the execution, or to define a time slot.
- *default delay.* This is the maximum delay allowed to the CoSSMic scheduler for shifting forward the start of the device.
- *default EST and LST.* If these parameters are used the device will start between EST and LST of the current day when the switch-on is detected before LST, otherwise the day after in the same time slot.
- *Program.* It specifies if the default working program set for the related device.

Such parameters are set by the GUI shown in Figure 31. The user will select

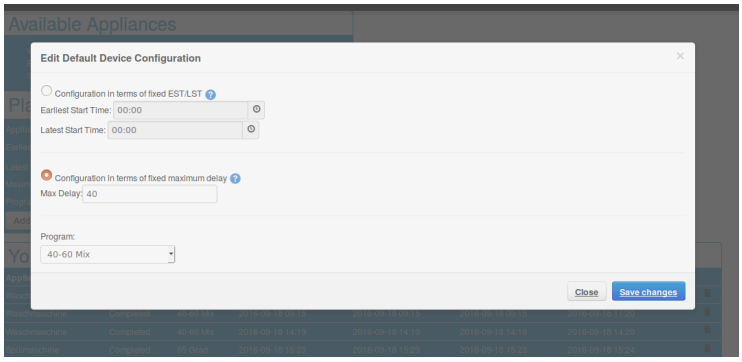


Figure 31: Scheduler interface showing the list of planned, executing and completed tasks.

the radio option to enable the form of interest. After that he will set either the EST and LST parameters, or the delay. At the end of the form he will set the preferred working program of the device.

In particular CoSSMic detects the start of devices. If default parameters have been defined by the user a smart-plug is used to unplug the power. Contextually a task is generated using the default parameters and it is submitted to the scheduler. The system waits for an assigned start time. The smart plug will be switched on again at the assigned start time.

The user can still override the default configuration described before by setting the needed preference only for the next run using the form shown in Figure 32. In this case a planned task is stored in the system and it is found when the device start is detected. In this case the new task is not generated by the default parameters.

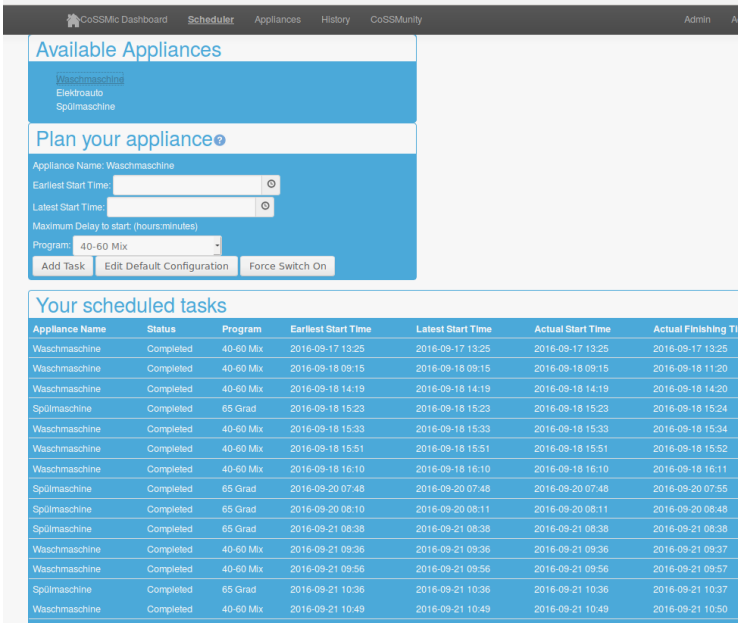


Figure 32: Configuration of default user's preference for task scheduling of a single run device.

2.2.2 Automatic task planning for continuously running devices

The current approach is based on the the working model of many simple internal controllers of continuously running devices. For example, in the case of the fridge, the internal controller switches on and off the device engine when the temperature goes across some specific thresholds. The internal control is able to modify frequency and the duty cycle according to the actual needs. As the devices currently installed at trials sites do not provide open interfaces we can only delay the switch-on of the device disconnecting the power by a smart-plug.

The automatic task planning model designed in CoSSMic uses the following three parameters:

- *default_max_delay*. It is a configuration parameter. It specifies how long the start time of the device can be delayed (to not let the temperature go out of a comfort interval).
- *allowed_delay*. It is a configuration parameter. This the maximum cumulative delay that a device can accept in a day. In fact the internal controller could not be able to recover the effects of all previous delay in a day.

- *accumulated_ delay.* It is a dynamic changing parameter. It is set to 0 at the beginning of the day. It is incremented when the switch on is delayed.

When the switch off is detected by the learning process, a new task is submitted to the scheduler if the accumulated delay is less than the allowed delay. The schedule constraints will be:

- $EST = t_{off_{time}} + off\_duration$
- $LST = EST + min[default\_max\_delay, allowed\_delay - accumulated\_delay]$

The smart-plug is switched off and it will be switched on again at the Assigned Start Time (AST). If the internal control does not start the device right after the AST, it means that the delay has not been used. In the opposite case the used delay (AST-EST) is added to the *accumulated_ delay.*

The appliance configuration form shown in Figure 33 allows to set the configuration parameters. In the CoSSMic trials it has been set by the personnels who installed and configured the platform.

Because of the lack of open interfaces for continuously run devices it has not been possible neither to read the temperature of the internal sensor, nor to overcome the internal controller. The developed approach does not use the temperature (or other parameters) at all and switches the smart-plug according to the introduced parameters valued, which are learned by experience, and are currently not directly related to temperature. For this reason we did not spend effort to design and implement an user's dashboard. The user dashboard will be useful only when we will be able to read the temperature from the device interface and to control directly it.

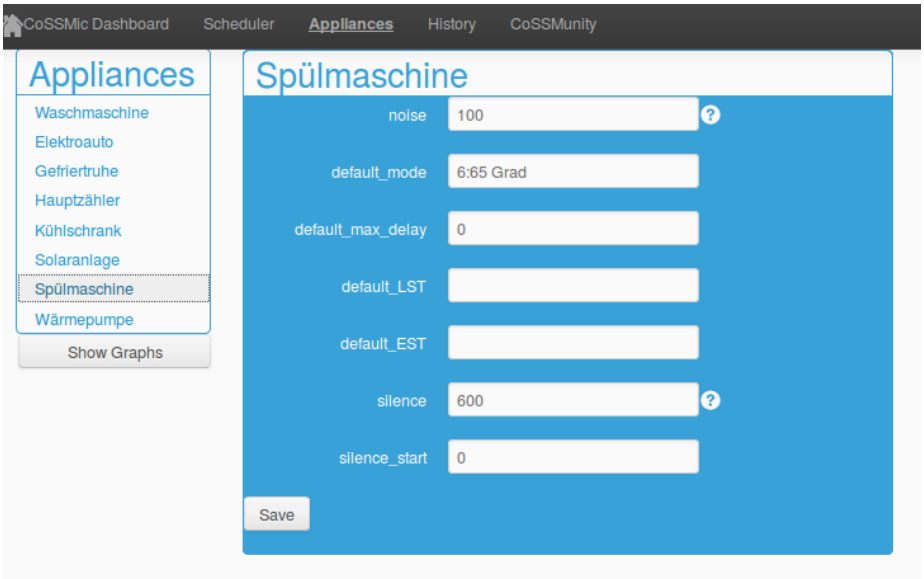


Figure 33: Admin interface for heat-pump configuration

2.2.3 Automatic task planning for e-cars

The automatic task planning for e-cars works according to the technique explained in section 2.1.3. In Figure 34 the default parameters values for the e-car are set.

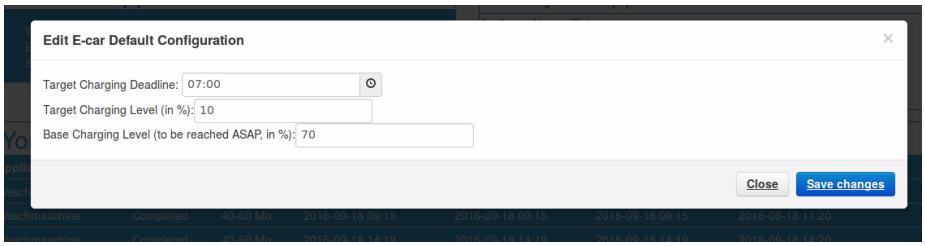


Figure 34: User interface for e-car default configuration

The forms shown in Figure 35 allows for overriding the default parameters values with new ones, which are valid just for the next charge.

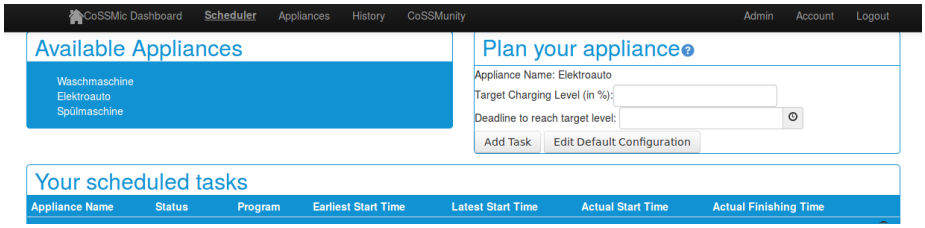


Figure 35: User interface for e-car next charge configuration

2.3 Evaluation tools and validation of results

Validation of results of learning techniques has been done by testing learning algorithms on data collected from trials. SUN developed evaluation tools to process off-line data, for evaluating and improving algorithms and for generating input for the simulation tool developed in WP6.

2.3.1 Trials Backup

Collected data as well as CoSSMic configuration files are periodically uploaded to a remote server for each trial. In particular a backup includes:

- The dump of mysql Emoncms database
- A directory with all time-series. They are stored in different php-time-series binary files.

Some bash scripts have been developed to extract necessary information from the dump of the mysql database and to convert all binary files to csv. Users can process data manually, using spreadsheets, or can do it automatically using the tools presented in the next sections.

In Figure 2.3.1 each directory, named $kn01, ..., kn10$, contains the emoncms database and a sub-directory, name *phptimseries* with binary files. The root directory of the backup, named *trials*, contains the scripts to convert binary files to csv. Figure 2.3.1 shows the results after the execution of the *convertall.sh* script:

1. all the csv files ($feed_x x.MYD.csv$) are created in the related phptime-series subdirectory.
2. all original binary files have been deleted ($feed_x x.MYD$).
3. for each trial two new files $devices\_info.csv$ and $series\_info.csv$ are also created, extracting information from the emoncms.mysql. They respectively contain information about each device and about time-series, such as device type, content of time-series (power, energy, ...).

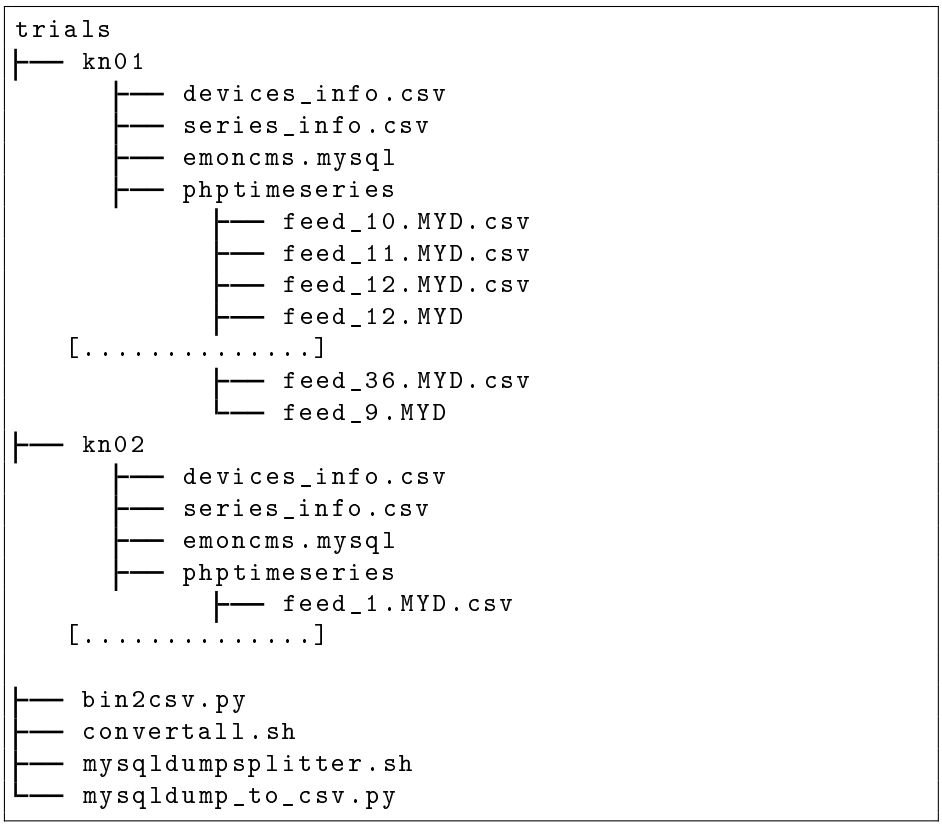


Figure 36: Directory tree of trials backup

2.3.2 Computing loads for single-run devices

A set of utilities allows to process the time series of consumed power for:

- *identifying* the different runs of the device.
- *clustering*, by a supervised technique, the loads belonging to different working modes.
- *applying the B-spline approximation*, to evaluate the learning results and to export the parametric representation of profiles.

Identifying runs of a device

The *cluster.py* script of the learning tools implements this functionality. The man page of the *cluster.py* utility is shown below:

```
usage: %prog [options] ts_filename [-h] [-d NOISE SILENCE]
                                        [-c NCLUSTERS NOISE] [-p]
                                        [-r RUNID [RUNID ...]] [-i] [-v]
                                        FILENAME

positional arguments:
  FILENAME

optional arguments:
  -h, --help            show this help message and exit
  -d NOISE SILENCE, --detect-runs NOISE SILENCE
                        detect runs from file
  -c NCLUSTERS NOISE, --cluster-runs NCLUSTERS NOISE
                        interactive kmeans clustering of detected runs
  -p, --print-images    print clustered runs to png files
  -r RUNID [RUNID ...], --remove-runs RUNID [RUNID ...]
                        print clustered runs to png files
  -i, --interactive     ask the user to delete runs in small clusters (default
                        false)
  -v, --verbose
```

Detections of start-time and stop-time

In the first step we detect each start-time and stop-time of the device, from the beginning to the end of the time-series. In the following example the script gets as an input the csv file, which includes power samples of a washing machine.

```
python cluster.py --detect-runs 40 600  -i -v feed_92.MYD.csv
detected  304 runs
saving to  feed_92.MYD.csv.runs
```

The output file ($feed\_92.MYD.csv.runs$) contains a row for each detected run:

```
0 , 72 ,0, 1444919344
407 , 413 ,0, 1444979993
424 , 530 ,0, 1444981323
577 , 693 ,0, 1444995185
775 , 882 ,0, 1445015768
[...]
```

The different elements of each row are:

- the sample of the time-series that corresponds to the start of the run
- the sample of the time-series that corresponds to the end of the run
- the id of the working mode (all 0 at this stage)
- the start-time of the corresponding the run in *linux epoch* format.

Clustering of working programs

In the CoSSMic approach, the user tells to the platform the working program is being executed, because there is not an available interface to read it by a driver. Maybe in the future we will have device that allows to read also the current working program. However for evaluation purpose we investigated the utilization of clustering techniques for identifying the different working modes from back-up of time-series. Techniques for automatically detecting, on the fly and in real time are out of the scope of CoSSMic research activities, but the issue we address here is a different problem. In fact the time-series to be classified are already all available.

The clustering program uses the *K-means algorithm*, that needs to know the number of clusters we want to build. Each run has been represented with the following features:

- total energy,
- duration,
- peak value,
- peaks number,
- duty cycle (that is the ratio between the time spent over the noise threshold and the time below the noise threshold),
- peaks position.

If we suppose that there are 3 relevant programs for this device, we can trust the algorithm, or set the number of cluster equal to 5 and use the interactive option to supervise the clustering. In the latter case it allows us to search wrong detections, remove them and to repeat the clustering.

An example of supervised clustering follows. First of all we use the $-i$ option, to ask for an interactive detection and to set the number of clusters and the value of the noise. The noise value is necessary to improve the result of clustering.

```
python cluster.py -c  5 40  -i -v feed_92.MYD.csv
```

In Figure 37 different points correspond to different runs of the device, while colors represent different clusters. The x axis represents duration and y axis represents energy.

When the window is closed by the user, the interactive approach shows the runs belonging to the two clusters with the minimum number of runs, and allows to the user:

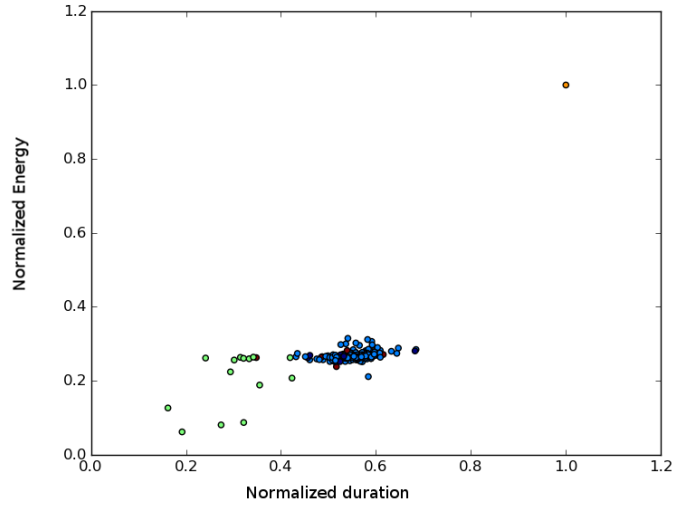
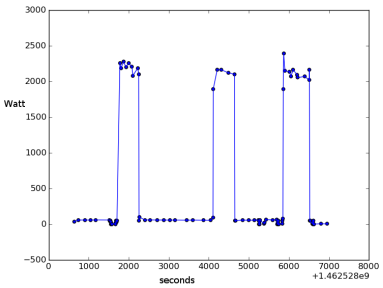


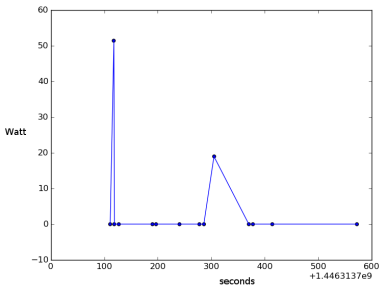
Figure 37: Clustering of detected runs.

- to keep them
- to delete them
- to skip the cluster evaluation and save the results.

In Figure 38(a) and 38(b) the loads of two different runs are shown. The first image is clearly a feasible power profile and is kept. The second one should be deleted.



(a) Power profile of correct detection.



(b) Power profile of wrong detection.

Figure 38: Power profiles of clustered runs.

If the user asks for deleting at least one run, the program re-executes the clustering algorithm . In Figure 39 an excerpt of the console output is shown.

```
evaluate and clusters, evaluation restart if some runs are deleted
evaluating 1/1 individual of cluster 1 ....
0 1462528642.0 1462534960.0
remove run? (y/n/s) n
remove run? (y/n/s) y
evaluating 1/11  individual of cluster 1 ....
0 1452514495.0 1452521013.0
remove run? (y/n/s) n
evaluating 2/11  individual of cluster 1 ....
0 1452866394.0 1452872862.0
[...]
clusters cardinality: Counter({3: 232, 0: 32, 1: 14, 4: 8, 2: 1})
clusters cardinality: Counter({3: 232, 0: 32, 1: 14, 4: 8, 2: 1})
evaluate and clusters, evaluation restart if some runs are deleted
evaluating  1/1  individual of cluster 1 ....
0 1462542998.0 1462549370.0
remove run? (y/n/s) n
evaluating  1/8  individual of cluster 1 ....
0 1450352890.0 1450359197.0
remove run? (y/n/s) n
```

Figure 39: Console output of cluster program

The clustering results are saved in two files:

- *feed_92.MYD.csv.runs* is now updated with the related cluster identification number of each row.
- *feed_92.MYD.csv.features* contains the list of features of each run.

The third column of the runs file is now:

```
0 , 72 ,1, 1444919344
424 , 530 ,3, 1444981323
577 , 693 ,3, 1444995185
775 , 882 ,3, 1445015768
[...]
```

Learning the average profile

The last step consists of the emulation of learning process. It uses the last runs of the same working mode to compute the average energy profile, which corresponds to the prediction of the next energy requirements. The utility man page is:

```
usage: usage: %prog [options] ts_filename [-h] [-lm MODEL] [-pl] [-n N-RUNS]
                                          [-s STEPS] [-i] [-pm] [-v]
                                          FILENAME

positional arguments:
  FILENAME

optional arguments:
  -h, -help            show this help message and exit
  -lm MODEL, -learning-model MODEL
                        interactive kmeans clustering of detected runs
  -pl, -print-learning  print clustered runs to png files
  -n N-RUNS, -history-length N-RUNS
                        use N-RUNS time series for regression
  -s STEPS, -steps STEPS
                        number of learning steps
  -i, -interactive     ask the user to delete runs in small clusters (default
                        false)
  -pm, -print-model    print clustered runs to png files
  -v, -verbose
```

The following command uses the last three runs of the same working mode to compute the energy profile.

```
feed_92.MYD.csv  -n 3 -pl
```

The result is an output file that contains, for each row of the input file, the B-spline parametric representation of the average profile computed from the previous runs. The output file will be named *feed_92.MYD.csv.spline*.

```
 1;1444919344.0;1444924325.0;
  {"k": 3,
   "c": [171.207158, -3.77746695, 429.335, 486.9062, 993.00896, 929.944],
   "t": [0.0, 2348.0, 3924.0, 4981.0]}
 3;1444981323.0;1444987854.0;
  {"k": 3,
   "c": [31.30866, 146.80942, 402.725746, 755.27821, 791.65071, 1318.720],
    "t": [0.0, 1204.0, 4726.0, 6531.0]}
[...]
```

The *-pl* option creates a directory with one subdirectory per cluster. The filename of each image is the starting date-time of the run and contains both the time-series used for learning, and the time-series reconstructed by the B-spline model. In Figure 40 we show a B-Spline reconstruction that approximates the load profile, merging the samples of last three runs. In Figure 40, the black samples belong to first three runs. The red samples belong to the time-series obtained sampling the B-spline approximation.

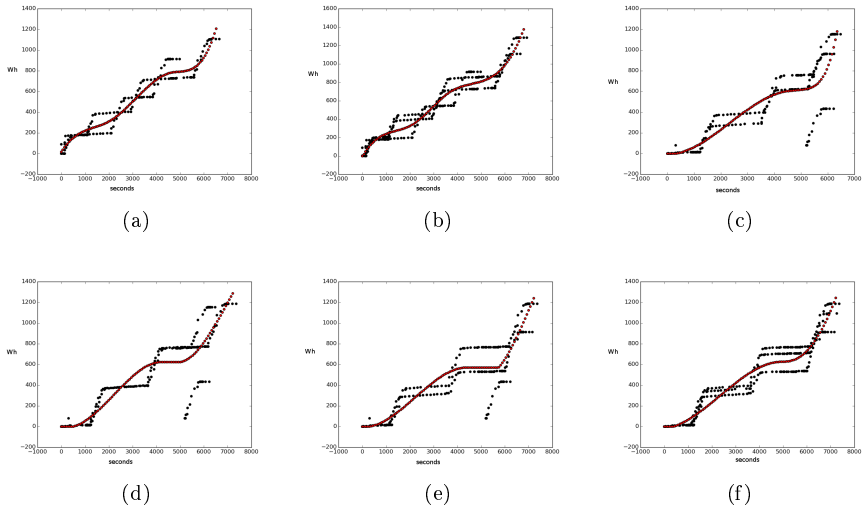


Figure 40: Each chart represents in black the measures of last three monitored loads and in red the learned profile, which will be used for the next schedule.

2.3.3 Computing the background load

The background load is computing splitting the background energy consumed by an household in many chunks of fixed length The functionality is implemented by the *splitloads.py* script. Its man page is shown below.

```
Man page

usage: python  splitloads.py  directory_name
       [-h] [-d DATE] [-n DAYS]
       [-l SECONDS]
       [-p] [-v]
       [-e INPUTID [INPUTID ...]] [-i]

positional arguments:
  directory_name

optional arguments:
  -h, -help            show this help message and exit
  -d DATE, -starting-date DATE
                        yyyymmdd
  -n DAYS, -days-numbers DAYS
                        number of day from starting date (default 1)
  -l SECONDS, -length SECONDS
                        number of seconds of each chunk
  -i, --show-inputs    list of existing devices for this household
  -p, -predictive      predict 24h background load from the previous
        day of the week (default is false)
  -v, -verbose
  -e INPUTID [INPUTID ...], -exclude INPUTID [INPUTID ...]
                        list of input_ids to be exclude
```

Examples

In order to compute the background load for a time-period it needs to download and extract a backup archive in some directory (we assume here */var/trials/kn*10).

The $-i$ options allows to see which are the available consuming inputs for that household:

```
python splitloads.py  /var/trials/kn10 -i

INPUTID:  40 type:  Hauptzahler
INPUTID:  49 type:  Warmepumpe
INPUTID:  53 type:  Elektroauto
INPUTID:  56 type:  Waschmaschine
INPUTID:  59 type:  Gefriertruhe
INPUTID:  62 type:  Splmaschine
INPUTID:  65 type:  Kahlschrank
```

The next step is the computation of background load. Let's suppose we want to compute the background load of April 10th 2016, splitting it in chunks of 20 minutes.

The first input of the list shown before is the household smart meter, which must be excluded from the background computation using the -e option.

N.B. The -e option must be the last one.

```
python splitloads.py  /var/trials/kn10 -d 20160410 -n 1 -l 1200 -e 40
```

```
duration: 1200

task 1 est: 2016-04-09 21:57:22 creation_time: 2016-04-09 21:57:22
task 2 est: 2016-04-09 22:21:22 creation_time: 2016-04-09 22:21:22
task 3 est: 2016-04-09 22:51:12 creation_time: 2016-04-09 22:51:12
task 4 est: 2016-04-10 00:03:22 creation_time: 2016-04-10 00:03:22
task 5 est: 2016-04-10 00:24:27 creation_time: 2016-04-10 00:24:27
task 6 est: 2016-04-10 00:48:18 creation_time: 2016-04-10 00:48:18
[...]
```

The program prints out the list of chunks generated. Duration is not exactly 20 minutes because start and termination of chunks correspond to sample times. Moreover time slots without background load are ignored.

The result is saved in the current directory. In Figure 41 it is shown the cumulative energy (background, self-consumption, total consumption and energy consumed by the devices).

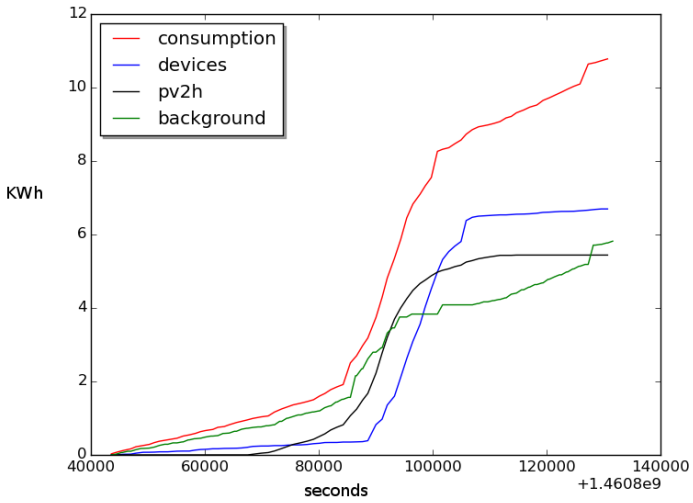


Figure 41: Graphical representation of background cumulative energy and of energy time-series used for its computation.

In Figure 42 it is shown the directory containing the list of loads (kn10_background_tasks.xml) and one file for each load profile.

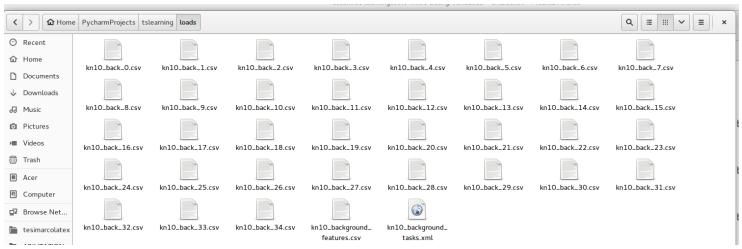


Figure 42: Files generated by the script for the background computation.

N.B.: A reactive approach for scheduling is suggested. Creation time is equal to Earliest Start Time and to Latest Start Time (it cannot be shifted, but a short interval is added to allow for the scheduling of load before the LST expires).

The content of taks.xml looks like:

```
<household name="kn10">
 <device>
   <id>background</id>
   <est> 1460239042 </est>
   <lst> 1460239062 </lst>
   <creation_time> 1460239042 </creation_time>
   <profile>back1</profile>
 </device>
 <device><id>background</id>
   <est> 1460240482 </est>
   <lst> 1460240502 </lst>
   <creation_time> 1460240482 </creation_time>
   <profile>back2</profile>
 </device>

 [...]
</household>
```

The predictive option

The predictive option assumes that the workload of the fixed day is predicted using the one of the same day of the previous week. In this case:

- the loads are computed from data collected 7 days before the defined date
- the creation-time of each loads will be midnight of the defined date.

The option is specified as follow:

```
python splitloads.py  /var/trials/kn10 -d 20160410 -n 1 -l 1200 -p -e 40
```

The content of tasks.xml looks like:

```
<household name="kn10">
 <device><id>background</id>
   <est> 1460239035 </est>
   <lst> 1460239055 </lst>
   <creation_time> 1460160000 </creation_time>
   <profile>back1</profile>
 </device>
 <device><id>background</id>
   <est> 1460240667 </est>
   <lst> 1460240687 </lst>
   <creation_time> 1460160000 </creation_time>
   <profile>back2</profile></device>
[...]
</household>
```

2.3.4 Requirements and installation

The Learning Tool has been developed as a Python package and made available at the bitbucket repository: https://bitbucket.org/cossmic/learningtool. You need just to install python and the required libraries on your machine. In particular you need to install:

- python-numpy
- python-scipy
- python-sklearn
- python-matplotlib

Requirements are satisfied running on any debian distribution the following bash command:

```
sudo apt-get install python-numpy python-scipy python-sklearn python-matplotlib
```

At KN10	10. April-Sun	11 April-Mon	12. April - Tue	13. April-Wed	14. April - Thu	15.April-Fri	16. April-Sat
HP	5.8(on till 10am; 2/h 2kW on/off 1/2)	0.2	0.2	0.2	0.2	0.2	1.5
WM	0	0	0.85 Start-End: 9.25-11.45	0	0	0	1.1 Start-End: 12.30-16.45
DW	0.72 Start-End: 2.15-3.15pm)	0	0.05	0.68 Start-End: 14.15-15.30:	0	0.03	0.03
Fridge	0.4 (2.5/4h; on/off: 1/2;60W)	0.36	0.36	0.36	0.34	0.34	0.34

Table 1: Manually evaluated data for checking the results of the automatic evaluation tools.

2.3.5 Comparison with manual evaluation

Table 1 shows some information about device usage. They have been evaluated manually using data collected from trials.

We will show how validation of results can be performed using the developed tools.

Experiment 1 - background generation

First of all the background load of kn10 from April 10th to April 16th, is computed splitting consumed energy in 20 minutes chunks.

It is obtained executing the command to list the monitored devices in KN10.

python splitloads.py /var/cossmic/e_shape/trials/kn10 -i

- INPUTID: 40 type: Global Meter
- INPUTID: 49 type: Heat Pump
- INPUTID: 53 type: E-car
- INPUTID: 56 type: Washing machine
- INPUTID: 59 type: Freezer
- INPUTID: 62 type: Dishwasher
- INPUTID: 65 type: Fridge

The tool wants to know which of these input must be included into the background. We want to use only the global meter (*option : -e 40*) while the other will be subtracted as they are independently managed by CoSSMic.

Another choice would be to let CoSSMic schedule only the single-run devices and consider all the rest background load. In this case we have to use the -e option to include all periodic devices (-e 40 49 59 65).

python splitloads.py -d 20160410 -n 7 /var/cossmic/e_shape/trials/kn10 -l 1200 -e 40

In Figure 43 the background time-series computed from original measures is shown.

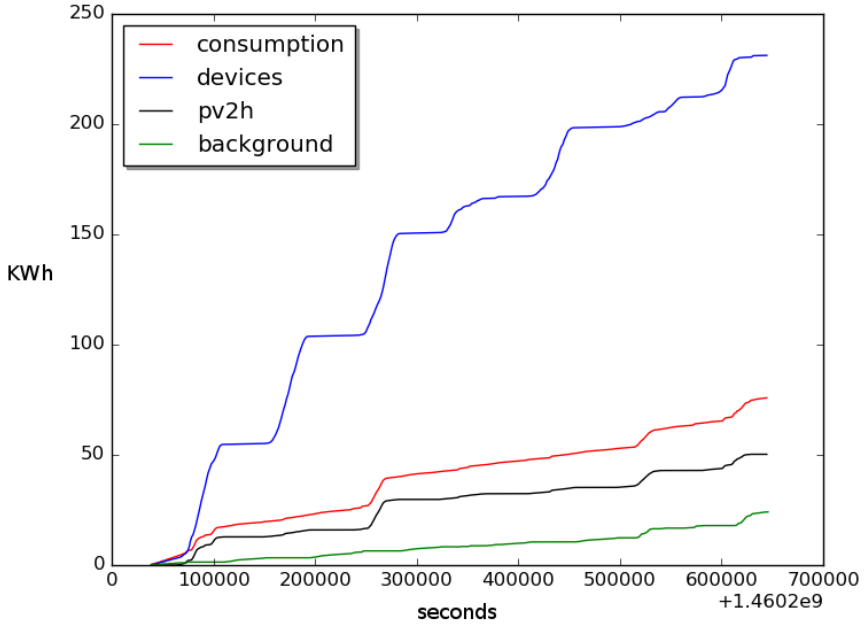


Figure 43: Background energy of kn10 from April $10_{th} - 16th$ 2016

Moreover the same utility creates a directory that includes:

- one file containing the list of chunks ***background_ tasks.xml*** **with the references to their energy profiles**
- for each chunk a file containing the energy profile ***backs[id].csv***
- a file with features of different energy profile.

The directory tree is shown in Figure 44.

In this case Load creation time = EST. Using the -p option, a predictive approach is used. It means that all background chunks will be submitted at midnight for the next day using the measures of one week before.

In this case the command will be:

```
 loads/
├── back0.csv
        [.....]
├── kn10_back_91.csv
├── kn10_back_92.csv
├── kn10_back_93.csv
├── kn10_back_94.csv
├── kn10_back_95.csv
├── kn10_back_96.csv
├── kn10_back_97.csv
├── kn10_back_98.csv
├── kn10_back_99.csv
├── kn10_back_9.csv
├── kn10_background_features.csv
├── kn10_background_tasks.xml
└── tasks.xml
```

Figure 44:

python splitloads.py -d 20160410 -n 7 /var/cossmic/e_shape/trials/kn10 -l 1200 -e 40 -p

The content of file *features.csv* is shown in Table 2.3.5. For each day it provides date, total energy, average energy consumed by chunks, maximum energy consumed by chunks .

	device	kwhd	average kwh	max kwh
2016-04-09 00:00:00	background	1.2911773111	0.6279705917	0.6374338957
2016-04-10 00:00:00	background	17.5974823318	0.7347591788	3.9324843483
2016-04-11 00:00:00	background	5.6227380177	0.2334512423	0.8804279774
2016-04-12 00:00:00	background	17.1194478318	0.7103914725	3.3879164314
2016-04-13 00:00:00	background	5.3025582131	0.221851468	0.8728448276
2016-04-14 00:00:00	background	5.0796341232	0.2116759213	1.2558613859
2016-04-15 00:00:00	background	11.2175012413	0.4616202886	2.0939955593
2016-04-16 00:00:00	background	12.594632984	0.5790266106	2.7092974799

Validation of learning algorithm on KN10 Washing Machine

In this case we will validate the learning algorithm on a single run device. The goal is to detect and evaluate all the loads of washing machine installed in kn10 from April 10^{th} to April 16^{th}. In order to produce also an ouput for the simulator we set a default delay equals to 2 hours (7200 seconds). It means that for each run, the corresponding task to be scheduled will be generated with EST= start-time and LST=start-time + default-delay.

First we show the list of available time-series.

python export-tasks.py -i /var/cossmic/e_shape/trials/kn10

- INPUTID: 41 type: HauptzÃ¤hler consumption

	device	kwhd	average kwh	max kwh
2016-04-12 07:26:44	57	0.8362426013	0.363320464	1.98
2016-04-16 10:32:30	57	0.813938812	0.2802390707	1.9938461914
2016-04-16 14:05:46	57	0.2981780459	0.4607042769	1.9337142334

Table 2: Features of detected loads

- INPUTID: 44 type: HauptzÃ¤hler production
- INPUTID: 47 type: Solaranlage production
- INPUTID: 50 type: WÃ¤rmepumpe consumption
- INPUTID: 54 type: Elektroauto consumption
- INPUTID: 57 type: Waschmaschine consumption
- INPUTID: 60 type: Gefriertruhe consumption
- INPUTID: 63 type: Splmaschine consumption
- INPUTID: 66 type: KÃ¼hlschrank consumption

After that we choose to export the runs of the INPUT 57 (Washing Machine) using the following command.

python export-tasks.py -d 20160410 /var/cossmic/e_shape/trials/kn10 -n 7 -t single-run -delay 7200 -p -ids 57

This facility use the learning tools described in Section 2.3.

If the run detection and learning has not already been done before, file with the runs is not available in the current directory , the script asks for some parameters. If the detection and clustering has been already executed, the scripts uses the available results and simply select and to export the loads of interest.

In the first case, an example of requests from the console are:

- SILENCE parameter(Secs)? 600
- NOISE parameter (Watt)? 40
- SILENCE_START parameter(Secs)? 20

The output of the script includes:

- a set of *csv* files containing load profiles
- images of runs if -p option is used, such as Figure 2.3.5.
- a file with features of loads, containing the information shown in Table 2.3.5 to be checked against Table1.

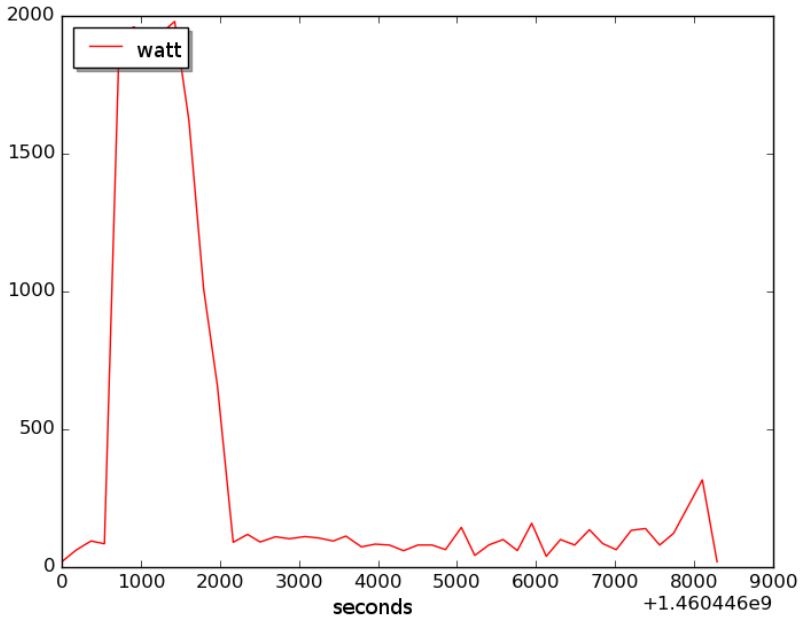


Figure 45: Example of detected load.

2.4 Integration of learning capability in the CoSSMic platform

Development activities focused on the integration of learning functionalities in the CoSSMic platform are tested at trial sites and to provide tools for off-line evaluation.

2.4.1 CoSSMic learning configuration

In order to provide a better performing and reliable implementation the learning capabilities have been implemented as extended functionalities of the Emoncms web application.

Learning functionalities are triggered on each update of the energy inputs by smart meters. The trigger has been developed as a new Emoncms *process*

A process is a function of Emoncms that elaborates a sample when it is received from the driver. Each process executes the implemented function and returns the same sample, or the result of its elaboration. A chain of processes can be attached to an input stream. The result of the elaboration of a process is passed as input to the next process. In the example a "log" process saves the sample into a time-series and returns the sample itself. The "kwh2power" process computes the power value from the last two energy samples, stores the power value into a time-series and returns to the next process the power value.

In order to enable the learning process for a device, it needs to add the p_learn process to its input "EnergyIn" just after the kwhpower process. The p_learn process needs that input itself as parameter. In Figure 46 the chain of processes for some devices of KN10 is shown. In particular the energy input is updated by a sensor with the cumulative energy consumed.

The chain of processes stores the energy value, computes the average power from the previous sample, forwards this value to the learning process and finally updates the daily consumed energy. The learning process can be attached to any input according the Emoncms model. When a new device is added to the micro-grid configuration, the process is automatically added to the chain if the device type is single-running or continuously-running.

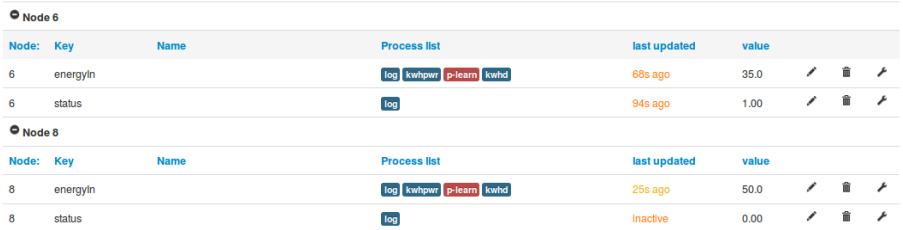


Figure 46: Learning process attached to Emoncms inputs

The status of the learning process is stored in Redis as a tuple of elements with key $profiles : \$inputid$. To restart the learning process, it needs to FLUSH the Redis memory or to delete the corresponding key.

The p_learn process calls API of MAS module in order to monitor the life-cycle of the device.

In particular the software implementing the learning capabilities presented in the previous sections are coded in the CoSSMic TaskManager (mas Module of Emoncms). The module is organized according to the structure shown in Figure 47.

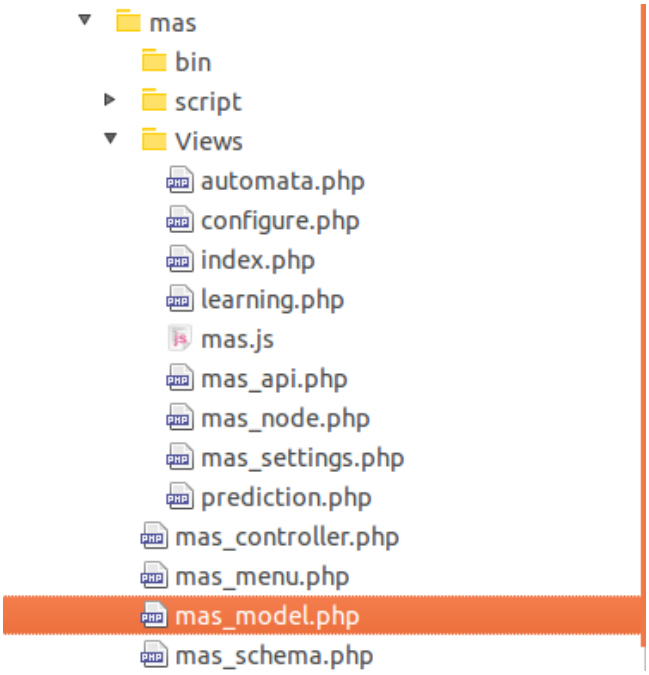


Figure 47: Structure of *mas* module in Emoncms

In particular:

- *view/mas_learning.php* provides a web list of devices the system is learning.
- *view/automata.php* provides a graphical view of the learning process for the specific device.
- *mas_model.php* integrates learning capabilities as methods of the mas software module.

The implemented methods added to the mas_model.php package are shown in Figure 48:

- *updateLearning*. It is the method triggered by the arrival of incoming samples. According to the kind of device it uses the related functions.
- *updateSingleRunLearning*. It is the method that implements the learning behaviour for single-run devices.
- *updatePeriodicLearning*. It is the method that implements the learning behaviour for continuosly-run devices.

- *loadTemplate*. It is a method that retrieves the status of the learning behaviour for the related device. The status of the device is saved in the Redis Key-Value store, shared with the Emoncms web application, to improve performance and to guarantee persistence.
- *updateProfile*. It is a method that uses the historical information to update the average profile after each run of a device.
- *updateSinglRunProfile* is a specialization of updateProfile for single-run devices. This method is supported by a python script that implements the regression and approximation models presented before.

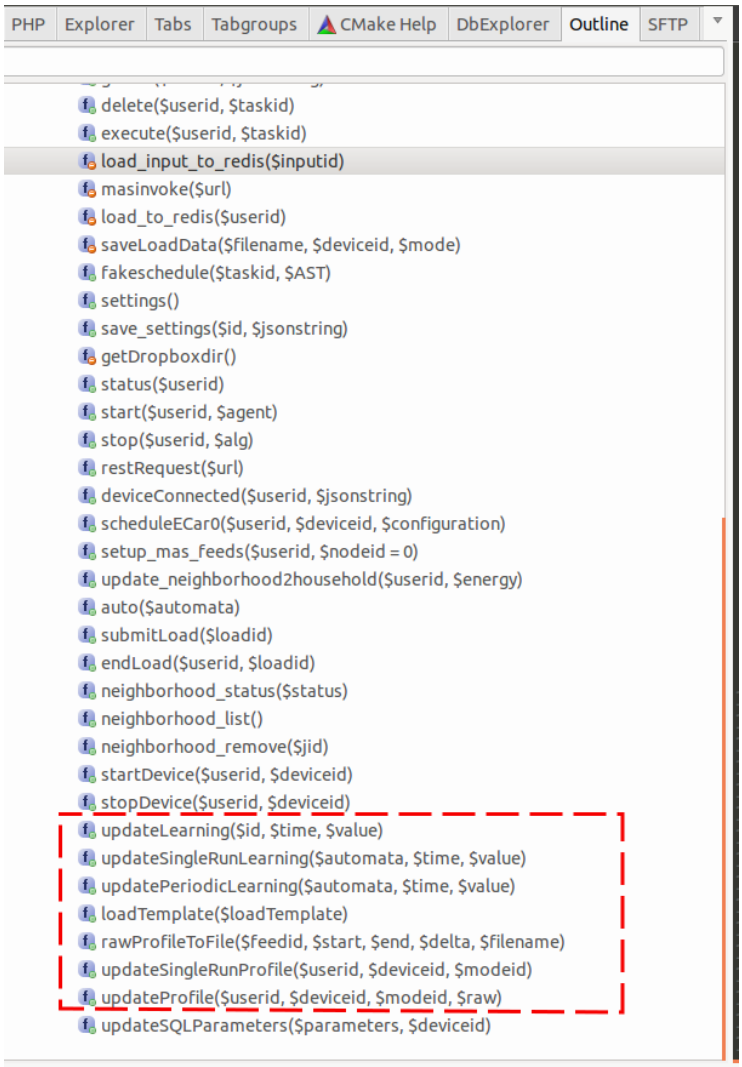


Figure 48: PHP methods implementing learning algorithms.

The load profile is stored in two files:

- /var/www/emoncms/profiles/u[$userid]d[$deviceid]m[$modeid].prof (it is a symbolic representation of the profile according to the B-Spline model defined before)
- /var/www/emoncms/profiles/u[$userid]d[$deviceid]m[$modeid].prof_raw (it is a time-series representation of the profile)

To compute the profiles the last 5 tasks, which are in completed status (=5) are queried from the mysql table *user_tasks*. In Figure 2.4.1 the query results are shown. The ast and ltime parameters correspond to start time and stop time of each load.

```
select id, ast,ltime from  user_tasks
   where deviceid=1 and modeid=13
   and ast <> 0 and status=5 order by ast desc limit 5

+----+------------+------------+
| id |    ast     | ltime      |
+----+------------+------------+
| 40 | 1467193140 | 1467193335 |
| 38 | 1467189840 | 1467189869 |
| 37 | 1467189540 | 1467189583 |
| 36 | 1467189180 | 1467189242 |
+----+------------+------------+
```

Figure 49: Example of start time and stop time for 5 loads in Emoncms database.

The energy consumed after ast and before ltime are read and extracted from the energy time-series (emoncms feed) and saved into a temporary csv file. Then a python program (/var/www/emoncms/Modules/mas/bin/updateprof.py) produces two files with the symbolic representation and the raw time-series of that load (u1d1m13.prof, u1d1m13.prof_raw).

The status of each learning process is saved in the Redis key value store. In this way we save all static and dynamic parameters in memory, improving performance and reducing the overhead due to the usage of database and file-system. This solution also increases the life of the secure digital memory storage of the raspberry.

In order to monitor the status of the learning processes we developed a web interface whose link is hidden to the user, but available to the administrator. The first page, available at *http* : *//localhost/emoncms/mas/learning.html*, is shown in Figure 50. It shows the list of existing processes and gives information about the related devices, the working programs and the status of each device.

Moreover the web page, for each learning process, allows to view: the time-series of the last profile learned for all working modes of the corresponding devices and the status of the automata implementing the process behaviour.

In Figure 51 the time-series shows the load profile for the selected device and specific working mode. Only the working mode already learned is shown by the select widget. It is shown the average profile that has been or will be scheduled.

Figure 50: List of active learning processes.

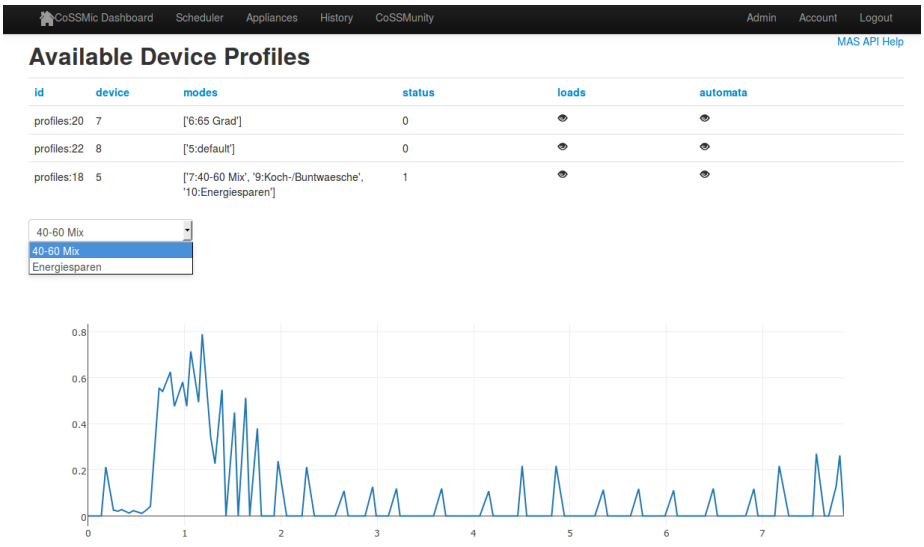
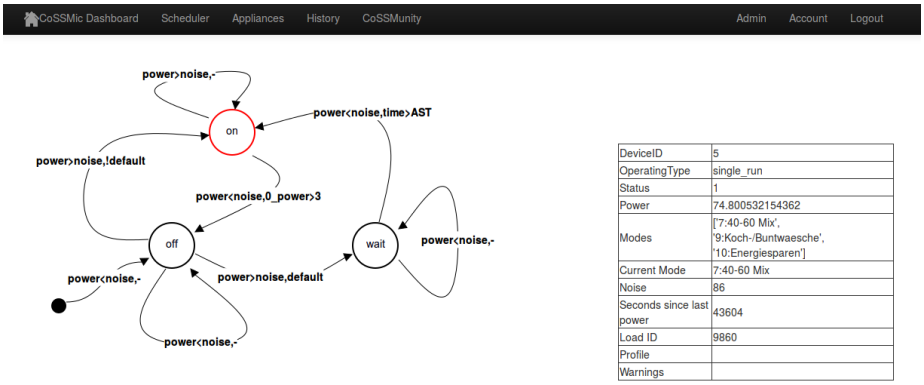


Figure 51: Example of learned profile. This has been scheduled or will be at next switch of the device.

In Figure 52 the automata of the learning process is shown graphically. The current status is colored in red and the table shows the values of the main parameters stored in Redis.



DeviceID	5
OperatingType	single_run
Status	1
Power	74.800532154362
Modes	['7:40-60 Mix', '9:Koch-/Buntwaesche', '10:Energiesparen']
Current Mode	7:40-60 Mix
Noise	86
Seconds since last power	43604
Load ID	9860
Profile	
Warnings	

Figure 52: Graphical visualization of the learning automata.

2.4.2 Integration in the simulation environment

The simulator uses the learning tool presented before to create configurations from back-up of trials. This is used to reproduce a situation as similar as possible to the real trials, from which the user can start to further specify or change the configuration. In fact the simulation uses the CoSSMic scheduler to evaluate different testing conditions (number of devices, different flexibility, different devices, multiple runs, ...).

In Figure 53 it is shown the simulator GUI by which the user is selecting the households and devices to be used for the configuration of simulation. In particular the user is selecting the continuously-run devices to include them into the background load and to schedule only single-runs.

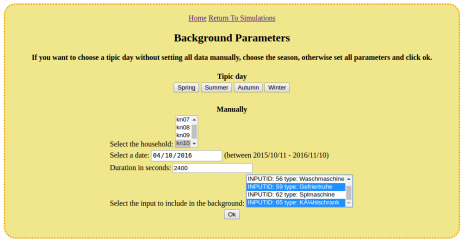


Figure 53: Simulator GUI. Continuously run devices are included into the background load.

In Figure 53 the user is configuring the background load in a way that only the global meter is not subtracted from the total consumption. The parameters of the form are the same explained when we illustrated the command line interface of the tools.

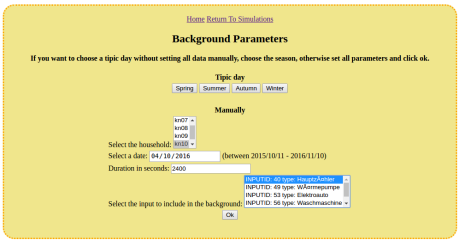


Figure 54: Simulator GUI. Continuously both single-run devices and continuously-run devices are scheduled and not included into the background load.

3 Optimisation Algorithm

The task scheduler solves the problem of assigning start times to loads whose earliest start time and latest start time are provided by the user, and there are load profiles available describing the cumulative energy consumption of the load. For the PV producers there must be production prediction files describing the cumulative energy production by the producer.

The Consumer Agent represents one load profile, and the user parameters for its earliest start time and its latest start time. The PV Producer represents a photo voltaic (PV) panel characterised by the fact that it has a prediction of the energy it will produce based on the weather forecast for the location of the PV panel.

In order to be scalable and resilient to households joining or leaving the neighbourhood, each Consumer Agent acts independently. Each Consumer Agent tries to allocate energy for its load by:

1. Selecting a producer
2. Letting the producer solve the related optimisation problem to assign a start time to each of the associated loads.

These mechanisms are described in the following sections.

3.1 Negotiation: Selection of producers

When a load is submitted, a Consumer Agent is created in the system, and it will select a producer to provide the energy needed by the load. If this producer has sufficient energy according to its prediction to start the load between the load's earliest start time and its latest start time, it will assign a start time to the load. If not it will refuse the allocation, and the Consumer Agent will select another producer.

It could happen that this new load will make a more optimal consumption of a producer's predicted energy production than its current set of load assignments. In this case the producer could cancel any of the previously assigned loads. Each of these rejected Consumer Agents will then have to select other producers to serve their needs.

The selection process repeats until every load has assigned a start time from a producer. As the Consumer Agents act autonomously, this is a game where each play corresponds to a consumer selecting a producer, and the epoch of the game is the number of plays needed to have a new solution when the system is perturbed by either a new prediction or the arrival of a new load to the system. A particular set of assignments,*i.e.* bindings of loads to producers, is called a *configuration* of the game.

The game will always converge to a configuration provided that the grid accepts to start any load within its allowed start time interval given by the load's earliest start time and its latest start time. This condition must be met even if one uses a grid model that is artificially limited in order to avoid peaks, meaning that peak avoidance cannot be guaranteed if there is no feasible schedule for the loads that have selected the grid as their producer under the maximum grid peak limit. Furthermore, the game is cooperative [5], because all the involved consumers and producers jointly try to minimise the grid energy consumed by the neighbourhood.

A consumer's selection of a producer is carried out using a *variable structure stochastic automaton* (VSSA) [9]: A consumer has a probability vector $\mathbf{p} = [p_1, \ldots, p_n]^T$ with one probability for each producer, the grid inclusive. This vector represents a probability distribution with $\sum_i p_i = 1$. For each play, the Consumer Agent selects the candidate producer according to this empirical probability distribution, and the available producers are called *actions* in the field of *Learning Automata* (LA) research, a branch of reinforcement learning [14]. In order to ensure that the PV producers are evaluated first, a reduced action set automata is used where the actions are initially the set of all PV producers, but when a selected producer fails to provide energy for the load, it will be excluded from the reduced action set. Should the system be loaded, and all PV producers refuse the load, the restricted action set will eventually become empty after which the grid is the only candidate producer and will be selected with probability one.

When the configuration is rewarded, the action probabilities will be changed according to some algorithm that will increase the probabilities for the producers that normally tend to accept the load, and reduce the probabilities for the others since the probabilities of the empirical distribution must sum to unity. The probability update proposed by Pozniak and Najim [1] is used for the reduced action set probabilities.

3.2 Optimisation: Assigning start times

Classical scheduling originated in manufacturing disciplines and considers the problem of *assigning* a set of n jobs onto m machines. Each job j is assumed to have a known processing time on machine i. It should be noted that classical scheduling only implicitly considers the resources provided by the machine, *i.e.* the capacity of the machine is reflected in the time it takes to complete the job on that machine.

The situation considered here is different in that the "machine" is a PV panel that provides time variant resources, and the scheduling problem is to start the time variant "jobs" that are the assigned loads according to the resource availability on the "machine". The load profiles are *continuous* and once a load has started it will have to run to completion,*i.e.* the problem is a nonpremptive single-machine scheduling problem. In contrast to classical scheduling problems, the PV Producer Agent may start two or more loads with overlapping execution periods if the predicted production profile allow this.

In contrast to the cominatorial assignedment prolem, a *relaxed* form of the problem is considered here where it is possible to acquire additional resources for PV Producer Agent to supply the loads with energy since it can supplement with energy from the grid. This will guarantee that the problem has a solution, and transform the problem to find the schedule that minimises the cost of the additional resources, *i.e.* the grid energy.

Each PV Producer Agent solves independently an optimisation problem to find the schedule for the loads assigned to it given its predicted energy production. Hence, the following text refers to the algorithm run independently by each of the PV Producer Agents in the neighbourhood.

3.2.1 Consumption intervals

The schedule is fundamentally a vector $\mathbf{s} \in \mathbb{R}^n$ that assigns a *start time* s_j to each of the n loads accepted by the PV producer. Each load has a *duration* Δ_j. Since the loads are nonpremptive, the continuous interval for which a load j consumes energy is referred to as the load's *consumption interval* and it is denoted $\mathbb{I}_j = [s_j, s_j + \Delta_j] \subset \mathbb{R}$.

One or more other loads can be started within a given load's consumption interval. Thus, in the extreme, all consumption intervals will overlap and there will be only one consumption *period*. In the general case, however, there will be several disjoint consumption periods where one or more loads are executed in each period. Formally, a given schedule $\mathbf{s}$ will partition the loads' index set, $\mathbb{L} = \{1, \ldots, n\}$, into disjoint index sets $\mathbb{P}_{k|\mathbf{s}}$ such that

$$\bigcup_k \mathbb{P}_{k|\mathbf{s}} = \mathbb{L}$$

Each of these consumption periods correspond to a consumption interval $\mathbb{I}_{k|\mathbf{s}}$ being the union of the consumption intervals of the loads in the consumption period.

Each load's consumption profile is a *continuous* function whose value represent the cumulative amount of energy consumed by the load at a given time, *i.e.* $L_j^0(t)$ for $t \in [0, \Delta_j]$. The amount of energy consumed between two time stamps $0 \leq t_1 \leq t_2 \leq \Delta_i$ is therefore $L_j^0(t_2) - L_j^0(t_1) \geq 0$.

Assigning a start time s_j to a load will have the effect of shifting its energy consumption in time. There will obviously not be any energy consumed before the load starts, and no further energy will be consumed when the load finishes. Hence, the load's cumulative energy consumption conditioned on a given the schedule $\mathbf{s}$ is

$$L_j\left(t|\mathbf{s}\right) = \begin{cases} 0 & t < s_j \\ L_j^0\left(t - s_j\right) & s_j \leq t \leq s_j + \Delta_j \\ L_j^0\left(\Delta_j\right) & s_j + \Delta_j < t \end{cases} \tag{3}$$

Based on this, it is straight forward to define the total energy consumption in a consumption period as the sum of the energy demands of the various loads scheduled for that interval. Hence,

$$L_{\mathbb{I}_{k|\mathbf{s}}}(t) = \sum_{j \in \mathbb{P}_{k|\mathbf{s}}} L_j\left(t|\mathbf{s}\right) \tag{4}$$

In order to schedule the loads according to the predicted energy production of a PV panel, the prediction must be known as a continuous and cumulative function, $R(t)$. Again, this implies that the function is strictly increasing with the property that $R(t_1) \leq R(t_2)$ if $t_1 \leq t_2$.

3.2.2 Objective function

A *feasible schedule* $\mathbf{s}$ satisfies

$$L_{\mathbb{I}_{k|\mathbf{s}}}(t) - \left[R\left(t\right) - R\left(\min \mathbb{I}_{k|\mathbf{s}}\right)\right] \leq 0 \quad \text{for all } t \tag{5}$$

The last term in the bracket reflects the fact that the predicted energy production $R(t)$ is cumulative, and only the energy being produced over the consumption interval can be used to execute the loads scheduled for this interval. It is therefore necessary to subtract the cumulative amount of enargy produced up to the start of the consumption interval, implying that the expression in the bracket is zero at the beginning of the consumption interval.

The assumption (5) is an absolute requirement if it is not possible to change the provisioning of energy. It is a *goal* in situations where additional energy can be obtained from the grid, albeit possibly at a high cost.

In this relaxed problem, one could therefore face situations where assumption (5) momentarily does not hold. In these cases it would be desirable for the total energy consumption over the entire interval not to exceed the available energy produced over that interval. In other words, one would integrate assumption (5) over the whole consumption interval:

$$\int_{\min \mathbb{I}_{k|\mathbf{s}}}^{\max \mathbb{I}_{k|\mathbf{s}}} \left[L_{\mathbb{I}_{k|\mathbf{s}}}(t) - \left[R\left(t\right) - R\left(\min \mathbb{I}_{k|\mathbf{s}}\right)\right]\right] \, \mathrm{d}t \leq 0 \tag{6}$$

The scheduling problem is therefore a standard non-linear mathematical programming problem [3] aiming at minimising the external grid energy requirement:

$$\min_{\mathbf{s}} \sum_{k} \int_{\min \mathbb{I}_{k|\mathbf{s}}}^{\max \mathbb{I}_{k|\mathbf{s}}} \left[L_{\mathbb{I}_{k|\mathbf{s}}}(t) - \left[R\left(t\right) - R\left(\min \mathbb{I}_{k|\mathbf{s}}\right)\right]\right] \, \mathrm{d}t \tag{7}$$

subject to the constraints on the starting times for the jobs.

It is important to note that despite the fact that the value of the integral (6) only depends on the consumption interval for which it is evaluated, the actual schedule $\mathbf{s}$ is common to all consumption intervals, and the schedule $\mathbf{s}$ defines the partitioning of the load set. The problem is therefore not separable, *i.e.* one cannot interchange the sum and the minimisation to solve the problem as a sum of smaller minimisation problems.

Even though the optimisation problem cannot be separated, the functional form (7) can be significantly simplified rendering the problem tractable for numerical solution, and it can be proved that the following holds: The minimisation problem (7) is independent of the loads' temporal energy consumption, and the non-linear mathematical programme can be written as

$$\min_{\mathbf{s}} \left[\sum_{k} \sum_{j \in \mathbb{P}_{k|\mathbf{s}}} L_j^0 \left(\Delta_j\right) \left[\max \mathbb{I}_{k|\mathbf{s}} - \left(s_j + \Delta_j\right)\right] + \sum_{k} \left(R\left(\min \mathbb{I}_{k|\mathbf{s}}\right) \left[\max \mathbb{I}_{k|\mathbf{s}} - \min \mathbb{I}_{k|\mathbf{s}}\right] - \int_{\min \mathbb{I}_{k|\mathbf{s}}}^{\max \mathbb{I}_{k|\mathbf{s}}} R(t)\,\mathrm{d}t \right) \right] \tag{8}$$

and solved subject to the constraints on the start times.

Any kind of standard non-linear solver can be used to find the schedule $\mathbf{s}$ that minimises (8). CoSSMic uses Powells algorithm for solving constrained optimisation problems by linear interpolation [11] as it seems to be one of the better gradient free algorithms [12].

3.3 Reinforcement learning: Computing the rewards

Shapley proved the optimal fair distribution of value of the game to each player in a cooperative game [10]. In CoSSMic the Shapley values attributed to each Consumer Agent and Producer Agent adds up to the value of the grid energy saved by the neighbourhood. Note that even though the consumption of energy is only done by the Consumer Agents, the Producer Agents should have their share of the saved energy cost to the neighbourhood since they enable the savings to be made.

Unfortunately, it is an NP-complete problem to compute the Shapley values, but in the case where the values are produced by pairwise relationships, the Shapley value is half the sum of the weights on the edges incident with a node in the relationship graph [17].

When one of the Consumer Agents terminate, its consumed PV energy is measured, and the edge weights in the neighbourhood's pairwise relationship graph is updated with this new value. The Shapley value is then computed for each node in this graph, *i.e.* for each Consumer Agent and each Producer Agent, and they are all rewarded. It may seem strange that agents that are not part of the finalised transaction are rewarded. The argument is that a *configuration* is a set of active pairwise assignments, and when one of the Consumer Agents terminate a *new* configuration results. This new configuration is identical to the previous one, except for the now terminated Consumer Agent. This marks the end of the previous game epoch, and a reward for this epoch should be given to all players.

A result of this is that the a Consumer Agent may be rewarded repeatedly for its choice of a Producer Agent as the other Consumer Agents finishes, even if its own choice of producer does not change. The positive side effect of this is that it will speed up the learning of the good producers since each Consumer Agent can be rewarded many times for the same choice of "action". However, more research is needed to understand the potential impact of this on the Learning Automata (LA) updating algorithms since their convergence is proved based on the repeated pattern of an action selection to be followed by a probability vector update, which is again followed by an action selection based on the updated probability vector. The last part is violated by the CoSSMic approach as the next action is unchanged, and hence identical to the one selected according to the probability vector as it was several rounds of feedback ago.

A strength reward to a learning automaton is a number in the interval $[0, 1]$, and hence twice the Shapley value for each Consumer Agent is normalised on the total PV energy consumed by the neighbourhood at the time of computation. The Shapley value is multiplied by two since the original division by two was a result of the double counting of the edge weight in the relationship graph – once for each end of the edge – so that the sum of Shapley values for all nodes will equal the Shapley value for the whole game. It is not necessary that the rewards will add up to a given number. The reward then indicates how much each Consumer Agents probabilities for the producers will be changed.

The reward to the household will be computed as the sum of the Shapley values (not multiplied by two) for all the active Consumer Agents in the household and all its Producer Agents. This is number is then divided by the total PV energy consumption in the neighbourhood in this game epoch to give an index for the houshold's relative performance in the neighbourhood.

3.3.1 Emulation

Even if the validation of the taskscheduler algorithm has been done by the simulator developed in WP6, the test of the integration with the CoSSmic platform has been done by a new facility developed in the last year of the project. In particular we provided a methodology and a tool that allow to restore a trial and to use the historical data for emulating at the current time one ore more days in the past

In order to emulate a trial, first of all it needs to restore a backup.
A backup contains:

- the emoncms database the emoncms settings.php file.
- the directory /var/lib/phptimeseries with binary files containing timeseries.

It is suggested to backup the current installation of emoncms firt. Moreover if you do not know username and password of trial user, the current installation could be used to replace it.

After the local data have been saved it needs to perform the following steps:

- Drop the emoncms database using mysql.
- Create again an empty emoncms database
- Import the emoncms database from the backup
- Delete the content of /var/lib/phptimeseries
- Copy the files from the phptimeseries directory of the backup to /var/lib/phptimeseries

After this, it needs to avoid that the current installation starts to synchronize with the remote server. Then open the file settings.php of emoncms and delete, or change, the server APIKEY to avoid sinchronization as follows:

```
#!php
$cossmic['apikey'] = '';
```

Now it needs to know the username and password of backup trial to use emoncms, otherwise there are two alternatives:

1. copy password and salt fields from users table of your old emoncms database to the current one.
2. generate a new value for the password field and insert into users table. In particular for this option using the linux bash:

```
#!bash
$ echo -n mypassword | sha256sum
  89e01536ac207279409d4de1e5253e01f4a1769e696db0d6062ca9b8f56767c8  -
$echo -n [salt]89e01536ac207279409d4de1e5253e01f4a1769e696db0d6062ca9b8f56767c8
  | sha256sum
  1d5eed48874df313375187ec1c70e90ac96514dd5b8208b25aba69b8a17738dc  -
```

where [salt] should be replaced with the current value in the users table of emoncms database.

The emulation is implemented by the *TimeMachine* agent presented in the next section. This agents reads old samples from the time-series and write them into the emoncms again. To let TimeMachine emulate the stop and the start of devices, it needs to update the driver of all devices. It can be done by mysql executing the following query:

```
#!sql
update node_parameters set driver="tester"
```

Finally MAS Configuration must be updated with the current APIKEY.

The TimeMachine Agent

This Agent is conceived to emulate a list of configured devices using past samples from the time-series. Input of the TimeMachine are:

- a configuration file (tm.conf)
- a directory with binary php time-series previously recorded by Emoncms.

The implementation currently only works with cumulative energy inputs.

The Configuration File of TimeMachine must be named *tm.conf* and stored in the mas directory. It is composed of multiple lines, one line per input. Each line must specify the following parameters separated by a semicolon as delimiter:

```
#!csv
[nodeid];[inputname];[timeseriesid];[startingtime];[deviceaddress];[deviceid]
```

- nodeid: is the node where the measures of this timeseries must be written
- *inputname*: is the name of the input
- *timeseriesid*: it is the id of the corresponding time-series (emoncms feed) and it is used to identify the binary file where the data are stored (feed_id.MYD)
- *startingtime*: it is the date-time in the past from which we want to start the emulation of the corresponding time-series
- *deviceaddress*: it is the address of the device, used by emoncms to send command by the driver.
- *deviceid*: it is used by TimeMachine match address and device

An example of line in the configuration file is:

```
#!csv
4;energyIn;56;201611071025;GPIO-Relais9;5
```

We want to emulate the stored in time-series 56, writing it to energiIn input of node 4 in emoncms. The corresponding device has ID=5 and address=GPIO-Relais9.

At startup TimeMachine parses the configuration file. For each line it:

- search the closest epoch to the start time in the binary file

- for each input it gets the last sample from emoncms (to emulate the past cumulating energy)
- it reads the next sample in the time series, computes the time to the next sample and sets a time-out
- the time-out triggers the write of an energy increment, the reading of the next sample and the set of a new time-out

When a stop command is received for a device the agents always writes the last sample and waits for a time-out. When the start is received it continues to read samples from the time-series according to the steps listed before.

4 Distributed Payoff Calculation

4.1 Visualization of distributed payoff

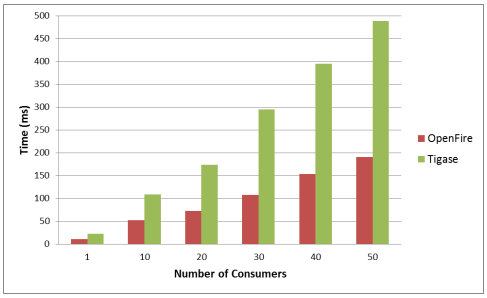
In order to visualize both the exchange of energy between the household and the neighborhood three inputs, and related feeds, are updated by the mas module of CoSSMic:

- The *neighborhood2household* is a cumulative energy feed. It contains the amount of energy that the household has consumed from all the PVs of the neighborhood. It is updated at the end of each load. When the device is not consuming anymore, and the end of the execution has been detected, the mas module checks: 1) if that load was scheduled by CoSSMic and 2) which vendor provided the energy. If the vendor is not the grid this feed is incremented with the amount of consumed energy.
- The *pv2neighborhood* is a cumulative energy feed. It contains the amount of energy that the PV of the current household has provided to this household and to all the neighbors. The containing information is known by the producer agent that shares the information about the energy provided to the neighborhood with all the other producers, and uses it to compute the distributed payoff. As this information is updated in a local file, the Task Manager agent periodically reads from the file and writes the updated data into this feed.
- The *reward* feed is updated periodically according to the same approach used for the previous item. However it is restored at the beginning of each day and decreases until the end of the same day. The feed is updated when other producers share additional energy with the neighborhood.

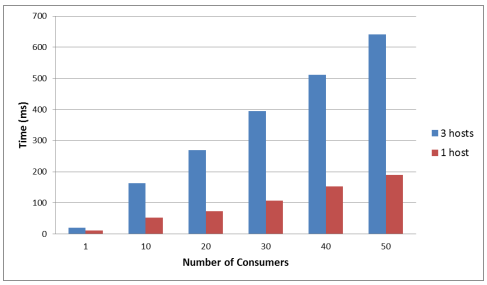
4.2 Scalability of Communication protocol

To validate the proposed architecture and to test the performances of the P2P communication overlay built for the neighbourhood communication, we set some experiments based on the proposed infrastructure. REST performances have already been discussed in [15], so we do not present them here. Besides, in this application they are not relevant since the speed depends on the frequency of measurement that is of the order of minutes, thus being very slow. Instead, it is important the speed at which the agents communicate, considering that they have to exchange several messages and to repeat the negotiation when environment conditions change. The XMPP server used in CoSSMic has been compared with the Tigase implementation and OpenFire. *Tigase* [6] is an open source project to develop an XMPP server implementation in Java. *OpenFire* [13] is an instant messaging and groupchat server that uses XMPP server written in Java and licensed under the Apache License.

The first experiment aims at evaluating the performances of the chosen XMPP servers; to do this, we set up a stress test by using only one host equipped with a 2.67 GHz i5 processor, 4 GB of memory and Windows 7 operating system. On this machine we install the whole testbed and the tested XMPP server, in order not to contaminate the tests with network delays. The testbed is com-



(a) OpenFire vs Tigase



(b) OpenFire Performance

Figure 55: XMPP servers experimental results

posed by a single producer that always accepts offers coming from consumers running within the simulated neighbourhood: we measure the *transaction time* as the timespan within the sending of an energy request and the receipt of the request's outcome. The number of consumers has been increased in each experiment: furthermore, we consider 500 transactions for each consumer in

order to have a mean value quite truthful. The chart in Figure 55 (a) shows the mean time to close a transaction for the two servers. It is clear that OpenFire exhibits better performance than Tigase in terms of messages' management and dispatching. In addition, when the number of transactions exceeded 2000 Tigase crashed and the server connection went down. We have therefore chosen OpenFire as the server to be used for subsequent experiments.

To evaluate the impact of the network in the overlay performances, we repeated the same experiment by using three machines, the first one hosting OpenFire, the second one hosting the consumers and the third one hosting the producer: as depicted in Figure 55 (b), the network delay impacts on the transaction time but the obtained results (around 600 ms with 50 consumers) do not impact too much if related to our application's context. Through this

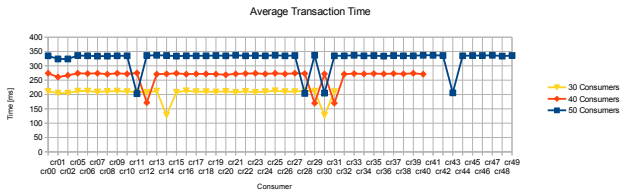


Figure 56: Average transaction time per consumer

experiment it is also possible to evaluate prototype performance in terms of transaction time with the growth of the consumers' number. Figure 56 depicts the average transaction time per consumer for three kinds of tests: 30 consumers, 40 consumers, 50 consumers. As it is deductible, each consumer exhibits the same mean transaction time within the same experiment: thus, the time's growth depends on the number of consumers in the community. In order to highlight the impact of the consumers' number on the performance, in Figure 57 we plotted the average transaction time for each experiment: the time's growth exhibits a linear behaviour with the consumers' growing, with an increase of about 60 ms every 10 consumers, that is symptoms of good performance in terms of scalability with the community's growth. To understand

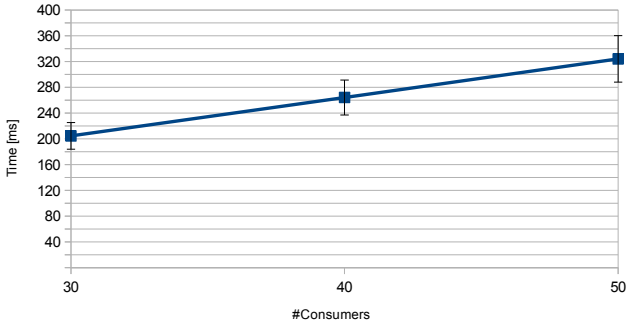
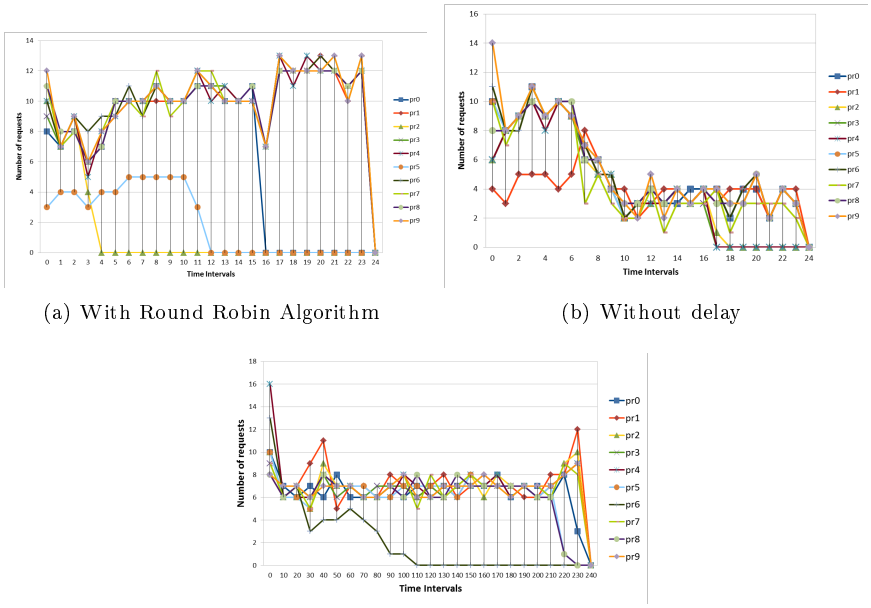


Figure 57: Average transaction time for 30, 40, 50 consumers within the community

the behaviour of the proposed architecture in a real scenario, we create a new testbed with 10 producers and 50 consumers (producers and consumers running on different machines). We performed the tests using three different algorithms, starting from a purely theoretical case and making it gradually more realistic. The first test was carried out using a round robin algorithm. In this test the first producer to be contacted is chosen randomly by each consumer and in succession all the others are contacted. In the second and in the third test, instead, we use an algorithm that exploits the ranking of the producers. In particular each consumer has a vector that stores all the producers and for each a certain reputation. The reputation, which is initially 0 for all, is incremented by 1 each time the producer accepts the proposal. The first time everyone chooses randomly which producer to contact, and thereafter always contact the one that has the highest score, as long as this does not reject the proposal. Once a producer rejects a proposal, it will reject all subsequent requests. We made this choice to simulate the depletion of energy available by a certain producer. The producer may agree to a proposal of the consumer in accordance with a certain probability α. α represents the capacity of a producer to sell energy, thus acting as a *production strength index*: an higher value of α indicates more ability to produce energy and so an higher probability to sell the produced energy. In the third test we introduce a delay between one request and the other whereas in the second one the requests are made in succession. The delay follows a Poisson distribution. We chose this type of distribution because it should be used when an event has a probability to happen very low but the amount of repetitions of the experiment (or the sample size) is so high as to make the event probable to occur.



(a) With Round Robin Algorithm

(b) Without delay

(c) With delay

Figure 58: Number of simultanoeus requests

Figure 58 (a), (b) and (c) show the tests results: the charts represent the number of simultaneous requests per producer when we use, respectively, a Round Robin algorithm, a ranking algorithm without delays among requests and a ranking algorithm with delays. The x-axis scale in Figure 58 (c) is different with respect to the other experiments (Figure 58 (a) and (b)) because the messages are sent with different splits. In the first two cases, the scale is 1 second, in the latter 10 seconds. Besides, the experiments without delays take about 24 seconds, while with delays the duration is about 240 seconds. In all the experiments we can see that there is an initial transient and that the number of requests decreases when the energy providers finishes. In the first test (Figure 58 (a)) the number of requests for each second is quite uniform without particular peaks. When it is used a ranking mechanism (Figure 58 (b) and (c)) a regime is reached when all the producers accept requests. At the end of regime there is a new transitory since all consumers are turning to the same producer who is continuing to provide energy. By comparing the tests represented in Figure 58 (b) and (c) we can see that the maximum number of simultaneous requests when there is a delay decreases. In particular the number of requests in the second test is comparable to those of the first one. In the last test, instead, to have the same number of requests of the first two tests we must consider an interval of 10 seconds.

By Figure 59 it is possible to understand that with the introduction of the delay that follows a Poisson distribution, (thus approaching to a more realistic case), the decrease of the number of simultaneous requests means that response times are lower: for this reason, the performances are compliant with the requirements of scalability and with the dynamics of the system we want to follow.

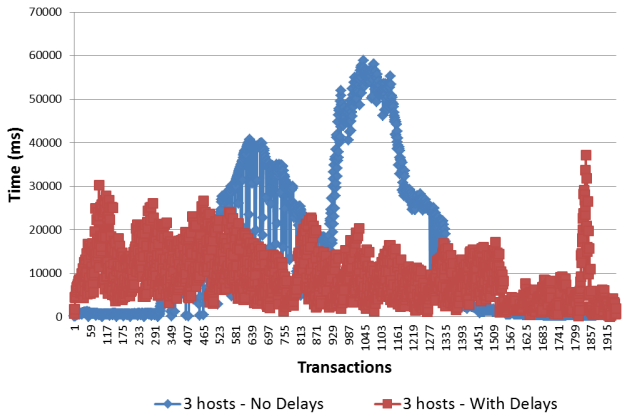


Figure 59: Transactions Time with three hosts

4.3 Evaluation tool for distributed energy utilization

In this section, we present the technologies used to collect the energy data and to evaluate the energy improvements, also integrated with the simulator developed in WP6.

The main objective of the system is to give the opportunity to analyze the energy consumption for each different input configuration. There was therefore the need to save all information relating to the consumption and production of energy of each device. For this purpose it was decided to use a Time Series Database (TSDB). A TSDB is a software system that is optimized for handling time series data. A time series can be seen as an array of numbers indexed by time. In some fields these time series are called profiles, curves, or traces. As TSDB we decide to use InfluxDB. InfluxDB (https://influxdb.com/) is an open-source distributed time series database with no external dependencies where everything is managed as a time series. It provides some standard functions and its data model supports arbitrary event data. It is possible to write in a hash of associated data and count events, uniques, or grouped columns on the fly. It provides native HTTP-API that allows to read and write data from JavaScript. A SQL-like query language designed for working with time series and analytics is provided by InfluxDB. Besides, InfluxDB is designed to scale horizontally.

Grafana provided the technology for chart drawing and visualization. Grafana (http://grafana.org/) is an open source, feature rich metrics dashboard and graph editor for different Time Series DB such as Graphite, InfluxDB and OpenTSDB. It allows to create fast and flexible client side charts with a multitude of options. Grafana is organized in dashboards where it is possible to add and manage several types of graphs. It includes a built in Graphite query parser that takes writing graphite metric expressions to a whole new level. It offers the possibility to connect and interact with InfluxDB through a query editor.

Figure 60: Visualization and evaluation of energy utilization

After the user has configured the neighborhood, the emulation can be started. In essence, the entire configuration is built using the mediator (Emoncms) services and devices are started. At the end of the emulation, the system will bring in InfluxDB all consumption/production time series. From now on, the

user can analyze what happened with that particular configuration. The GUI of our system (Figure 60) follows the tree structure of a CoSSMic neighborhood. In fact, on the left, there are all the micro-grids that constitute the neighborhood. For each node there are the individual user (All in Home configuration) or all users (All in Cloud configuration). Finally for each user there are all devices. The system gives the possibility to analyze consumption/production of each individual device, of user (as sum of all devices), of node (as sum of all users) and of neighborhood (as sum of all nodes). In the screenshot in Figure 60, for example, we can see the GUI. On the left, we can see the Neighborhood with Users and Devices structured as the tree shown in Figure 1. On the right two charts are shown: the power produced by solar panels and the power provided to the households of the neighborhood. Other charts that the system is able to elaborate are: the amount of power provided to the grid, the amount of power received from the grid and the amount of power consumed by devices.

5 Conclusion

This deliverable described the improvements designed and developed in the last phase of the CoSSMic project to maximize the effectiveness of the energy management optimization pursued by the CoSSMic multi agent-system. In particular we distinguished two main contributions: improved learning capabilities and a smarter negotiation model. Learning capabilities allowed agents to better predict the energy profiles of next loads, for different kinds of consuming appliances. They also made easy the usage of the system by the user. In fact we exploited the automatic detection of starting devices to update the user's plan by allowing him to set default parameters. The same learning capabilities, together with machine learning techniques, were validated off-line, using data collected by trials. The results of validation activities include in fact a set of tools for processing of time-series collected almost since the beginning of the project.

Distributed negotation has also been improved to support management of energy storages, which consume and produce at the same time, and to speed up convergence to the maximum self-consumption of the neighborhood. In fact to accomplish the distributed policy each agent is able now to learn which are its best vendors. The vendors' priority are used to increase performance and scalability of the system. A mechanism for the distributed computation of user's reward has also been implemented. That is necessary to compensate the contribution of each user in terms of energy sharing and of flexibility in planning the usage of its own devices. Performance evaluation and scalability have been evaulated here, and in WP6 using simulation.

Finally a functionality that implements the emulation of trials has been developed for testing purpose. It allowed to test the CoSSMic software without the the need to use the infrastructures at trial sites. It allows also for changing the CoSSMic configuration and repeat the tests reproducing the same conditions.

All presented technologies have been integrated in the platform itself and into the simulator developed in WP6.

References

[1] Alexander Semenovich Poznyak and Kaddour Najim. Learning automata with continuous input and changing number of actions. *International Journal of Systems Science*, 27(12):1467– 1472, 1996.

[2] A. Amato, M. Scialdone, and S. Venticinque. An application of learning agents to smart energy domains. In *CEUR Workshop Proceedings*, volume 1382, pages 11–18, 2015. cited By 0.

[3] David G. Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. Springer, 3rd edition, 2008.

[4] G. Horn, S. Venticinque, and A. Amato. Inferring appliance load profiles from measurements. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9258:118–130, 2015. cited By 1.

[5] Imma Curiel. *Cooperative Game Theory and Applications - Cooperative Games Arising from Combinatorial Optimization*, volume 16 of *Theory and Decision Library C: Game Theory, Social Choice, Decision Theory, and Optimization*. Springer, Berlin Heidelberg, 1997.

[6] Tigase Inc. Tigase XMPP server. http://projects.tigase.org/projects/tigase-server.

[7] John G. Saw, Mark C. K. Yang, and Tse Chin Mo. Chebyshev inequality with estimated mean and variance. *The American Statistician*, 38(2):130–132, May 1984.

[8] Klaus Höllig and Jörg Hörner. *Approximation and Modeling with B-Splines*. Society for Industrial and Applied Mathematics, Philadelphia, mar 2014.

[9] Kumpati S. Narendra and Mandayam A. L. Thathachar. *Learning Automata: An Introduction*. Prentice Hall, May 1989.

[10] Lloyd S. Shapley. A value for n-person games. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 2/28 of *Annals of Mathematical Studies*, page 307–317. Princeton University Press, 1953. Paper 17.

[11] M. J. D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. In Susana Gomez and Jean-Pierre Hennart, editors, *Advances in Optimization and Numerical Analysis*, number 275 in Mathematics and Its Applications, pages 51–67. Springer Netherlands, 1994. DOI: 10.1007/978-94-015-8330-5_4.

[12] M. J. D. Powell. Direct search algorithms for optimization calculations. *Acta Numerica*, 7:287–336, January 1998.

[13] Ignite Realtime. OpenFire XMPP server. http://www.igniterealtime.org/projects/openfire/.

[14] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning*, volume 9. MIT Press, Boston, MA, USA, 1998.

[15] Marco Scialdone, Luca Tasquier, Rocco Aversa, and Salvatore Venticinque. Communication overlay for communities of collaborative agents in smart grid domains. *International Journal of Bio-Inspired Computation*, in press.

[16] Luca Tasquier, Marco Scialdone, Rocco Aversa, and Salvatore Venticinque. Agent Based Negotiation of Decentralized Energy Production. In *Intelligent Distributed Computing VIII*, pages 59–67. Springer, 2015.

[17] Xiaotie Deng and Christos H. Papadimitriou. On the complexity of cooperative solution concepts. *Mathematics of Operations Research*, 19(2):257–266, May 1994. ArticleType: research-article / Full publication date: May, 1994 / Copyright © 1994 INFORMS.