Master Thesis

Doble Máster Universitario en Ingeniería Industrial e Ingeniería de la Energía

Integration of a microgrid laboratory into an aggregation platform and analysis of the potential for flexibility

ANNEXES

Author: Maite Etxandi Santolaya
Director: Cristina Corchero García

Call: June 2021

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

ANNEX A. Modbus mapping

ADDRESS	FIELD	DESCRIPTION	SCADA	MODE	TYPE
0	Type of cabinet	The type of the cabinet.	RW_Type	rw	Int16_t
1	Power switch	1 if the cabinet is switching, 0 if not switching.	R_PowerSwitch	r	Int16_t
2	Max active power	Maximum active power the cabinet allows flowing through its DC bus [W].	RW_PMax	rw	Int16_t
3	Min. active power	Minimum active power the cabinet allows flowing through its DC bus [W].	RW_Pmin	rw	Int16_t
4	Max reactive power	The maximum reactive power the cabinet allows flowing through [var].	RW_QMax	rw	Int16_t
5	Min. reactive power	The minimum reactive power the cabinet allows flowing through [var].	RW_QMin	rw	Int16_t
6	Active power setpoint	This variable overwrites the loaded profile [W].	RW_PSetPoint	rw	Int16_t
7	Reactive power setpoint	This variable might overwrite the loaded profile [var]	RW_QSetPoint	rw	Int16_t
10	State	It shows the state of the converter before an error happens.	RW_State	rw	Int16_t
11	CAN State	The SW received from the converter.	R_State	r	Int16_t
12	CAN Error	It shows the error code when the converter fails. Otherwise, 0.	R_Error	r	Int16_t
13	CAN read ID	CAN identification for reading messages.	RW_CANRead ID	rw	Int16_t
14	CAN transmit ID	CAN identification for sending messages.	RW_CANTrans mitID	rw	Int16_t
15	CAN mode	Manual (20) or automatic (10).	RW_ManAuto	rw	Int16_t
16	CAN command	The CW being sent.	RW_CANCmd	rw	Int16_t
17	CAN P	Active power injected [W].	R_P	r	Int16_t
18	CAN Q	Reactive power injected [var].	R_Q	r	Int16_t
19	CAN read 11 iterations	Number of CAN messages received.	R_CANR11Iterations	r	Int16_t
20	CAN Iac	AC current [mA].	R_Iac	r	Int16_t
21	CAN Vac	AC voltage [V].	R_Vac	r	Int16_t
22	CAN Frequency	Frequency of the grid [mHz].	R_Freq	r	Int16_t
23	CAN read 12 iterations	Number of CAN messages received by package 12.	R_CANR12Iterations	r	Int16_t

24	CAN Idc	DC current of the cabinet [mA].	R_Idc	r	Int16_t
25	CAN Vdc	CAN Vdc DC voltage of the cabinet [V].	R_Vdc	r	Int16_t
26	CAN Temperature	Temperature of the cabinet [ºC].	R_Temp	r	Int16_t
27	CAN read 13 iterations	Number of CAN messages received by package 13.	R_CANR13Iterations	r	Int16_t
28	CAN Type	Out of scope.		rw	Int16_t
29	CAN P setpoint	Active power setpoint sent to the emulator [W].		r	Int16_t
30	CAN Q setpoint	Reactive power setpoint sent to the emulator [var].		r	Int16_t
31, 32	Modbus operation read iterations	Num of iterations of the main program every time a Modbus TCP/IP message is read.	R_MBRiterations	r	Int32_t
33, 34	Modbus operation transmit iterations	Num of iterations of the main program every time a Modbus message is written.	R_MBTransmit Iterations	r	Int32_t
35, 36	Modbus operation read write iterations	Num of iterations of the main program every time in case of read-write of Modbus.	R_MBRWIterations	r	Int32_t
40	File num power profile	The file number of the configured profile under the specified folder.	RW_ProfileNum	rw	Int16_t
41	Initial min in profile	The initial minute of the profile from where the switching shall start.	RW_ProfileStartMin	rw	Int16_t
42	Current min in profile	In case of a running emulation it shows the current minute of the profile.	RW_ProfileCurrentMin	r	Int16_t
43,44	Current it. in profile	Current iteration in power profile. Its significance is out of scope.		r	Int32_t
45	Type Set			r	Int16_t
46	Samples per second	Samples per second in the power profile.		r	Int16_t
47	Current P in profile	Value of P in profile at current minute	RW_ProfileCurrentP	r	Int16_t
48	EMS cab			rw	Int16_t
51	SoC	State of Charge in case of a battery or of a battery emulation [%].	R_SOC	r	Int16_t
52	CAN State 2	State of the converter back.		r	Int16_t
53	CAN RX 2	CAN identification for reading messages in the second converter.		rw	Int16_t
54	Initial Iteration	Initial iteration of the power profile.		rw	Int16_t

55	Profile Config	Variable to set new profile.		rw	Int16_t
56	Controllable	Boolean indicating the device's controllability.	RW_Controllable	rw	Int16_t
57	Start switch	Variable to start switching.		rw	Int16_t
58	Start Profile	Variable to start profile.	RW_ProfileStart	rw	Int16_t
59	Experiment reset	Variable to restart profile.		rw	Int16_t
60	Stop profile	Variable to stop profile.		rw	Int16_t

ANNEX B. CAN messages

Status Word (SW)

SW	STATE	DESCRIPTION
1	Standby	Cabinet is powered and ready to precharge
2	Transition towards precharging	Cabinet is in transition between receiving the command for precharge and starting the process
3	Precharging	DC voltage of the cabinet increases slowly to 90-95% of the operating voltage. The current flow is limited to avoid overcurrents
4	PLLing	The internal control system of the cabinet is getting tuned for the start of the emulation (Phase-Locked Loop, PLL)
5	Operating	The cabinet is ready to receive power setpoints
6	Stopped	The cabinet is stopped because of LC command or internal algorithms
7	Emergency stop	Emergency stop can be triggered manually or through a command by the LC or internal algorithms

Command Word (CW)

CW	STATE	DESCRIPTION
1	Emergency stop	Emergency stop is requested. Cabinet reboot is needed
3	Precharge	Command for precharging start
4	Start Switching	Sends the initial and the upcoming power setpoints of P and Q
5	Stop Switching	Sends stop command
F	Ask for an update	Sends request for data update

ANNEX C. SCADA Configuration

1 Classes

Class name	Tag name	Type	Description
Load	P	Real	Current active power (W)
	Q	Boolean	Quality (1=OK 0=Not OK)
	S	Integer	Status (1=ON 0=OFF -1=Error)
HVAC	COP	Real	COP of machine
	Pmax	Real	Maximum power (kW)
	Tstp	Real	Temperature setpoint (ºC)
	P	Real	Current active power (W)
	Ti	Real	Indoor temperature (ºC)
	Q	Boolean	Quality (1=OK 0=Not OK)
	S	Integer	Status (1=ON 0=OFF -1=Error)
Zone	Aw	Real	Area of windows (m2)
	R	Real	Resistance of external wall (ºC/kW)
	C	Real	Capacitance (kWh/ºC)
Battery	P_nom	Real	Nominal power of the battery (W)
	SOC_max	Real	Maximum SOC of the battery
	SOC_min	Real	Minimum SOC of the battery
	SOC	Real	Current SOC of the battery
	P	Real	Current active power of the battery (W)
	S	Integer	Status (1=charge 0=OFF -1=discharge -2=ERROR)
	Q	Boolean	Quality (1=OK 0=Not OK)
	E_nom	Real	Nominal capacity of the battery
	State	String	State of the battery
	eff_charge	Real	Charge efficiency
	eff_discharge	Real	Discharge efficiency
	t_state	Integer	Counts the time that the battery is at the same state
	t1	Integer	Variable defined for the hysteresis

2 Tags

Name	Type	Description
cHVAC	HVAC	Contains data of HVAC
cZone_Properties	Zone	Contains data of zone
cLoad	Load	Contains data of load

cMeter	Load	Contains data of meter
cPV	Load	Contains data of PV
cBattery	Battery	Contains data of Battery
Save_test_config	Integer	To save test configuration parameters (Recipe)
Load_test_config	Integer	Loads test config recipe when toggled
Aggregator_ON	Boolean	True when communicating with aggregator
Demo_on	Boolean	True when demo is running
test_period	String	Either summer or winter
test_type	String	Residential or prosumer test
Demo_Duration	Integer	Hours of demo duration
Demo_StartTime	String	Start time of demo (depending on summer or winter scenario)
Demo_State	Integer	State of demo
Demo_RunningSecs	Integer	Seconds since demo start
Demo_RemainingSecs	Integer	Seconds till demo end
Demo_RunningHours	Real	Hours since demo start
Demo_RemainingHours	Real	Hours till demo end
test_start_date	String	Test start date
Current_DemoDate	String	Demo date (not real time, summer or winter)
Current_DemoTime	String	Demo hour (not real time, summer or winter)
Secs_dif	Integer	Seconds of difference between real and demo time
interval	Integer	Interval of test (15 minute interval since demo start)
temperature_outdoors	Real	Ambient temperature in ºC
solar_radiation	Real	Solar radiation (W/m2)
delta_t_bat	Integer	Delta of time for battery control calculation
batt_control	Integer	Sets to 1 every time battery script needs to run
index_PV	String	Index of PV in csv file
index_load	String	Index of load in csv file
t0-t7	Boolean	Toggles for tasks
t0_r - t7_r	Boolean	Results of tasks (0: no error)
Task_Script	Boolean	When True Task_Script get executed
t_Grid	Integer	Hours that the battery is allowed at SOCmin before charge from grid
t_hist	Integer	Minimum number of minutes that the battery state is maintained
t_Grid_maintain	Integer	Hours to charge battery from Grid
SOC_start	Real	Starting SOC of the sized battery
Setpoint1 and Setpoint2	Real	Used to read setpoints from csv files
DB_save	Boolean	Used to save prosumer data to DB

State_calc	String	Calculated battery state (often different to the real one)
act_battery	String	Holds the activation for the battery
act_openADR	String	Holds the activation for the load in the openADR test
oadr_trigger	Boolean	Trigger for openADR script
event_status	String	Status of current event received from VTN
event_signal_type	String	Signal type of event received from VTN
event_start	String	Start time of event received from VTN
event_duration	String	Duration of event received by VTN
event_opt_response	String	Opt response to event received from VTN
event_intervals	Integer	Number of intervals of event received from VTN

3 Scripts

3.1 Demo

Execution: Demo_on

```
'Variables available only for this group can be declared here.
Dim CurrentTime

'The code configured here is executed while the condition configured in the Execution field is TRUE.
If $test_period="Summer" Then
        $test_start_date =  "07/17/2019"
End If

If $test_period="Winter" Then
        $test_start_date =  "12/10/2019"
End If

'When Demo_on is set to True the timestamp of the test is recorded
If $Demo_State=0 Then
        $Demo_StartTime = $DateTime2Clock($Date,$Time) 'Seconds since 1970 at Demo start (reference)
        $Demo_State = 1
        $Secs_dif=$Demo_StartTime-$DateTime2Clock($test_start_date,"00:00:00")
End If

'State 1 is the state during demo running
If $Demo_State=1 Then
        CurrentTime=$DateTime2Clock($Date,$Time) 'Seconds since 1970
        $Demo_RunningSecs=CurrentTime-$Demo_StartTime 'Seconds since start of test
        $Demo_RunningHours=($Demo_RunningSecs)/(60*60) 'Hours since start of test
        $Demo_RemainingSecs=($Demo_Duration*60*60)-($Demo_RunningSecs) 'Remaining demo seconds
        $Demo_RemainingHours=($Demo_RemainingSecs)/(60*60) 'Remaining demo hours
        $interval=$Trunc($Demo_RunningSecs/(15*60))+1 '15 minute interval tracking
        'Demo Date and time calculation:
        $Current_DemoDate=$ClockGetDate($DateTime2Clock($Date,$Time)-$Secs_dif)
        $Current_DemoTime=$ClockGetTime($DateTime2Clock($Date,$Time)-$Secs_dif)
End If
```

```
'When Demo duration is reached State 2 is set
If $Demo_RunningSecs>$Demo_Duration*60*60 Then
                        $Demo_State=2
End If

'Demo is turned off
If $Demo_State=2 Then
        $Demo_RunningSecs=$Demo_Duration*60*60
        $Demo_RunningHours=$Demo_RunningSecs/(60*60)
        $Demo_RemainingHours=0
        $Demo_RemainingSecs=0
        $Demo_State=0
        $Demo_on=0
End If
```

3.2 Battery control

Execution: Demo_on and test_type="Prosumer" and batt_control=1

```
'Variables available only for this group can be declared here.

Dim State_0, Pb_max, E_SOC_max, E_SOC_min, E_SOC, t_state_0, batt_assigned

'The code configured here is executed while the condition configured in the Execution field is TRUE.
'For easier interpretation we use variables that will hold the tags from the LCs
$cPV.P=$cLC[7].R_P 'PV production (<0 when producing)
$cLoad.P=$cLC[4].R_P 'Load power
'Battery needs to be sized to a smaller one, SOC is calculated using the energy coming in or out of the battery (V*I*t)
$cBattery.SOC=$cBattery.SOC-(($cLC[9].R_Idc*$cLC[9].R_Vdc)/3600)/$cBattery.E_nom

E_SOC_max=$cBattery.E_nom*$cBattery.SOC_max 'Capacity at max SOC
E_SOC_min=$cBattery.E_nom*$cBattery.SOC_min 'Capacity at max SOC
E_SOC=$cBattery.E_nom*$cBattery.SOC 'Capacity at current SOC
State_0=$cBattery.State 'State at previous timestamp
t_state_0=$cBattery.t_state 'Delta of time at previous state

batt_assigned=False 'True when the battery power and state for this timestamp have been defined
$delta_t_bat=1/(60*60) ‘Frequency of calculations
$cBattery.eff_charge=0.98
$cBattery.eff_discharge=0.98

'First, to avoid fast jumps charge-discharge, we maintain the previous state for at least t_hist minutes
'For maintaining charging:
If ((State_0="C" Or State_0="Max") And $cBattery.t1<=$t_hist And $cBattery.t1>0) Then
    'If we reach maximum SOC we neither charge or discharge, but we maintain P=0 to arrive to t_hist minutes
    If $cBattery.SOC>=$cBattery.SOC_max Then
        $cBattery.t1=$cBattery.t1+$delta_t_bat*60
        $cBattery.P=0
        $cBattery.State="Max"
        batt_assigned=True
    'Otherwise we keep charging
    Else
        $cBattery.t1=$cBattery.t1+$delta_t_bat*60
        Pb_max = $Min($cBattery.P_nom, (E_SOC_max-E_SOC)/($delta_t_bat*$cBattery.eff_charge))
```

```
            $cBattery.P=$Min(Pb_max,-($cLoad.P+$cPV.P))
            $cBattery.State="C"
            batt_assigned=True
        End If
End If
'For discharging:
If ((State_0="D" Or State_0="Min") And $cBattery.t1<=$t_hist And $cBattery.t1>0) Then
        'If we reach minimum SOC we neither charge or discharge, but we maintain P=0 to arrive to t_hist minutes
        If $cBattery.SOC<=$cBattery.SOC_min Then
            $cBattery.t1=$cBattery.t1+$delta_t_bat*60
            $cBattery.P=0
            $cBattery.State="Min"
            batt_assigned=True
        'Otherwise we keep discharging
        Else
            $cBattery.t1=$cBattery.t1+$delta_t_bat*60
            Pb_max = -$Min($cBattery.P_nom, (E_SOC-E_SOC_min)/($delta_t_bat*$cBattery.eff_discharge))
        $cBattery.P = $Max(Pb_max,  -($cLoad.P+$cPV.P))
            $cBattery.State="D"
            batt_assigned=True
        End If
End If

'If the state has been maintained for t_hist minutes, then t1 is set to zero and the rest of the conditions are analysed
If $cBattery.t1>$t_hist Then
        $cBattery.t1=0
End If

'State "G" is a particular case where, if the battery has been at min state for over t_Grid hours and we charge the
battery from grid at 2000 W for t_grid_maintain
If (State_0="Min" And t_state_0>$t_Grid*60) Or  (State_0="G" And $cBattery.t_state<$t_Grid_maintain*60 And
$cBattery.SOC<$cBattery.SOC_max) Then
        $cBattery.State="G"
        $cBattery.P = 2000
        batt_assigned=True
End If

'For the rest of the cases (batt_assigned as not been set to True) we analyse the PV production and the load
If batt_assigned=False Then
        'When there is solar surplus if the battery is not at max state we charge it
        If -$cPV.P>=$cLoad.P Then
            If $cBattery.SOC>=$cBattery.SOC_max Then
                $cBattery.P=0
                $cBattery.State="Max"
                batt_assigned=True
            Else
                'We charge (>0) at the min value of: nom power, power to arrive to SOCmax or the PV surplus
                Pb_max=$Min($cBattery.P_nom, (E_SOC_max-E_SOC)/($delta_t_bat*$cBattery.eff_charge))
                $cBattery.P=$Min(Pb_max, -($cLoad.P+$cPV.P))
                If $cBattery.P=0 Then
                    $cBattery.State="S"
                    batt_assigned=True
                Else
                    $cBattery.State="C"
                    batt_assigned=True
```

```
                End If
            End If
        End If
        'When there is not enough solar production we discharge the battery if it is not at min state
        If -$cPV.P<$cLoad.P Then
            'When the battery is over SOC min we discharge (<0) at the min value between nominal power, power to arrive to SOCmin or the required power from load
            If $cBattery.SOC>$cBattery.SOC_min Then
                Pb_max=-$Min($cBattery.P_nom, (E_SOC-E_SOC_min)/($delta_t_bat*$cBattery.eff_discharge))
                $cBattery.P = $Max(Pb_max,-($cLoad.P+$cPV.P))
                $cBattery.State="D"
                batt_assigned=True
            Else
                $cBattery.P=0
                $cBattery.State="Min"
                batt_assigned=True
            End If
        End If
End If
'We set the condition to maintain the discharge/charge when there is a change from C to D or D to C and the previous state lasted less than t_hist
If (State_0="C" And $cBattery.State="D" And t_state_0<$t_hist) Then
    $cBattery.State = "C"
  Pb_max = $Min($cBattery.P_nom, (E_SOC_max-E_SOC)/($delta_t_bat*$cBattery.eff_charge))
  $cBattery.P = $Min(Pb_max, -($cLoad.P+$cPV.P))
  $cBattery.t1 = $cBattery.t_state+$delta_t_bat*60
End If
If (State_0="D" And $cBattery.State="C" And t_state_0<$t_hist) Then
    $cBattery.State = "D"
    Pb_max = -$Min($cBattery.P_nom, (E_SOC-E_SOC_min)/($delta_t_bat*$cBattery.eff_discharge))
  $cBattery.P = $Max(Pb_max, -($cLoad.P+$cPV.P))
  $cBattery.t1 = $cBattery.t_state+$delta_t_bat*60
End If

' When communicating with the aggregator, the setpoint is overwritten (only allowed to discharge/charge in the SOC range defined)
If $Aggregator_ON And $act_battery <> "none" Then
    If ($act_battery<0 And $cBattery.SOC>$cBattery.SOC_min) Or ($act_battery>0 And $cBattery.SOC<$cBattery.SOC_max) Or ($act_battery=0) Then
        $cBattery.P = $act_battery
    End If
End If

$State_calc=$cBattery.State

'The real state of the battery depends on the actual power value of the LC
If (($cBattery.State <> "G") Or ($cBattery.State="G"And $cBattery.P<>2000)) Then
    If $cBattery.SOC<=$cBattery.SOC_min Then
        $cBattery.State="Min"
    ElseIf $cBattery.SOC>=$cBattery.SOC_max Then
        $cBattery.State="Max"
    ElseIf $cLC[9].R_P>1000 Then
        $cBattery.State="C"
    ElseIf $cLC[9].R_P<-1000 Then
        $cBattery.State="D"
```

```
        Else
            $cBattery.State="S"
            $cBattery.t1=0
        End If
End If

'We update t_state variable if the new state is equal to the previous one, otherwise it is set to 0
If $cBattery.State=State_0 Then
        $cBattery.t_state=$cBattery.t_state+$delta_t_bat*60
        Else
            $cBattery.t_state=0
End If

'We calculate the meter value
$cMeter.P=-($cPV.P+$cLoad.P-$cLC[9].R_Idc*$cLC[9].R_Vdc)
If $cMeter.P=0 Then
        $cMeter.S=0
Else
        $cMeter.S=1
End If
$cMeter.Q=1
'Load status and quality (to send to InfluxDB)
If $cLoad.P<>0 Then
        $cLoad. S=1
Else
        $cLoad.S=0
End If
If $cLC[4].R_Error=0 Then
        $cLoad.Q=1
Else
        $cLoad.Q=0
End If

'PV status And quality
If $cPV.P<>0 Then
        $cPV.S=1
Else
        $cPV.S=0
End If
If $cLC[7].R_Error=0 Then
        $cPV.Q=1
Else
        $cPV.Q=0
End If

'Battery status and quality
If ($cLC[9].R_Idc* $cLC[9].R_Vdc)<>0 Then
        $cBattery.S=1
Else
        $cBattery.S=0
End If
If $cLC[9].R_Error=0 Then
        $cBattery.Q=1
Else
        $cBattery.Q=0
```

```
End If

'We send the setpoint to the battery
$cLC[9].RW_PSetPoint=$cBattery.P
'We send the setpoint to PV and load
$index_PV=$FileReadFields("E:\InduSoft Web Studio v8.1 Projects\SCADA_v5\Tools\LAB-SCADA\Profiles\
PV.csv",$index_PV-1,"Setpoint1",1)
$cLC[7].RW_PSetPoint=$Setpoint1
$index_Load=$FileReadFields("E:\InduSoft Web Studio v8.1 Projects\SCADA_v5\Tools\LAB-SCADA\Profiles\
Load.csv",$index_Load-1,"Setpoint2",1)
$cLC[4].RW_PSetPoint=$Setpoint2
'Send data to DB
$DB_save=$Toggle($DB_save)
'We set batt_control to False, it will set to true after one second and the script will run again
$batt_control=0
```

3.3 Tasks

Execution: Task_Script and Demo_on

```
'The code configured here is executed while the condition configured in the Execution field is TRUE.
$Trace("Task Script run at "+$Time)

If $test_type="Residential" Then
    't0 is toggled and then the rest of the tasks (since they are defined as synchronous, they do not overlap)
    'If the functions called by the trigger give an error it gets printed to output screen
    $t0=$Toggle($t0)
    If $t0_r<>0 Then
        $Trace("Task 0 ERROR at " + $Time)
    End If

    $t1=$Toggle($t1)
    If $t1_r<>0 Then
        $Trace("Task 1 ERROR at" + $Time)
    End If

    $t2=$Toggle($t2)
    If $t2_r<>0 Then
        $Trace("Task 2 ERROR at " + $Time)
    End If

    $t3=$Toggle($t3)
    If $t3_r<>0 Then
        $Trace("Task 3 ERROR at " + $Time)
    End If

    $t4=$Toggle($t4)
    If ($t4_r1<>0) Or ($t4_r2<>0)  Or ($t4_r3<>0) Then
        $Trace("Task 4 ERROR at " + $Time)
    End If

    $t5=$Toggle($t5)
    If $t5_r<>0 Then
        $Trace("Task 5 ERROR at " + $Time)
    End If
```

```
        $t6=$Toggle($t6)
        If $t6_r<>0 Then
            $Trace("Task 6 ERROR at " + $Time)
        End If

        $t7=$Toggle($t7)
        If $t7_r<>0 Then
            $Trace("Task 7 ERROR at " + $Time)
        End If
End If

If $test_type="Prosumer" Then
        't0 is toggled and then the rest of the tasks (since they are defined as synchronous, they do not overlap)
        'If the functions called by the trigger give an error it gets printed to output screen
        $t0=$Toggle($t0)
        If $t0_r<>0 Then
            $Trace("Task 0 ERROR at " + $Time)
        End If

        $t1=$Toggle($t1)
        If $t1_r<>0 Then
            $Trace("Task 1 ERROR at" + $Time)
        End If

        $t2=$Toggle($t2)
        If ($t2_r1<>0) Or ($t2_r2<>0)  Or ($t2_r3<>0) Or ($t2_r4<>0) Then
            $Trace("Task 2 ERROR at" + $Time)
        End If

        $t7=$Toggle($t7)
        If $t7_r<>0 Then
            $Trace("Task 7 ERROR at " + $Time)
        End If
End If

'Task_Script is set to 0 until the next interval of 15 minutes
$Task_Script=0
```

3.4 openADR test

Execution: Demo_on and test_type="openADR" and oadr_trigger

```
'Variables available only for this group can be declared here.

Dim Pbase, i
'The code configured here is executed while the condition configured in the Execution field is TRUE.
' Read current load setpoint from csv
$index_Load=$FileReadFields("E:\InduSoft Web Studio v8.1 Projects\SCADA_v5\Tools\LAB-SCADA\Profiles\
load_2hours.csv",$index_Load-1,"Setpoint1",1)
' Base power is the setpoint in the beginning of the event received
If i=0 Then
        Pbase=$Setpoint1
End If
' Load current activations, "none" or LOAD_CONTROL multiplier
```

```
$Recipe("Load:Activations_openADR")
' Not aggregator_ON means that the communication happens directly with openADR
If Not $Aggregator_ON Then
	' When the event written in the openADR recipe is active, we read the value and send it to the LC
	If $event_status="active" Then
		If $event_signal_type="x-loadControlPercentOffset" Then
			$cLC[7].RW_PSetPoint=-$Num($act_openADR)*Pbase
			i=i+1
		Else $Trace("Event received from VTN contains an unknown signal")
		End If
		If $event_intervals>1 Then
			$Trace("Event received contains more than one signal, only able to read the first")
		End If
	Else
		$cLC[7].RW_PSetPoint=-Pbase
		i=0
	End If
End If

'Aggregator_ON means that the communication happens through influxDB
If $Aggregator_ON Then
	If $act_openADR<>"none" Then
		$cLC[7].RW_PSetPoint=-$Num($act_openADR)*Pbase
		i=i+1
	Else
		$cLC[7].RW_PSetPoint=-Pbase
		i=0
	End If
End If

$cLoad.P=-$cLC[7].R_P

'Load status and quallity (to send to influxDB)
If $cLoad.P<>0 Then
	$cLoad.S=1
Else
	$cLoad.S=0
End If
If $cLC[7].R_Error=0 Then
	$cLoad.Q=1
Else
	$cLoad.Q=0
End If
$Recipe("Save:Load")
$DB_save=$Toggle($DB_save)
$oadr_trigger = 0
```

4 Recipes

In all recipes options Save As XML and Unicode are not selected and recipes are saved in Web\Recipes folder as .DAT files. The next table shows the recipe name (in grey) and the tags saved for each recipe in the rest.

Activations_res	Battery	HVAC	Load	Meteo	Meter	PV	Test_config
cHVAC.Tstp	cBattery.P	cHVAC.P	cLoad.P	solar_radiation	cMeter.P	cPV.P	cHVAC.COP
Activations_pro	cBattery.SOC	cHVAC.Ti	cLoad.S	temperature_ outdoors	cMeter.S	cPV.S	cHVAC.Pmax
act_battery	cBattery.S	cHVAC.S	cLoad.Q		cMeter.Q	cPV.Q	cZone_Properties.Aw
Activations_openADR	cBattery.Q	cHVAC.Q					cZone_Properties.R
act_openADR							cZone_Properties.C
event_status							test_period
event_signal_type							test_type
event_start							interval
event_duration							cBattery.SOC_max
event_opt_response							cBattery.SOC_min
event_intervals							cBattery.P_nom
							cBattery.E_nom
							delta_t_bat
							t_hist
							t_Grid
							t_Grid_mantain

5 Schedulers

5.1 Recipes

Event	Trigger	Time	Tag	Expression	Disable
Change	Save_test_config			Recipe(“Save:Test_config”)	Demo_on
Change	Load_test_config			Recipe(“Load:Test_config”)	Demo_on
Change	t0			Recipe("Save:Test_config")	Not Demo_on and test_type <>”Residential”
Change	t1		t1_r	Recipe("Load:Activations_res")	Not Aggregator_ON or Not Demo_on or test_type<>"Residential"
Change	t2			$Exec("E:\InduSoft Web Studio	Not Demo_on or

				v8.1 Projects\SCADA_v5\Tools\ LAB-SCADA\Profiles\ run_read_csv.bat",0,1,"t2_r")	test_type<>"Residential"
Change	t4		t4_r1	Recipe("Load:Meteo")	Not Demo_on or test_type<>"Residential"
Change	t4		t4_r2	Recipe("Load:HVAC")	Not Demo_on or test_type<>"Residential"
Change	t4		t4_r3	Recipe("Load:Load")	Not Demo_on or test_type<>"Residential"
Change	t5		cMeter.P	cLoad.P+Abs(cHVAC.P)	Not Demo_on or test_type<>"Residential"
Change	t5		cMeter.S	If(cMeter.P<>0,1,0)	Not Demo_on or test_type<>"Residential"
Change	t5		cMeter.Q	1	Not Demo_on or test_type<>"Residential"
Change	t6		t6_r	Recipe("Save:Meter")	Not Demo_on or test_type<>"Residential"
Change	t1		t1_r	Recipe("Load:Activations_pro")	Not Aggregator_ON or Not Demo_on or test_type<>"Prosumer"
Change	t2		t2_r1	Recipe("Save:Load")	Not Demo_on or test_type<>"Prosumer"
Change	t2		t2_r2	Recipe("Save:PV")	Not Demo_on or test_type<>"Prosumer"
Change	t2		t2_r3	Recipe("Save:Battery")	Not Demo_on or test_type<>"Prosumer"
Change	t2		t2_r4	Recipe("Save:Meter")	Not Demo_on or test_type<>"Prosumer"
Clock		00:15:00	Task_Script	1	Not Demo_on

5.2 InfluxDB

Event	Trigger	Expression	Disable
Change	t7	$Exec("E:\InduSoft Web Studio v8.1 Projects\SCADA_v5\ Tools\LAB-SCADA\ run_InfluxDB_ReadDAT_WriteDB.bat",0,1,"t7_r")	Not Aggregator_ON or not Demo_on or test_type="openADR"
Change	t0	$Exec("E:\InduSoft Web Studio v8.1 Projects\SCADA_v5\ Tools\LAB-SCADA\run_InfluxDB_Query.bat",0,1,"t0_r")	Not Aggregator_ON or not Demo_on or

			test_type="openADR"
Change	minute	$Exec("E:\InduSoft Web Studio v8.1 Projects\SCADA_v5\Tools\LAB-SCADA\run_InfluxDB_ReadDAT_WriteDB.bat",0,1)	Not Aggregator_ON or not Demo_on or test_type<>"openADR"
Change	minute	$Exec("E:\InduSoft Web Studio v8.1 Projects\SCADA_v5\Tools\LAB-SCADA\run_InfluxDB_Query.bat",0,1)	Not Aggregator_ON or not Demo_on or test_type<>"openADR"

5.3 HVAC model

Event	Trigger	Expression	Disable
Change	t3	$Exec("E:\InduSoft Web Studio v8.1 Projects\SCADA_v5\Tools\LAB-SCADA\HVAC_model\HVAC_model.bat",0,1,"t3_r")	Not Demo_on or test_type<>"Residential"

5.4 Battery control

Event	Trigger	Tag	Expression	Disable
Change	second	batt_control	1	test_type<>"Prosumer" or not Demo_on
Change	second	cBattery.SOC	SOC_start	test_type<>"Prosumer" or Demo_on

5.5 OpenADR

6 Drivers

Residential	Prosumer	openADR
History Format: Database (smartlab_TFM_res) Save on Trigger: t7 Tag names: temperature_outdoors solar_radiation cHVAC.P cHVAC.Ti	History Format: Database (smartlab_TFM_pro) Save on Trigger: DB_save Tag names: cMeter.P cPV.P cBattery.SOC cBattery.P cLoad.P	History Format: Database (smartlab_TFM_openADR) Save on Trigger: DB_save Tag names: cLoad.P cLC[7].RW_PSetPoint Setpoint1 act_openADR event_status

cHVAC.Tstp	cBattery.State	event_signal_type
cLoad.P	Current_DemoDate	event_start
cMeter.P	Current_DemoTime	event_duration
test_type	cLC[9].RW_Idc	event_opt_response
test_period	cLC[9].RW_Vdc	event_intervals
Current_DemoDate	cLC[9].RW_PSetPoint	Aggregator_ON
Current_DemoTime	Aggregator_ON	
Aggregator_ON		

7 Drivers

7.1 LC4 – RB6 (LC7 – RB12 is equivalent)

Read Trigger: readtrigger

Write Trigger: writetrigger

Read Status: cLC[4].MB_int16_ReadStatus

Write Status: cLC[4].MB_int16_WriteStatus

Station: {cLC[4].StationID}

Enable Read when Idle: cLC[4].Enable

Enable Write on Tag Change: cLC[4].Enable

Read Completed: cLC[4].MB_int16_ReadCompleted

Write Completed: cLC[4].MB_int16_WriteCompleted

Header: 4X:0

Tag Name	Address	Div
cLC[4].RW_Type	1	
cLC[4].R_PowerSwitch	2	
cLC[4].RW_Pmax	3	
cLC[4].RW_Pmin	4	
cLC[4].RW_Qmax	5	
cLC[4].RW_Qmin	6	
cLC[4].RW_PSetPoint	7	
cLC[4].RW_QSetPoint	8	
cLC[4].RW_State	11	
cLC[4].R_State	12	
cLC[4].R_Error	13	

cLC[4].RW_CANReadID	14	
cLC[4].RW_CANTransmitID	15	
cLC[4].RW_ManAuto	16	
cLC[4].RW_CANCmd	17	
cLC[4].R_P	18	1.0
cLC[4].R_Q	19	
cLC[4].R_Iac	21	1000.0
cLC[4].R_Vac	22	
cLC[4].R_Freq	23	100.0
cLC[4].R_Idc	25	10.0
cLC[4].R_Vdc	26	
cLC[4].R_Temp	27	
cLC[4].RW_ProfileNum	41	
cLC[4].R_ProfileStartMin	42	
cLC[4].R_ProfileCurrentMin	43	
cLC[4].RW_SamplesxSecond	47	
cLC[4].RW_ProfileCurrentP	48	
cLC[4].R_LCID	49	
cLC[4].R_SOC	52	
cLC[4].R_ProfileActualized	57	
cLC[4].RW_Controllable	58	
cLC[4].RW_Start	59	
cLC[4].RW_Stop	60	
cLC[4].RW_Precharge	61	
cLC[4].RW_StartCAN	62	
cLC[4].RW_Qtype	63	
cLC[4].RW_Vmin	64	
cLC[4].RW_Vmax	65	
cLC[4].R_Vnom	66	
cLC[4].RW_d	67	
cLC[4].R_Snom	68	
cLC[4].RW_QSetPoint	31	

7.2 LC9 – 2LBG – Holding

Read Trigger: readtrigger

Enable Read when Idle: cLC[9].Enable

Write Trigger: writetrigger

Enable Write on Tag Change: cLC[9].Enable

Read Status: cLC[9].MB_int16_ReadStatus

Read Completed: cLC[9].MB_int16_ReadCompleted

Write Status: cLC[9].MB_int16_WriteStatus

Write Completed: cLC[9].MB_int16_WriteCompleted

Station: {cLC[9].StationID}

Header: 4X:0

Tag Name	Address	Div
cLC[9].RW_PSetPoint	5	-0.01
cLC[9].RW_QSetPoint	6	0.01
Bat_MaxV	8	10.0
Bat_MinV	9	10.0
Bat_MaxSOC	10	10.0
Bat_MinSOC	11	10.0
Bat_StartStop	1	
cLC[9].RW_Start		
cLC[9].RW_Pmax	3	
cLC[9].RW_Pmin	4	
Bat_RW_Ramp	7	

7.3 LC9 – 2LBG – Input

Read Trigger: readtrigger

Enable Read when Idle: cLC[9].Enable

Write Trigger: writetrigger

Enable Write on Tag Change: cLC[9].Enable

Read Status: cLC[9].MB_int16_ReadStatus

Read Completed: cLC[9].MB_int16_ReadCompleted

Write Status: cLC[9].MB_int16_WriteStatus

Write Completed: cLC[9].MB_int16_WriteCompleted

Station: {cLC[9].StationID}

Header: 3X:0

Tag Name	Address	Div	Add
cLC[9].R_P	36	-0.01	
cLC[9].R_Q	37	-0.01	
cLC[9].R_Idc	38	10.0	
cLC[9].R_Vdc	39	10.0	
Bat_SOC	48	10.0	
Bat_MaxBatChg	45	0.01	
Bat_MaxBatDis	46	0.01	
cLC[9].R_State	14		-5.0

8 Event logger

	Tag Name	Dead band	Message
1	cHVAC.Tstp	0	New temperature setpoint defined for {Current_DemoDate}
2	act_battery	0	New setpoint received for the battery
3	act_openADR	0	New setpoint received for the load
4	event_status	0	Event starting at {event_start} is {event_status}. {event_signal_type} is {act_openADR} for {event_duration}

9 Screens

9.1 Demo

Size: 1600x780
Location: 120x0

Enable Aggregator Pushbutton

Type: maintained
Tag/Expression: Aggregator_ON
State: Normally Open
Style: Rectangle 3D
Open: Caption OFF, colour grey
Closed: Caption ON, colour green

Start/Stop Pushbutton

Type: maintained
Tag/Expression: Demo_on
State: Normally Open
Style: Rectangle 3D
Open: Caption START, colour green
Closed: Caption STOP, colour red

Period for test ComboBox

Label: test_period
Data sources: static labels

Test type ComboBox

Label: test_type
Data sources: static labels

Labels: Summer, Winter

Labels: Residential, Prosumer, openADR

Logging Alarm/Event Object

Type: Event
Columns: Message, event time, previous, value

Curves Trend Object

Data Sources: tags
Depending on the test, different tags were included

Save Test Parameters Button

Caption: save
Command Type: Toggle Tag
On Down: Save_test_config

Load Test Parameters Button

Caption: save
Command Type: Toggle Tag
On Down: Load_test_config

HVAC Button

Caption: HVAC
Command Type: VBScript
On Down: $Open("Demo_HVAC_Selection")

Building Button

Images: Resources\home.png
Command Type: VBScript
On Down: $Open("Demo_Building_Properties")

Battery Button

Caption: Battery
Command Type: VBScript
On Down: $Open("Demo_Battery_Selection")

In addition, visibility animations have been included so that the corresponding test elements appear after selecting test_type. Also, when Demo_on, information about the test run time appears in the screen.

9.2 Demo_Building_Properties

Size: 800X400
Location: 200x200
Titlebar: Building property selection
System Menu: style Dialogue, border Thin

9.3 Demo_HVAC_Selection

Size: 500x250
Location: 315x500
Titlebar: HVAC Definition
System Menu: style Dialogue, border Thin

9.4 Demo_Battery_Selection

Size: 500x475
Location: 250x450
Titlebar: Demo Battery Selection
System Menu: style Dialogue, border Thin

ANNEX D. HVAC Model

1. Script HVAC_model

```
"""
RC Model for HVAC (simple - 1 thermal zone). This script reads configuration
parameters and inputs. And outputs power and indoor temp after a deltat

Ci [kWh/ºC]: Capacitance, represents the inertia of the building
Ra [ºC/kW]: Resistance, represents the thermal transmission of a wall
Ps [kW]: Heating source, due to solar gains
P [kW]: Heating source, due to the HVAC
Aw [m2]: Equivalent area of the windows
COP [-]: Coefficient of performance
Ti [ºC]: interior temperature
Tstp [ºC]: temperature setpoint
Ta [ºC]: ambient temperature
Pmax [kW]: maximum power of HVAC

"""
import random

# Read configuration parameters from recipe (building and HVAC parameters)
dir = 'E:\\InduSoft Web Studio v8.1 Projects\\SCADA_v5\\Web\\Recipes'

with open(dir + '\\TEST_CONFIG.DAT', 'r') as f:
    COP = float(f.readline())
    Pmax = float(f.readline())
    Aw = float(f.readline())
    Ra = float(f.readline())
    Ci = float(f.readline())
    test = f.readline().strip()

with open(dir + '\\ACTIVATIONS_RES.DAT', 'r') as f:
    Tstp = float(f.readline())

# Other inputs (real time)
with open(dir + '\\HVAC.DAT', 'r') as f:
    P_0 = float(f.readline())
    P_0 = P_0 / 1000
    Ti_0 = float(f.readline())

with open(dir + '\\METEO.DAT', 'r') as f:
    Ps_0 = float(f.readline())
    Ps_0 = Ps_0 / 1000
```

```
    Ta_0 = float(f.readline())

# Delta of time for data is 15 minutes
deltat = 15/60

# HVAC Power calculation
temp_dif = abs(Ti_0-Tstp)
temp_dif_max = 1
temp_dif_control = 0.3

# Check if the machine was ON or OFF in the beginning of the period
if P_0 == 0:
    C = 0
else:
    C = 1
# For cooling:
if test == 'Summer':
    Pmax =  -Pmax
    Pmin = 0.15 * Pmax
    # When the machine was OFF in the begging of the period
    if C == 0:
        # Cooling only when temp_dif is higher than temp_dif_control
        if Ti_0 > (Tstp + temp_dif_control):
            C  = 1
            P = max(Pmax, Pmax*((Ti_0-(Tstp-temp_dif_control))/temp_dif_max))
            if P > Pmin:
                P = Pmin
        else:
            P = 0
    # When the machine was ON in the begging of the period
    elif C ==1:
          # We continue cooling until the temperature is temp_dif_control below
setpoint
        if Ti_0 > (Tstp - temp_dif_control):
           P = max(Pmax, Pmax *((Ti_0-(Tstp-temp_dif_control))/temp_dif_max))
            if P > Pmin:
                P = Pmin
        else:
            P = 0
            C  = 0
# For heating:
elif test == 'Winter':
    Pmin = 0.15 * Pmax
    # When the machine was OFF in the beginning of the period
```

```
    if C == 0:
        # Heating only when temp_dif is higher than temp_dif_control
        if Ti_0 < (Tstp - temp_dif_control):
            C =  1
           P=min(Pmax, Pmax *((Tstp + temp_dif_control-Ti_0)/temp_dif_max))
            if P < Pmin:
                P = Pmin
        else:
            P = 0
    # When the machine was ON in the beginning of the period
    elif C ==1:
          # We continue heating until the temperature is temp_dif_control above
setpoint
        if Ti_0 < (Tstp + temp_dif_control):
           P = min(Pmax, Pmax*((Tstp +temp_dif_control-Ti_0)/temp_dif_max))
            if P < Pmin:
                P = Pmin
        else:
            P = 0
            C  = 0
if abs(P) > 0:
    status = 1
else:
    status = 0
quality = 1

# RC Model definition --> To calculate indoor temperature.
ran = random.uniform(-0.1, 0.1)
Ti   =   Ti_0   +   ((1/(Ra*Ci))*(Ta_0-Ti_0)*deltat)   +   (Aw/Ci*Ps_0*deltat)   +
(COP/Ci*P)*deltat + ran

P = P * 1000
# Write HVAC data to recipe (to send to InfluxDB)
with open(dir + '\\HVAC.DAT', 'w') as f:
    f.writelines(str(P) + '\n')
    f.writelines(str(Ti) + '\n')
    f.writelines(str(status) + '\n')
    f.writelines(str(quality) + '\n')
```

ANNEX E. InfluxDB Scripts

1. Script InfluxDB_ReadDAT_WriteDB.py

```
"""
This script is used to read data from SCADA recipes and send it to InfluxDB
"""
from influxdb import InfluxDBClient
import datetime

client = InfluxDBClient(host="smartlab.ddns.net", port=8087,
username="smartlab",password="case", database="caseDB")

# Recipe folder definition
folder_name = "E:\\InduSoft Web Studio v8.1 Projects\\SCADA_v5\\Web\\Recipes"

# Scenario to be tested, according to TEST_CONFIG recipe
with open(folder_name + '\\TEST_CONFIG.DAT', 'r') as f:
    data = f.readlines()
    scenario = data[6].strip()

# Asset definition according to test to be developed
if scenario == "Residential":
    list_assets=["meter","hvac","load"]
    list_file_names=["\\METER.DAT","\\HVAC.DAT","\\LOAD.DAT"]
elif scenario == "Prosumer":
    list_assets=["PV","battery","load"]
    list_file_names=["\\PV.DAT","\\BATTERY.DAT","\\LOAD.DAT"]
elif scenario == "openADR":
    list_assets=["load"]
    list_file_names=["\\LOAD.DAT"]

# Current time
time = datetime.datetime.now()
time=time.strftime("%Y-%m-%dT%H:%M:%SZ")

# Json definition and sending data to InfluxDB
for asset in list_assets:
    data_file=open(folder_name+list_file_names[list_assets.index(asset)],"r")
    data_array=data_file.readlines()
    if asset=="meter" or asset == "PV" or asset=="load":
        json_body= [{
                    "measurement": "measures",
                    "tags": {
                            "Asset": asset,
```

```
                        "ID": "smartlab",
                        "ID_prosumer": "IREC"
                        },
                    "time": time,
                    "fields": {
                                "power": float(data_array[0]),
                                "status": float(data_array[1]),
                                "quality": float(data_array[2])
                                }
                    }]
    elif asset=="battery":
        json_body = [{
                    "measurement": "measures",
                    "tags": {
                            "Asset": asset,
                            "ID": "smartlab",
                            "ID_prosumer": "IREC"
                            },
                    "time": time,
                    "fields": {
                                "power": float(data_array[0]),
                                "SOC": float(data_array[1]),
                                "status": float(data_array[2]),
                                "quality": float(data_array[3])
                                }
                    }]
    elif asset=="hvac":
        json_body = [{
                    "measurement": "measures",
                    "tags": {
                            "Asset": asset,
                            "ID": "smartlab",
                            "ID_prosumer": "IREC"
                            },
                    "time": time,
                    "fields": {
                                "power": float(data_array[0]),
                                "temperature": float(data_array[1]),
                                "status": float(data_array[2]),
                                "quality": float(data_array[3])
                                }
                    }]
    client.write_points(json_body)
    data_file.close()
```

2. Script InfluxDB_Query.py

```
"""
This script is used to query the last activation command to InfluxDB Database
"""
from influxdb import InfluxDBClient
client = InfluxDBClient(host="smartlab.ddns.net", port=8087,
username="smartlab",password="case", database="caseDB")
dir = 'E:\\InduSoft Web Studio v8.1 Projects\\SCADA_v5\\Web\\Recipes'
# Scenario to be tested, according to TEST_CONFIG recipe
with open(dir + '\\TEST_CONFIG.DAT', 'r') as f:
    data = f.readlines()
    scenario = data[6].strip()
# Asset definition according to test to be developed
if scenario == "Residential":
    list_assets=["hvac"]
    recipe = '\\ACTIVATIONS_RES.DAT'
elif scenario == "Prosumer":
    list_assets=["battery"]
    recipe = '\\ACTIVATIONS_PRO.DAT'
elif scenario == "openADR":
    list_assets=["load"]
    recipe = '\\ACTIVATIONS_PRO.DAT'
# Reading activation commands for each asset and saving to recipe
with open(dir + recipe, 'w') as f:
    if "hvac" in list_assets:
        query_statement= "select last(activations) from caseDB.autogen.commands
where Asset='hvac' and ID_prosumer='IREC' and ID='smartlab'"
       results=client.query(query_statement)
       for command in results.get_points(measurement='commands'):
            act = command['last']
       f.write(str(act) + '\n')
    if "battery" in list_assets:
         query_statement= "select last(activation) from caseDB.autogen.commands
where Asset='battery' and ID_prosumer='IREC' and ID='smartlab'"
        results=client.query(query_statement)
        for command in results.get_points(measurement='commands'):
            act = command['last']
        if act == 1:
            query_statement= "select last(power) from caseDB.autogen.commands
           where Asset='battery' and ID_prosumer='IREC' and ID='smartlab'"
            results=client.query(query_statement)
            for command in results.get_points(measurement='commands'):
                power = str(command['last'])
           if act == 0:
```

```
                power = "none"
        f.write(power + '\n')

if "load" in list_assets:
         query_statement= "select last(activation) from caseDB.autogen.commands
where Asset='Load' and ID_prosumer='IREC' and ID='smartlab'"
        results=client.query(query_statement)
        for command in results.get_points(measurement='commands'):
            act = command['last']
        if act == 1:
            query_statement= "select last(power) from caseDB.autogen.commands
           where Asset='Load' and ID_prosumer='IREC' and ID='smartlab'"
            results=client.query(query_statement)
            for command in results.get_points(measurement='commands'):
                power = str(command['last'])
           if act == 0:
                power = "none"
        f.write(power + '\n')
```

ANNEX F. DEMO simulations

The lines of code used for graphing and saving results have not been included to reduce the length of the document. In addition, the HVAC model and battery control functions are equivalent to the ones presented in Annexes D and C (but using Python language) and so they are not included.

1. Residential

```
# This script is used to develop the residential scenario simulation
from HVAC_model import HVAC_model
from read_csv import update_recipe
import datetime as dt
import pandas as pd

aggregator_on = True # If True, HVAC works at lower setpoint during peak hours

# Test configuration parameters
with open('TEST_CONFIG.DAT', 'r') as f:
    COP = float(f.readline())
    Pmax = float(f.readline())
    Pmin = Pmax * 0.15
    Aw = float(f.readline())
    Ra = float(f.readline())
    Ci = float(f.readline())
    test = f.readline().strip()

with open('ACTIVATIONS_RES.DAT', 'r') as f:
    Tstp_config = float(f.readline())

# HVAC initial values
with open('HVAC.DAT', 'r') as f:
    P_0 = float(f.readline())
    Ti_0 = float(f.readline())

# Test date definition. Setpoints can be defined for night and 2 events
if test == "Summer":
    start_date = dt.datetime(2019,7,17,0,0,0)
    Tstp_night = Tstp_config + 2
    Tstp_peak1 = Tstp_config + 1
    Tstp_peak2 = Tstp_config + 2
    day1 = 17

elif test == "Winter":
    start_date = dt.datetime(2019,12,10,0,0,0)
```

```
    Tstp_night = Tstp_config - 3
    Tstp_peak1 = Tstp_config - 1
    Tstp_peak2 = Tstp_config - 2
    day1 = 10

day2 = day1 + 1
day3 = day1 + 2

time = start_date
day = 1
len = 3*24/4
# For every 15 minute during the three days of the test:
for i in range(len):
    # Recipes are updated from csv data
    power, radiation, temperature = update_recipe(test, i)
    # Event periods can be defined here
    night = (time.hour >= 24 or time.hour <=7)
    peak_1 = (time.hour == 8 and time.minute < 30 and time.day == day1)
    peak_2 = (time.hour == 18 and time.minute >= 30 and time.day == day2)
    peak_3 = (time.hour == 12 and time.day == day3)
    if night:
        Tstp = Tstp_night
    elif aggregator_on and (peak_1 or peak_2):
        Tstp = Tstp_peak1
    elif aggregator_on and peak_3:
        Tstp = Tstp_peak2
    else:
        Tstp = Tstp_config
    # HVAC_model.py is executed
    HVAC_power, Ti = HVAC_model(test, Tstp, COP, Pmax, Ra, Ci, Aw)
    meter = power + abs(HVAC_power)
    time = time + dt.timedelta(minutes=15)
    day = day + 15/(60*24)
```

2. Prosumer

```
# This script is used to test the simulate the prosumer.
import matplotlib.pyplot as plt
import datetime as dt
import pandas as pd
# Battery parameters
SOC_max = 0.8 # Max SOC can be defined
SOC_min = 0.2 # Min SOC can be defined
eff_charge = 0.98 # Charge efficiency
eff_discharge = 0.98 # Discharge efficiency
```

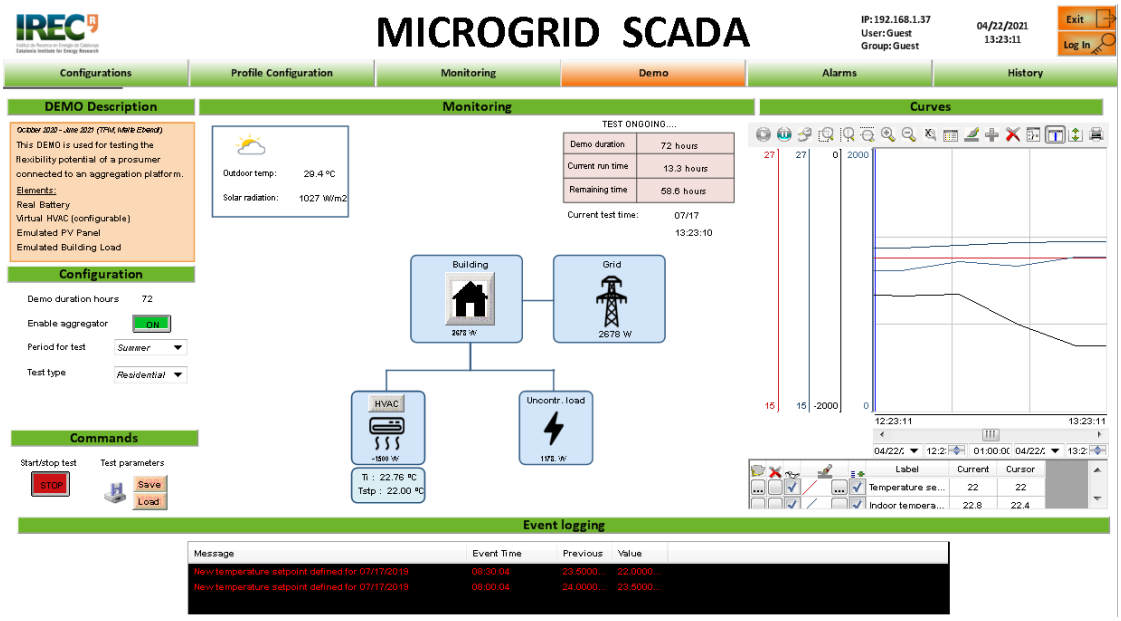
```
delta_t = 1/(60*60) # Frecuency of control execution
E = 5000 # Usable battery capacity can be defined
E_SOC_max = E*SOC_max
E_SOC_min = E*SOC_min
Pbat_nom = 10000 # Nominal power can be defined
t1 = 0
# To simulate that the battery does not follow setpoints below 1500W
restricted = True
# To simulate events from the aggregator
aggregator_on = True
"""
Battery states
D: discharging
C: charging
G: battery charging from grid
Min: battery at SOC min
Max: battery at SOC max
S: between min and max SOC
"""
# We load the csv profiles
with open('PV.csv', 'r') as f:
    data_pv = f.readlines()
with open('load.csv', 'r') as f:
    data_load = f.readlines()

num = 3*24*60*60 # Test duration is 3 days
SOC = 0.6 # Initial SOC
state_0 = "S"  # Initial state
date = dt.datetime(2019,7,17,0,0,0,0)
t = 0
for i in range(num):
    # We read the PV and load values from csvs
    Ppv = float(data_pv[i].strip())
    Pload = float(data_load[i].strip())
    SOC_0=SOC
    # We execute the battery control
    SOC, Pgrid, Pbat, state = battery_control(SOC, Ppv, Pload, state_0, t)
    # Event timings can be defined here
     event1 = (((date.hour == 18 and date.minute >= 30) or date.hour == 19 or
date.hour == 20 or date.hour == 21) and date.day == 17)
    event2 = ((date.hour == 14 or date.hour == 15) and date.day == 18)
    # For the aggregator case, we can overwrite the setpoints
    if aggregator_on:
        # For event 1 we impose Pbat=0
```

```
        if event1 or event2:
            Pbat = 0
        # For event 2 we impose Pbat=-2000
        if event2:
            if SOC>SOC_min:
                Pbat = -2000
            else:
                Pbat = 0
    # Setpoints below 1500 will not be followed
    if abs(Pbat)<1500 and restricted:
        Pbat=0
        SOC=SOC_0
        t1=0
    # State definition
    if SOC <= SOC_min:
        state = "Min"
    elif SOC >= SOC_max:
        state = "Max"
    elif state=="G" and Pbat==2000:
        state = "G"
    elif Pbat > 0:
        state ="C"
    elif Pbat < 0:
        state = "D"
    else:
        state = "S"
    # Grid calculation
    Pgrid = -(Ppv + Pbat + Pload)
    # Consecutive minutes at same state
    if state == state_0:
        t = t + (delta_t*60)
    else:
        t = 0
    state_0 = state
    date = date + dt.timedelta(hours = delta_t)
```

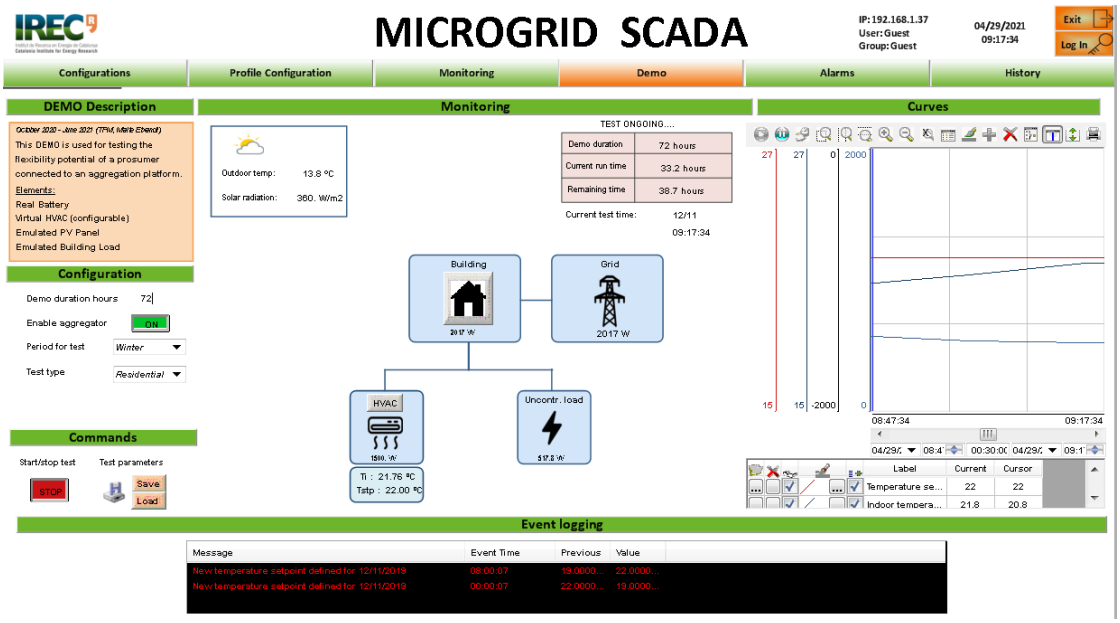
ANNEX G. Scenario screenshots

1. Residential

Aggregator summer case (run from 22/04/2021 to 24/04/2021)

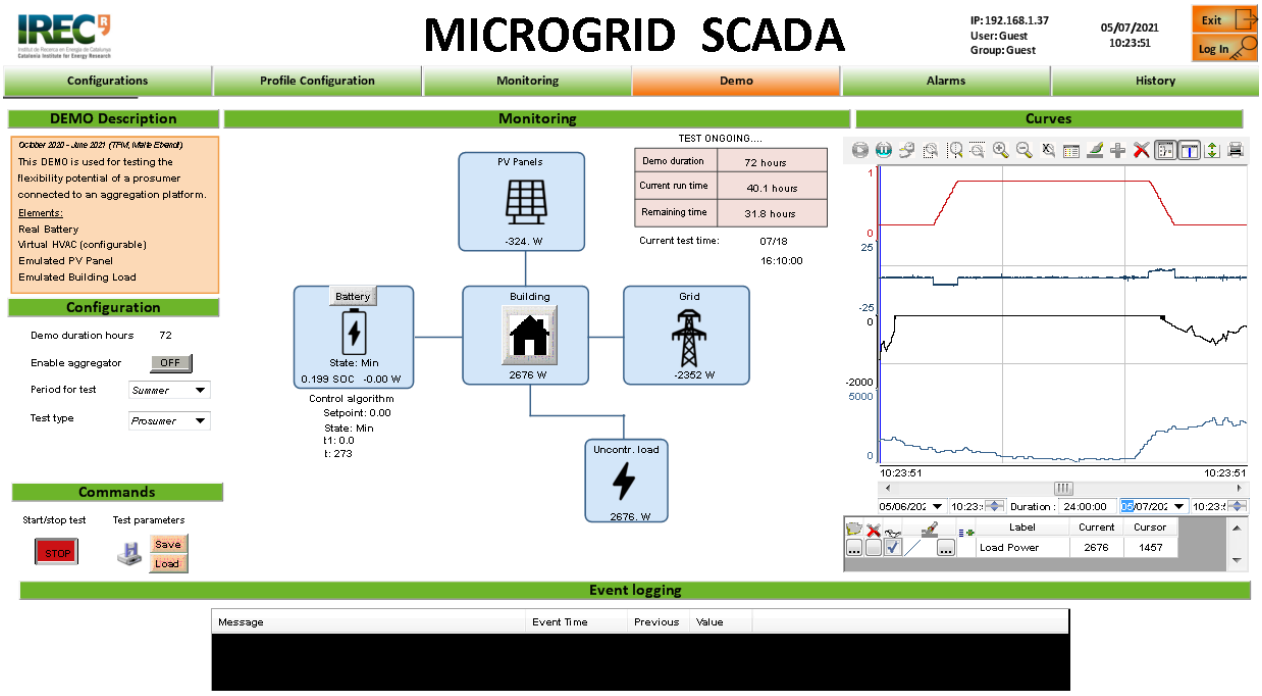


Aggregator winter case (run from 28/04/2021 to 30/04/2021)

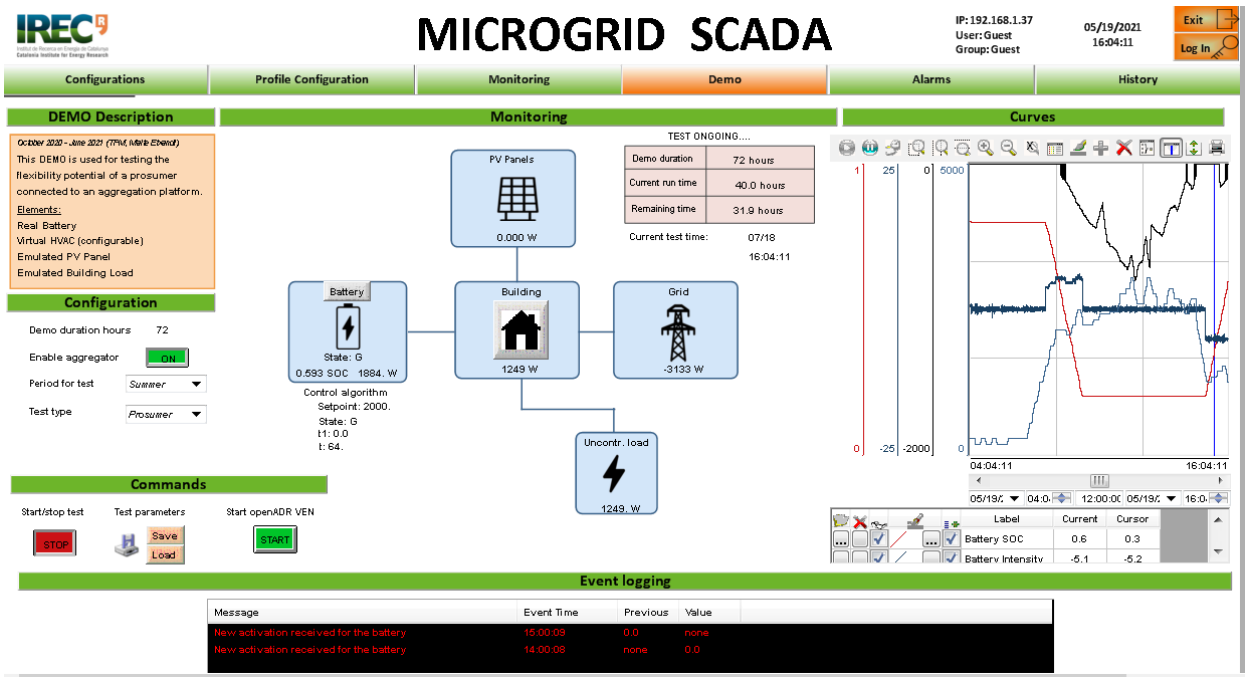


2. Prosumer

Base case (run from 06/05/2021 to 08/05/2021)

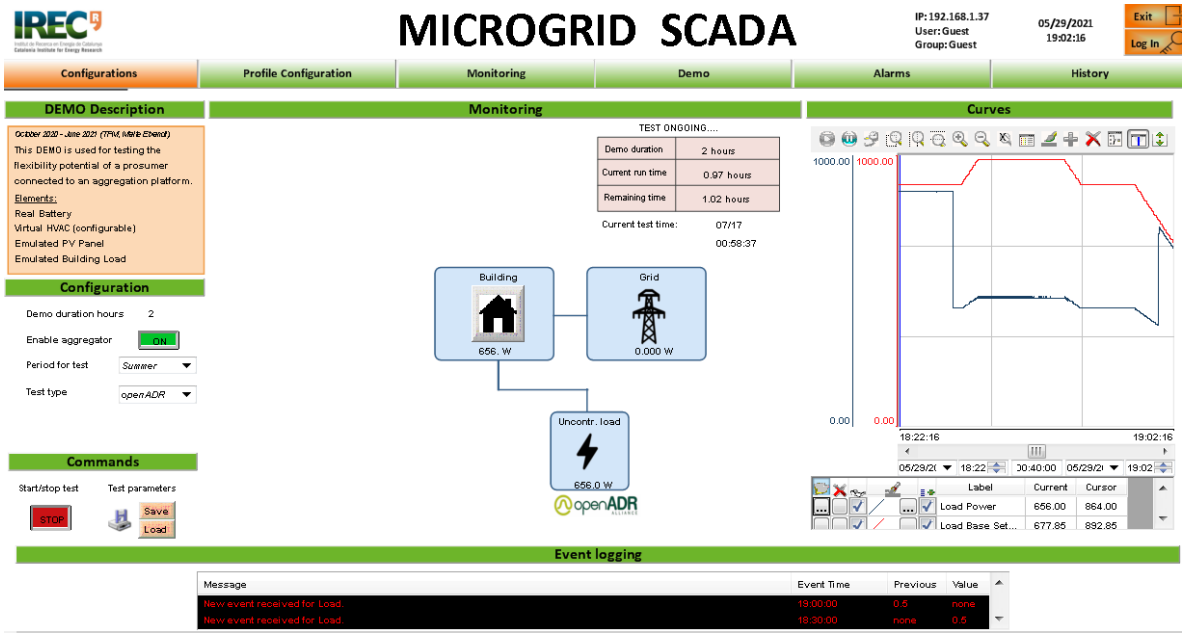


Aggregator case (run from 18/05/2021 to 20/05/2021)

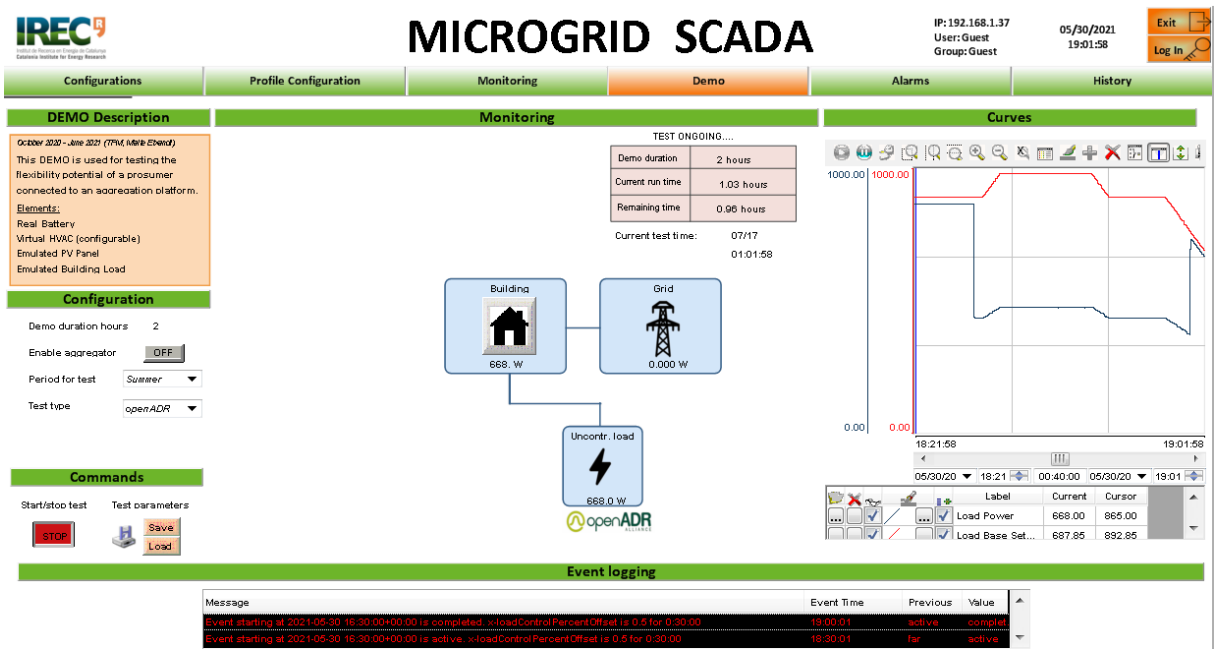


3. OpenADR

Aggregator API case (run during 18:00 to 20:00 on 29/05/2021)



openADR case (run during 18:00 to 20:00 on 30/05/2021)



ANNEX H. OpenADR test scripts and logging

1. test_VTN Python Script

```
"""
This script is used for the openADR implementation of a VTN (for testing phase)
"""

import asyncio
from datetime import timedelta, datetime, timezone
from openleadr import OpenADRServer
from openleadr.objects import Interval, Target
from functools import partial
import logging
import openleadr
from random import randint
from aux_functions import event_creation, print_event_object

# Name of the VTN
vtn_id = 'test_VTN'

# Creation of the VTN
server = OpenADRServer(vtn_id=vtn_id)
# Optionally certificates can be defined for the server

DR_program=''
ven_resources = []
event_count = 0
created_events = []

# Enable logging
openleadr.enable_default_logging(level=logging.INFO, file_name='mylogVTN.txt')
logger = logging.getLogger('openleadr')

# This function decides if the VEN registration is allowed. Currently only one
ven is allowed to register (test_VEN)
async def register_client(registration_info):
    global DR_program, ven_resources
    if registration_info['ven_name']=='test_VEN':
        ven_id = 'ven1'
        registration_id = 'registration1'
        DR_program = 'DLC'
        ven_resources = ['HVAC','Load']
        logger.info(f"Client {registration_info['ven_name']} has registered
succesfully as ven1")
        print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S'))
+']'  + f"Client {registration_info['ven_name']} has registered succesfully as
ven1")
        return ven_id, registration_id
    else:
        print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S'))
+']'  + f"Client {registration_info['ven_name']} not allowed to register")
        logger.info(f"Client {registration_info['ven_name']} not allowed to re-
gister")
        return False
```

```
# This function decides which reports and at what rate should be sent by the
VEN.
# All reports are accepted, at the minimum sampling rate defined by the VEN
async def register_report(ven_id, resource_id, measurement, unit, scale,
min_sampling_interval, max_sampling_interval):
    desired_sampling_interval = min_sampling_interval
    callback = partial(store_data, ven_id=ven_id, resource_id=resource_id,
measurement=measurement, unit=unit)
    print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'
+ f"Report for {resource_id} containing {measurement} in {unit} will be re-
ceived every {desired_sampling_interval}")
    logger.info(f"Report for {resource_id} containing {measurement} in {unit}
will be received every {desired_sampling_interval}")
    return callback, desired_sampling_interval

# This function sets the callback when report updates are received
async def update_report(resource_id, measurement, unit, scale, min_sampling_in-
terval, max_sampling_interval):
    callback = partial(store_data, ven_id=ven_id, resource_id=resource_id,
measurement=measurement)
    return callback

# When a new report is received the following function is called:
async def store_data(data, ven_id, resource_id, measurement, unit):
    global event_count
    logger.info(f"Report from {resource_id} received.")
    for time, value in data:
        value=int(value)
        logger.info(f"{measurement} at {time.strftime('%H:%M:%S')} for {re-
source_id} is {value} {unit}")
        file_name = ven_id + '/' + resource_id + '_DB.txt'
        DB_txt=open('C:/Users/Lenovo/TFM_IREC/openADR_tests/' + file_name,"a")
        DB_txt.write(str(time) + ";" + str(value)+ f"{unit}" +"\n")
        DB_txt.close
        logger.info(f"Data Point stored in folder {ven_id} at file
{resource_id}_DB.txt")
        random_event_creation = randint(0,100)
        if random_event_creation >80:
            event_count = event_count + 1
            print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:
%S')) +']' + f"Adding {DR_program} event for {ven_id}")
            logger.info(f"Adding {DR_program} event for {ven_id}")
            new_event = event_creation(ven_id, ven_resources, DR_program,
event_count)
            print_event_object(new_event)
            server.add_raw_event(ven_id=ven_id, event=new_event,
callback=event_callback)
        elif resource_id == ven_resources[-1]:
            logger.debug('No event will be added')

# This functions informs of the VEN response
async def event_callback(ven_id, event_id, opt_type):
    global created_events
    if event_id not in created_events:
        created_events.append(event_id)
        print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S'))
+']'  + f"Client responded {opt_type} to event {event_id}")
        logger.info(f"Client responded {opt_type} to event {event_id}")

# Handler to manage registrations of VENs
```

```
server.add_handler('on_create_party_registration', register_client)

# Handler to manage report registration
server.add_handler('on_register_report', register_report)

# Handler to manage reports after being registered
server.add_handler('on_update_report', update_report)

loop=asyncio.get_event_loop()
loop.create_task(server.run_async())
loop.run_forever()
```

2. test_VEN Python Script

```
"""
This script is used for the openADR implementation of a VEN to test different
signals and event programs
"""
import asyncio
from openleadr import OpenADRClient
from functools import partial
from datetime import timedelta, datetime, timezone
from aux_functions import print_event_dict
import logging
import openleadr
from random import randint

# Name of the VEN
ven_name = 'test_VEN'

# Resource names that the VEN enrolls to the program and maximum power of each
of them
ven_resources = ['HVAC','Load']
ven_resources_max_power = [3000,1000]

 # Demand Response Program to test. Options:
 # 'Simple' --> events only contain a simple signal
 # 'CPP' (Critical Peak Pricing) --> events contain a simple signal + electri-
city_price_signal
 # 'DLC' (Direct Load Control) --> events contain a simple signal + load_con-
trol

DR_program = 'DLC'
opt_count = 0

# Possible report rate for all assets
report_rate = timedelta(seconds=30)

# Enable logging to external file
openleadr.enable_default_logging(level=logging.INFO, file_name='mylogVEN.txt')
logger = logging.getLogger('openleadr')

# Creation of the VEN
client = OpenADRClient(ven_name=ven_name, vtn_url='http://localhost:8080/Open-
ADR2/Simple/2.0b')
print()
print("*" * 80)
print("VEN Information display".center(80))
print("*" * 80)
print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'  +
'Requesting registration to VTN...')
```

```
logger.info('Requesting registration to VTN...')

# Create random values for power variables to send in reports
async def read_power(resource_id):
    ind = ven_resources.index(resource_id)
    w = randint(0,ven_resources_max_power[ind])
    logger.info(f"A new report for {resource_id} will be sent to VTN. Current
power: {w} W")
    return w

# When an event is called the VEN must respond with optIn or optOut. For CPP
and DLC always optIn, for the rest the schedule is checked
async def handle_event(event):
    global opt_count, DR_program
    print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'
+ 'An event has been received')
    print_event_dict(event)
    if DR_program == 'CPP' or DR_program == 'DLC':
        opt_status = 'optIn'
    # Responds optOut to second event received
    elif DR_program == 'Simple':
        opt_count = opt_count +1
        if opt_count == 2:
            opt_status ='optOut'
        else:
            opt_status = 'optIn'
    print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'
+ f"Will {opt_status} to {event['event_descriptor']['event_id']}. Informing the
VTN...")
    return opt_status

async def update_event(event):
    global DR_program
    status = event['event_descriptor']['event_status']
    ID = event['event_descriptor']['event_id']
    print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'
+ 'An event update has been received')
    print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'
+ f"Event {ID} is now {status}")
    logger.info(f"An event update has been received. Event {ID} is now
{status}")
    if status == 'near':
        logger.info(f"Ramp-up period ongoing")
    if DR_program == 'CPP' or DR_program == 'DLC':
        opt_status = 'optIn'
    elif DR_program == 'Simple':
        opt_status =  'optIn'
    return opt_status

# Add reports
for resources in ven_resources:
    client.add_report(callback=partial(read_power,resources),
                      resource_id=resources,
                      measurement='power',
                      sampling_rate=report_rate,
                      unit='W')
    print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'
+ f"Adding reporting capabilities for {resources}. Power can be sent every {re-
port_rate}")
```

```
#Add event handlers
client.add_handler('on_event', handle_event)
client.add_handler('on_update_event', update_event)


loop=asyncio.get_event_loop()
loop.create_task(client.run())
loop.run_forever()
```

3. Auxiliary Scripts

```
"""
Auxiliary functions to print events on screen and create events for specific DR
programs
"""
from openleadr.objects import Event, Target, Interval, EventDescriptor, Event-
Signal, ActivePeriod
from datetime import timedelta, datetime, timezone
from random import randint, choices

# Function to print event dictionaries on screen
def print_event_dict(event):
    print("-" * 80)
    print(f"Event ID: {event['event_descriptor']['event_id']}")
    targets=event['targets']
    print(f"The target for the event is: {targets}")

    event_signals=event['event_signals']
    signal_number=len(event_signals)
    SIMPLE_signal=event_signals[0]

    intervals_SIMPLE=SIMPLE_signal['intervals']
    interval_number=len(intervals_SIMPLE)

    if signal_number==1:

        print('Event only contains a SIMPLE Signal with the following informa-
tion.')
        print(f"The event has {interval_number} interval(s)")
        i=1
        for interval in intervals_SIMPLE:
            print(f" -------- Interval number {i} ---------")
            print(f"Event starts at: {interval['dtstart']}")
            print(f"Event duration is: {interval['duration']}")
            print(f"Event level is: {interval['signal_payload']}")
            i=i+1

    elif signal_number==2:

        second_signal=event_signals[1]
        intervals_second=second_signal['intervals']
        print(f"Event received contains {signal_number} signals.")
        print(f"The event has {interval_number} interval(s)")
        i=1
        for interval in intervals_SIMPLE:
            print(f" -------- Interval number {i} ---------")
            print(f"Interval starts at: {interval['dtstart']}")
            print(f"Interval duration is: {interval['duration']}")
            int_num=i-1
```

```
            print_second_signal_dict(second_signal, int_num)
            print(f"Interval priority is: {interval['signal_payload']}")
            i=i+1

    else:
        print('Event received contains more than two signals. Printing not
defined')
    print("-" * 80)

def print_second_signal_dict(event_signal, int_num):

    signal_name = event_signal['signal_name'].upper()
    signal_type = event_signal['signal_type'].lower()
    interval=event_signal['intervals'][int_num]

    if signal_name=='ELECTRICITY_PRICE':

        print('The electricity price for this interval is:')
        if signal_type=='price':
            print(f"{interval['signal_payload']}")
        if signal_type=='pricerelative':
            print(f"{interval['signal_payload']} in respect to the usual tar-
iff")
        if signal_type=='pricemultiplier':
            print(f"{interval['signal_payload']} times the usual tariff")

    elif signal_name=='ENERGY PRICE':
        print('Event description to be defined!! Check script')

    elif signal_name=='DEMAND CHARGE':
        print('Event description to be defined!! Check script')

    elif signal_name[0:2]=='BID':
        print('Event description to be defined!! Check script')

    elif signal_name=='LOAD_DISPATCH':
        print('Event description to be defined!! Check script')

    elif signal_name=='LOAD_CONTROL':
        print('The VTN will change the setpoint of the load...')
        if signal_type=='x-loadcontrolcapacity':
            if interval['signal_payload']==0:
                print('The load will be turned OFF')
            elif interval['signal_payload']==1:
                print('The load will work at maximum capacity')
            else:
               print(f"The load will work at {interval['signal_payload']} times
             the     maximum capacity")
        if signal_type=='x-loadcontrolleveloffset':
            if interval['signal_payload']==0:
                print('The load will work at normal operation')
            elif interval['signal_payload']>0:
             print(f"The load will work over normal operation. Load control
             level: {interval['signal_payload']}")
            elif interval['signal_payload']<0:
             print(f"The load will work below normal operation. Load control
             level: {interval['signal_payload']}")
        if signal_type=='x-loadcontrolporcentoffset':
             print(f"The load will work with an offset of:
             {interval['signal_payload']}/1.0 over normal operation")
```

```
        if signal_type=='x-loadcontrolsetpoint':
             print(f"The load will work at control setpoint:
             {interval['signal_payload']}")

    else:
       print(f"Signal name {signal_name} is not an standard signal defined in
       OpenADR specifications")
       print(f"Signal type {signal_type} is not an standard type defined in
       OpenADR specifications")
        print(f"Payload is {interval['signal_payload']}")

# Function to create events for DLC, CPP and Simple events
def event_creation(ven_id, ven_resources, DR_program, event_count):
    # Creation of random start and duration of events
    random_seconds_delay=randint(30,60)
    random_seconds_duration=randint(30,60)
    event_start = datetime.now(timezone.utc) +
timedelta(seconds=random_seconds_delay)
    event_duration = timedelta(seconds=random_seconds_duration)

    # Choosing random assets from ven_resources list
    id_list = choices(ven_resources, k=1)
    target_list =[]
    for ids in id_list:
        target_list.append(Target(resource_id=ids))

    if DR_program == 'Simple':
        # Simple signal should be an integer between 1-3
        simple_signal = randint(1,3)
        event = Event(event_descriptor=EventDescriptor(event_id='Simple_event'
+ str(event_count),
                                                       modification_number=0,
                                                       event_status='far',
                                                       market_context=None),
                      event_signals=[EventSignal(signal_id='signal001',
                                                 signal_type='level',
                                                 signal_name='simple',
                                                intervals= [Interval(dtstart=
                                                event_start, duration=event_dura-
                                                tion,
                                                 signal_payload=simple_signal)])],
                     targets=target_list)
    if DR_program == 'CPP':
    # Events have two intervals, first electricity is 6 times higher and then
10 times
        event = Event(event_descriptor=EventDescriptor(event_id='CPP_event' +
str(event_count),

modification_number=0,
                                                       event_status='far',
                                                       market_con-
text=None),
                           event_signals=[EventSignal(signal_id='signal001',
                                                      signal_type='level',
                                                      signal_name='simple',
                                                intervals =[Inter-
                                                val(dtstart=event_start,
                                                duration=event_duration/2,
                                                 signal_payload=1),
```

```
                                                Interval(dtstart=event_start +
                                                event_duration/2,
                                                duration=event_duration/2,
                                                signal_payload=2)]),
                                            EventSignal(signal_id='signal002',
                                                        signal_type='priceMulti-
plier',

signal_name='ELECTRICITY_PRICE',
                                                     intervals  =[Inter-
                                                val(dtstart=event_start,

                                                      duration=event_duration/2,
                                                          signal_payload=6),
                                                Interval(dtstart=event_start +
                                                  event_duration/2,
                                                       duration=event_duration/2,
                                                      signal_payload=10)])],
                            # Target not specified, only venID
                            targets=[Target(ven_id=ven_id)])
    if DR_program == 'DLC':
        # Create random offset for event (as a value between -10 and 10, but
not 0)
        value = randint(1,10)
        sign_list = choices([-1,1],k=1)
        sign=sign_list[0]
        event_leveloffset = value * sign

        # Decide importance of event depending on level offset
        if (event_leveloffset>3 and event_leveloffset<8) or
(event_leveloffset>-7 and event_leveloffset<-3):
            event_imp = 2
        elif event_leveloffset>7 or event_leveloffset<-7:
            event_imp = 3
        else:
            event_imp = 1

        if target_list[0].resource_id=='HVAC':
            ramp=timedelta(seconds=15)
        else:
            ramp=None

        event = Event(event_descriptor=EventDescriptor(event_id='DLC_event' +
str(event_count),

modification_number=0,
                                                       event_status='far',
                                                       market_con-
text=None),
                            event_signals=[EventSignal(signal_id='signal001',
                                                       signal_type='level',
                                                       signal_name='simple',
                                                    intervals=[ Inter-
                                        val(dtstart=event_start,
                                                duration=event_duration,
                                               signal_payload=event_imp)]),
                                           EventSignal(signal_id='signal002',
                                                       signal_type='x loadControl-
                                        LevelOffset',
```

```
                                                    signal_name='LOAD_CON-
TROL',
                                                intervals=[ Inter-
                                            val(dtstart=event_start,
                                                  duration=event_duration,
                                                signal_payload=event_levelof
                                                fset)])],
                                active_period=ActivePeriod(dtstart=event_start,
                                                           duration=event_duration,
                                                           ramp_up_period=ramp),
                                targets=target_list)
    return event
```

4. Message examples:

As the payloads for each message sent between VTN and VEN are very extensive, in the event logging this payload is not included. In this section an example of each message is shown in the form of python dictionaries for easier interpretation. In reality they are sent as XML payloads. Each message contains an explanation of the meaning of it and the payload. The format used is the same as used later on the logging.

oadrCreatePartyRegistration
Sent by the VEN to ask for registration, includes main information about VEN
{'ven_name': 'test_VEN', 'http_pull_model': True, 'xml_signature': False, 'report_only': False, 'profile_name': '2.0b', 'transport_name': 'simpleHttp', 'transport_address': None}
oadrCreatedPartyRegistration
Sent by VTN to confirm that the VEN has been registered. Includes registration information and polling frequency
{'ven_id': 'ven1', 'registration_id': 'registration1', 'profiles': [{'profile_name': '2.0b', 'transports': [{'transport_name': 'simpleHttp'}]}], 'requested_oadr_poll_freq': datetime.timedelta(seconds=10), 'vtn_id': 'test_VTN', 'response': {'request_id': '965cdfcd-9907-42a0-bdb0-6aeeab706ca8', 'response_code': 200, 'response_description': 'OK'}, 'request_id': '9774d7e5-3c03-4c98-9e14-f191b87fdc0a'}
oadrRegisterReport
Sent by the VEN to inform about reporting capabilities. Includes possible reports (measurement, unit…) that the VEN can generate and the frequency.
{'request_id': '0798355c-7320-4a66-85df-4fb253119c3b', 'ven_id': 'ven1', 'reports': [Report(report_specifier_id='bd73841e-6893-4e2e-b423-23d6e8684e22', report_name='TELEMETRY_USAGE', report_request_id=None, report_descriptions=[ReportDescription(r_id='cd88d0cc-9f67-416e-8f54-b146dab5f1bd', market_context=None, reading_type='Direct Read', report_subject=Target('resource_id=HVAC'), report_data_source=Target('resource_id=HVAC'), report_type='reading', sampling_rate=SamplingRate(min_period=datetime.timedelta(seconds=30), max_period=datetime.timedelta(seconds=30), on_change=False),

measurement=Measurement(name='customUnit', description='power', unit='W', scale='none', power_attributes=None, pulse_factor=None, ns='oadr')), ReportDescription(r_id='eeb2d80e-7410-4f13-90d0-84fb44750963', market_context=None, reading_type='Direct Read', report_subject=Target('resource_id=Load'), report_data_source=Target('resource_id=Load'), report_type='reading', sampling_rate=SamplingRate(min_period=datetime.timedelta(seconds=30), max_period=datetime.timedelta(seconds=30), on_change=False), measurement=Measurement(name='customUnit', description='power', unit='W', scale='none', power_attributes=None, pulse_factor=None, ns='oadr'))], created_date_time=datetime.datetime(2021, 1, 6, 17, 22, 21, 294266), dtstart=None, duration=None, intervals=None, data_collection_mode='incremental')], 'report_request_id': 0}

oadrRegisteredReport

Sent by the VTN to choose the reports to be received by the VEN. It includes the report information and the desired sampling rate.

{'report_requests': [(report_request_id='e6e4a564-9b37-4125-8411-a25dd83ec1d0', report_specifier=(report_specifier_id='2ae7ad2b-122d-48ec-b0de-24f247f68000', granularity=datetime.timedelta(seconds=30), specifier_payloads=[(r_id='e9ff4056-d708-499e-8e80-478fcf564c58', reading_type='Direct Read', measurement=None), (r_id='67816b5a-0983-4a6b-b6c3-b705f51610ae', reading_type='Direct Read', measurement=None)], report_interval=None, report_back_duration=datetime.timedelta(seconds=30)))], 'vtn_id': 'test_VTN', 'ven_id': 'ven1', 'response': {'request_id': 'cf51919e-8ee7-4c14-8a30-04902220371d', 'response_code': 200, 'response_description': 'OK'}, 'request_id': '59771c84-db75-4d02-9779-b47e036358aa'}

oadrUpdateReport

Sent by VEN when a new report is sent. It includes the report information and the time and value of the measurement sent.

report_specifier_id='bd73841e-6893-4e2e-b423-23d6e8684e22', report_name='TELEMETRY_USAGE', report_request_id='9a95ea4b-9627-4699-8083-6af03c2374c7', report_descriptions=[], created_date_time=datetime.datetime(2021, 1, 6, 16, 22, 30, 15914, tzinfo=datetime.timezone.utc), dtstart=None, duration=None, intervals=[ReportInterval(dtstart=datetime.datetime(2021, 1, 6, 16, 22, 30, 15914, tzinfo=datetime.timezone.utc), report_payload=ReportPayload(r_id='cd88d0cc-9f67-416e-8f54-b146dab5f1bd', value=2389, confidence=None, accuracy=None), duration=None), ReportInterval(dtstart=datetime.datetime(2021, 1, 6, 16, 22, 30, 15914, tzinfo=datetime.timezone.utc), report_payload=ReportPayload(r_id='eeb2d80e-7410-4f13-90d0-84fb44750963', value=228, confidence=None, accuracy=None), duration=None)], data_collection_mode='incremental')

oadrUpdatedReport

Sent by the VTN to acknowledge that the report has been received.

{'vtn_id': 'test_VTN', 'response': {'request_id': None, 'response_code': 200, 'response_description': 'OK'}, 'request_id': 'c41518ee-f3c5-4960-8c88-0a6fe2c419e6'}

oadrPoll

Sent by the VEN when polling for new messages. It only includes the VEN ID

{‘ven_id’ = ‘ven1’}
oadrResponse
Sent by the VTN to acknowledge message reception.
{'vtn_id': 'test_VTN', 'ven_id': 'ven1', 'response': {'request_id': None, 'response_code': 200, 'response_description': 'OK'}, 'request_id': 'edfd359e-87fd-419b-8b60-7215e32b0c32'}
oadrDistributeEvent
Sent by the VTN to send event information. It includes all the features of the events (signal type, active period…)
{'events': [{'event_descriptor': {'event_id': 'Simple_event1', 'modification_number': 0, 'market_context': None, 'event_status': 'far', 'created_date_time': datetime.datetime(2021, 1, 6, 15, 59, 7, 358966, tzinfo=datetime.timezone.utc), 'modification_date_time': datetime.datetime(2021, 1, 6, 15, 59, 7, 358966, tzinfo=datetime.timezone.utc), 'priority': 0, 'test_event': False, 'vtn_comment': None}, 'event_signals': [{'intervals': [{'dtstart': datetime.datetime(2021, 1, 6, 15, 59, 47, 358966, tzinfo=datetime.timezone.utc), 'duration': datetime.timedelta(seconds=42), 'signal_payload': 1, 'uid': None}], 'signal_name': 'simple', 'signal_type': 'level', 'signal_id': 'signal001', 'current_value': None, 'targets': None, 'targets_by_type': None, 'measurement': None}], 'targets': [{'aggregated_p_node': None, 'end_device_asset': None, 'meter_asset': None, 'p_node': None, 'service_area': None, 'service_delivery_point': None, 'service_location': None, 'transport_interface': None, 'group_id': None, 'group_name': None, 'resource_id': 'Load', 'ven_id': None, 'party_id': None}], 'targets_by_type': {'resource_id': ['Load']}, 'active_period': {'dtstart': datetime.datetime(2021, 1, 6, 15, 59, 47, 358966, tzinfo=datetime.timezone.utc), 'duration': datetime.timedelta(seconds=42), 'tolerance': None, 'notification_period': None, 'ramp_up_period': None, 'recovery_period': None}, 'response_required': 'always'}], 'vtn_id': 'test_VTN', 'ven_id': 'ven1', 'response': {'request_id': None, 'response_code': 200, 'response_description': 'OK'}, 'request_id': '241c4d56-ab86-4b31-b295-6808acf5d4b5'}
oadrCreatedEvent
Sent by the VEN to acknowledge to VTN that the event or event modification has been received and to send opt responses.
{'response_code': 200, 'response_description': 'OK', 'opt_type': 'optIn', 'request_id': '7a09cb58-fa26-4847-9650-ead6517a8249', 'modification_number': 1, 'event_id': 'Simple_event2'}

5. Simple event logging

Note that the polling for new messages is replaced by “…” when no information needs to be sent from VTN to VEN.

5.1. VTN

17:05:22: Client test_VEN has registered succesfully as ven1
17:05:22: Responding to **oadrCreatePartyRegistration** with **oadrCreatedPartyRegistration** message
17:05:22: Report for HVAC containing power in W will be received every 0:00:30

17:05:22: Report for Load containing power in W will be received every 0:00:30
17:05:22: Responding to **oadrRegisterReport** with **oadrRegisteredReport** message
…
17:05:40: Report from HVAC received.
17:05:40: power at 17:05:30 for HVAC is 1458 W
17:05:40: Data Point stored in folder ven1 at file HVAC_DB.txt
17:05:40: Adding Simple event for ven1
17:05:40: Report from Load received.
17:05:40: power at 17:05:30 for Load is 842 W
17:05:40: Data Point stored in folder ven1 at file Load_DB.txt
17:05:40: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
17:05:40: Responding to **oadrPoll** with **oadrDistributeEvent** message
17:05:40: Client responded optIn to event Simple_event1
17:05:40: Responding to **oadrCreatedEvent** with **oadrResponse** message
…
17:06:04: Report from HVAC received.
17:06:04: power at 17:06:00 for HVAC is 383 W
17:06:04: Data Point stored in folder ven1 at file HVAC_DB.txt
17:06:04: Report from Load received.
17:06:04: power at 17:06:00 for Load is 885 W
17:06:04: Data Point stored in folder ven1 at file Load_DB.txt
17:06:04: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
…
17:06:19: Responding to **oadrPoll** with **oadrDistributeEvent** message
17:06:19: Responding to **oadrCreatedEvent** with **oadrResponse** message
…
17:06:33: Report from HVAC received.
17:06:33: power at 17:06:30 for HVAC is 2136 W
17:06:33: Data Point stored in folder ven1 at file HVAC_DB.txt
17:06:33: Adding Simple event for ven1
17:06:33: Report from Load received.
17:06:33: power at 17:06:30 for Load is 197 W
17:06:33: Data Point stored in folder ven1 at file Load_DB.txt
17:06:33: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
17:06:40: Responding to **oadrPoll** with **oadrDistributeEvent** message
17:06:40: Client responded optOut to event Simple_event2
17:06:40: Responding to **oadrCreatedEvent** with **oadrResponse** message
…
17:06:59: Responding to **oadrPoll** with **oadrDistributeEvent** message
17:06:59: Responding to **oadrCreatedEvent** with **oadrResponse** message
…

17:07:14: Report from HVAC received.
17:07:14: power at 17:07:00 for HVAC is 559 W
17:07:14: Data Point stored in folder ven1 at file HVAC_DB.txt
17:07:14: Report from Load received.
17:07:14: power at 17:07:00 for Load is 403 W
17:07:14: Data Point stored in folder ven1 at file Load_DB.txt
17:07:14: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
…
17:07:43: Report from HVAC received.
17:07:43: power at 17:07:30 for HVAC is 1025 W
17:07:43: Data Point stored in folder ven1 at file HVAC_DB.txt
17:07:43: Report from Load received.
17:07:43: power at 17:07:30 for Load is 678 W
17:07:43: Data Point stored in folder ven1 at file Load_DB.txt
17:07:43: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
17:07:50: Responding to **oadrPoll** with **oadrResponse** message

5.2 VEN

17:05:20: Requesting registration to VTN...
17:05:20: VEN is sending **oadrCreatePartyRegistration** message
17:05:22: VEN is now registered with ID ven1
17:05:22: The polling frequency is 0:00:10
17:05:22: VEN is sending **oadrRegisterReport** message
17:05:22: VEN is sending **oadrCreatedReport** message
…
17:05:30: A new report for HVAC will be sent to VTN. Current power: 1458 W
17:05:30: Adding 2021-01-30 17:05:30.027034+00:00, 1458 to report
17:05:30: A new report for Load will be sent to VTN. Current power: 842 W
17:05:30: Adding 2021-01-30 17:05:30.027034+00:00, 842 to report
…
17:05:40: VEN is sending **oadrUpdateReport** message
17:05:40: Sending **oadrPoll** message
17:05:40: The VEN received an event
17:05:40: Responding to event with **oadrCreatedEvent**
…
17:06:00: A new report for HVAC will be sent to VTN. Current power: 383 W
17:06:00: Adding 2021-01-30 17:06:00.016039+00:00, 383 to report
17:06:00: A new report for Load will be sent to VTN. Current power: 885 W
17:06:00: Adding 2021-01-30 17:06:00.017038+00:00, 885 to report
….

17:06:19: Sending **oadrPoll** message
17:06:19: The VEN received an event
17:06:19: An event update has been received. Event Simple_event1 is now active
17:06:19: Responding to event with **oadrCreatedEvent**
….
17:06:30: A new report for HVAC will be sent to VTN. Current power: 2136 W
17:06:30: Adding 2021-01-30 17:06:30.013728+00:00, 2136 to report
17:06:30: A new report for Load will be sent to VTN. Current power: 197 W
17:06:30: Adding 2021-01-30 17:06:30.014727+00:00, 197 to report
….
17:06:33: VEN is sending **oadrUpdateReport** message
17:06:40: Sending **oadrPoll** message
17:06:40: The VEN received an event
17:06:40: Responding to event with **oadrCreatedEvent**
…
17:06:59: The VEN received an event
17:06:59: An event update has been received. Event Simple_event1 is now completed
17:06:59: Responding to event with **oadrCreatedEvent**
…
17:07:00: A new report for HVAC will be sent to VTN. Current power: 559 W
17:07:00: Adding 2021-01-30 17:07:00.011859+00:00, 559 to report
17:07:00: A new report for Load will be sent to VTN. Current power: 403 W
17:07:00: Adding 2021-01-30 17:07:00.012860+00:00, 403 to report
…
17:07:30: A new report for HVAC will be sent to VTN. Current power: 1025 W
17:07:30: Adding 2021-01-30 17:07:30.005738+00:00, 1025 to report
17:07:30: A new report for Load will be sent to VTN. Current power: 678 W
17:07:30: Adding 2021-01-30 17:07:30.005738+00:00, 678 to report
…
17:07:43: VEN is sending **oadrUpdateReport** message
17:07:50: Sending **oadrPoll** message
17:07:50: No events or reports available

6 CPP Event logging

Note that the polling for new messages is replaced by "…" when no information needs to be sent from VTN to VEN.

6.1 VTN

17:13:28: Client test_VEN has registered succesfully as ven1
17:13:28: Responding to **oadrCreatePartyRegistration** with

oadrCreatedPartyRegistration message
17:13:28: Report for HVAC containing power in W will be received every 0:00:30
17:13:28: Report for Load containing power in W will be received every 0:00:30
17:13:28: Responding to **oadrRegisterReport** with **oadrRegisteredReport** message
…
17:13:34: Report from HVAC received.
17:13:34: power at 17:13:30 for HVAC is 406 W
17:13:34: Data Point stored in folder ven1 at file HVAC_DB.txt
17:13:34: Adding CPP event for ven1
17:13:34: Report from Load received.
17:13:34: power at 17:13:30 for Load is 477 W
17:13:34: Data Point stored in folder ven1 at file Load_DB.txt
17:13:34: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
17:13:39: Responding to **oadrPoll** with **oadrDistributeEvent** message
17:13:39: Client responded optIn to event CPP_event1
17:13:39: Responding to **oadrCreatedEvent** with **oadrResponse** message
…
17:14:09: Report from HVAC received.
17:14:09: power at 17:14:00 for HVAC is 2153 W
17:14:09: Data Point stored in folder ven1 at file HVAC_DB.txt
17:14:09: Report from Load received.
17:14:09: power at 17:14:00 for Load is 766 W
17:14:09: Data Point stored in folder ven1 at file Load_DB.txt
17:14:09: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
…
17:14:39: Report from HVAC received.
17:14:39: power at 17:14:30 for HVAC is 2824 W
17:14:39: Data Point stored in folder ven1 at file HVAC_DB.txt
17:14:39: Report from Load received.
17:14:39: power at 17:14:30 for Load is 52 W
17:14:39: Data Point stored in folder ven1 at file Load_DB.txt
17:14:39: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
…
17:14:50: Responding to **oadrCreatedEvent** with **oadrResponse** message
…
17:15:07: Report from HVAC received.
17:15:07: power at 17:15:00 for HVAC is 617 W
17:15:07: Data Point stored in folder ven1 at file HVAC_DB.txt
17:15:07: Report from Load received.
17:15:07: power at 17:15:00 for Load is 291 W
17:15:07: Data Point stored in folder ven1 at file Load_DB.txt

17:15:07: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
…
17:15:33: Report from HVAC received.
17:15:33: power at 17:15:30 for HVAC is 311 W
17:15:33: Data Point stored in folder ven1 at file HVAC_DB.txt
17:15:33: Report from Load received.
17:15:33: power at 17:15:30 for Load is 425 W
17:15:33: Data Point stored in folder ven1 at file Load_DB.txt
17:15:33: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
…
17:16:00: Responding to **oadrPoll** with **oadrDistributeEvent** message
17:16:00: Responding to **oadrCreatedEvent** with **oadrResponse** message

6.2 VEN

17:13:26: Requesting registration to VTN...
17:13:26: VEN is sending **oadrCreatePartyRegistration** message
17:13:28: VEN is now registered with ID ven1
17:13:28: The polling frequency is 0:00:10
17:13:28: VEN is sending **oadrRegisterReport** message
17:13:28: VEN is sending **oadrCreatedReport** message
…
17:13:30: A new report for HVAC will be sent to VTN. Current power: 406 W
17:13:30: Adding 2021-01-06 17:13:30.002039+00:00, 406 to report
17:13:30: A new report for Load will be sent to VTN. Current power: 477 W
17:13:30: Adding 2021-01-06 17:13:30.002039+00:00, 477 to report
…
17:13:34: VEN is sending **oadrUpdateReport** message
17:13:39: Sending **oadrPoll** message
17:13:39: The VEN received an event
17:13:39: Responding to event with **oadrCreatedEvent**
…
17:14:00: Sending **oadrPoll** message
17:14:00: A new report for HVAC will be sent to VTN. Current power: 2153 W
17:14:00: Adding 2021-01-06 17:14:00.021913+00:00, 2153 to report
17:14:00: A new report for Load will be sent to VTN. Current power: 766 W
17:14:00: Adding 2021-01-06 17:14:00.021913+00:00, 766 to report
17:14:00: No events or reports available
17:14:09: VEN is sending **oadrUpdateReport** message
…
17:14:30: A new report for HVAC will be sent to VTN. Current power: 2824 W

17:14:30: Adding 2021-01-06 17:14:30.003850+00:00, 2824 to report
17:14:30: A new report for Load will be sent to VTN. Current power: 52 W
17:14:30: Adding 2021-01-06 17:14:30.003850+00:00, 52 to report
…
17:14:39: VEN is sending **oadrUpdateReport** message
…
17:14:50: Sending **oadrPoll** message
17:14:50: The VEN received an event
17:14:50: An event update has been received. Event CPP_event1 is now active
17:14:50: Responding to event with **oadrCreatedEvent**
…
17:15:00: A new report for HVAC will be sent to VTN. Current power: 617 W
17:15:00: Adding 2021-01-06 17:15:00.015444+00:00, 617 to report
17:15:00: A new report for Load will be sent to VTN. Current power: 291 W
17:15:00: Adding 2021-01-06 17:15:00.015444+00:00, 291 to report
…
17:15:07: VEN is sending **oadrUpdateReport** message
…
17:15:30: A new report for HVAC will be sent to VTN. Current power: 311 W
17:15:30: Adding 2021-01-06 17:15:30.003365+00:00, 311 to report
17:15:30: A new report for Load will be sent to VTN. Current power: 425 W
17:15:30: Adding 2021-01-06 17:15:30.003365+00:00, 425 to report
17:15:30: No events or reports available
17:15:33: VEN is sending **oadrUpdateReport** message
…
17:16:00: A new report for HVAC will be sent to VTN. Current power: 2552 W
17:16:00: Adding 2021-01-06 17:16:00.015161+00:00, 2552 to report
17:16:00: A new report for Load will be sent to VTN. Current power: 179 W
17:16:00: Adding 2021-01-06 17:16:00.015161+00:00, 179 to report
17:16:00: Sending **oadrPoll** message
17:16:00: The VEN received an event
17:16:00: An event update has been received. Event CPP_event1 is now completed
17:16:00: Responding to event with **oadrCreatedEvent**

7 DLC event logging

Note that the polling for new messages is replaced by "…" when no information needs to be sent from VTN to VEN.

7.1 VTN

20:10:20: Client test_VEN has registered succesfully as ven1
20:10:20: Responding to **oadrCreatePartyRegistration** with **oadrCreatedPartyRegistration** message
20:10:20: Report for HVAC containing power in W will be received every 0:00:30
20:10:20: Report for Load containing power in W will be received every 0:00:30
20:10:20: Responding to **oadrRegisterReport** with **oadrRegisteredReport** message
…
20:10:38: Report from HVAC received.
20:10:38: power at 20:10:30 for HVAC is 284 W
20:10:38: Data Point stored in folder ven1 at file HVAC_DB.txt
20:10:38: Adding DLC event for ven1
20:10:38: Report from Load received.
20:10:38: power at 20:10:30 for Load is 490 W
20:10:38: Data Point stored in folder ven1 at file Load_DB.txt
20:10:38: Adding DLC event for ven1
20:10:38: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
20:10:39: Responding to **oadrPoll** with **oadrDistributeEvent** message
20:10:39: Client responded optIn to event DLC_event1
20:10:39: Responding to oadrCreatedEvent with **oadrResponse** message
20:10:39: Responding to **oadrPoll** with **oadrDistributeEvent** message
20:10:39: Client responded optIn to event DLC_event2
20:10:39: Responding to **oadrCreatedEvent** with **oadrResponse** message
…
20:11:09: Report from HVAC received.
20:11:09: power at 20:11:00 for HVAC is 1850 W
20:11:09: Data Point stored in folder ven1 at file HVAC_DB.txt
20:11:09: Report from Load received.
20:11:09: power at 20:11:00 for Load is 440 W
20:11:09: Data Point stored in folder ven1 at file Load_DB.txt
20:11:09: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
…
20:11:30: Responding to **oadrPoll** with **oadrDistributeEvent** message
20:11:30: Responding to **oadrCreatedEvent** with **oadrResponse** message
20:11:30: Responding to **oadrPoll** with **oadrDistributeEvent** message
20:11:30: Responding to **oadrCreatedEvent** with **oadrResponse** message
…
20:11:36: Report from HVAC received.
20:11:36: power at 20:11:30 for HVAC is 79 W
20:11:36: Data Point stored in folder ven1 at file HVAC_DB.txt

20:11:36: Report from Load received.
20:11:36: power at 20:11:30 for Load is 751 W
20:11:36: Data Point stored in folder ven1 at file Load_DB.txt
20:11:36: Responding to **oadrUpdateReport** with **oadrUpdatedReport** message
20:11:40: Responding to **oadrPoll** with **oadrDistributeEvent** message
20:11:40: Responding to **oadrCreatedEvent** with **oadrResponse** message
…
20:11:59: Responding to **oadrPoll** with **oadrDistributeEvent** message
20:11:59: Responding to **oadrCreatedEvent** with **oadrResponse** message

7.2 VEN

20:10:18: Requesting registration to VTN...
20:10:18: VEN is sending **oadrCreatePartyRegistration** message
20:10:20: VEN is now registered with ID ven1
20:10:20: The polling frequency is 0:00:10
20:10:20: VEN is sending **oadrRegisterReport** message
20:10:20: VEN is sending **oadrCreatedReport** message
…
20:10:30: A new report for HVAC will be sent to VTN. Current power: 284 W
20:10:30: Adding 2021-01-06 20:10:30.012498+00:00, 284 to report
20:10:30: A new report for Load will be sent to VTN. Current power: 490 W
20:10:30: Adding 2021-01-06 20:10:30.012498+00:00, 490 to report
20:10:33: VEN is sending **oadrUpdateReport** message
…
20:10:39: The VEN received an event
20:10:39: Responding to event with **oadrCreatedEvent**
20:10:39: Sending **oadrPoll** message
20:10:39: The VEN received an event
20:10:39: Responding to event with **oadrCreatedEvent**
…
20:11:00: A new report for HVAC will be sent to VTN. Current power: 1850 W
20:11:00: Adding 2021-01-06 20:11:00.006318+00:00, 1850 to report
20:11:00: A new report for Load will be sent to VTN. Current power: 440 W
20:11:00: Adding 2021-01-06 20:11:00.006318+00:00, 440 to report
20:11:03: VEN is sending **oadrUpdateReport** message
…
20:11:30: A new report for HVAC will be sent to VTN. Current power: 79 W
20:11:30: Adding 2021-01-06 20:11:30.008002+00:00, 79 to report
20:11:30: A new report for Load will be sent to VTN. Current power: 751 W
20:11:30: Adding 2021-01-06 20:11:30.008002+00:00, 751 to report

20:11:30: Sending **oadrPoll** message
20:11:30: The VEN received an event
20:11:30: An event update has been received. Event DLC_event2 is now near
20:11:30: Ramp-up period ongoing
20:11:30: Responding to event with **oadrCreatedEvent**
20:11:30: Sending **oadrPoll** message
20:11:30: The VEN received an event
20:11:30: An event update has been received. Event DLC_event1 is now active
20:11:30: Responding to event with **oadrCreatedEvent**
…
20:11:36: VEN is sending **oadrUpdateReport** message
20:11:40: Sending **oadrPoll** message
20:11:40: The VEN received an event
20:11:40: An event update has been received. Event DLC_event2 is now active
20:11:40: Responding to event with **oadrCreatedEvent**
…
20:11:59: Sending **oadrPoll** message
20:11:59: The VEN received an event
20:11:59: An event update has been received. Event DLC_event1 is now completed
20:11:59: Responding to event with **oadrCreatedEvent**

ANNEX I. OpenADR SCADA client definition

openADR_client.py

```
"""
This script is used for the openADR VEN implementation in the LAB
"""
import asyncio
from openleadr.client import OpenADRClient
from functools import partial
from datetime import timedelta, datetime, timezone
from aux_functions import print_event_dict
import logging
import openleadr
from random import randint

# Name of the VEN
ven_name = 'smartlab'
# Resource names that the VEN enrolls to the program
ven_resources = ['Load']
# Possible report rate for the assets
report_rate = timedelta(seconds=60)
# URL of VTN
vtn_url = 'http://localhost:8080/OpenADR2/Simple/2.0b'

dir = 'E:\\InduSoft Web Studio v8.1 Projects\\SCADA_v5\\Web\\Recipes\\'

# Logging to external file
openleadr.enable_default_logging(level=logging.INFO)
logger = logging.getLogger('openleadr')
handler = logging.FileHandler('VEN_logfile.txt')
logger.addHandler(handler)
default_handler = logger.handlers[0]
logger.removeHandler(default_handler)
formatter = logging.Formatter(
    '%(asctime)s : %(message)s',"%H:%M:%S")
handler.setFormatter(formatter)

# Creation of the VEN
client = OpenADRClient(ven_name=ven_name, vtn_url=vtn_url)
print()
print("*" * 80)
print("VEN Information display".center(80))
print("*" * 80)
print('['+  str(datetime.now(timezone.utc).time().strftime('%H:%M:%S'))  +']'    +
'Requesting registration to VTN...')
logger.info('Requesting registration to VTN...')
```

```
# Read load power from the recipe
async def read_power(resource_id):
    with open(dir + "LOAD.DAT", 'r') as f:
        w = float(f.readline())
     print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'
+ f"A new report for {resource_id} will be sent to VTN. Current power: {w} W")
     logger.info(f"A new report for {resource_id} will be sent to VTN. Current
power: {w} W")
    return w

# When an event is called the VEN must respond with optIn or optOut.
async def handle_event(event):
     print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'
+ 'An event has been received')
    print_event_dict(event)
    opt_status = 'optIn'
    signal_number = len(event['event_signals'])
    # If the event contains one signal we read the first one, if it contains more
we read the second one
    if signal_number == 1:
        num = 0
    if signal_number > 1:
        num = 1
    interval_number = len(event['event_signals'][num]['intervals'])
    # We write the event data in the openADR recipe
    with open(dir + "ACTIVATIONS_OPENADR.DAT", "w") as f:
       f.write(str(event['event_signals'][num]['intervals'][0]['signal_payload'])
+ "\n")
        f.write(str(event['event_descriptor']['event_status']) + "\n")
        f.write(str(event['event_signals'][num]['signal_type']) + "\n")
         f.write(str(event['event_signals'][num]['intervals'][0]['dtstart']) + "\
n")
        f.write(str(event['event_signals'][num]['intervals'][0]['duration']) + "\
n")
        f.write(str(opt_status) + "\n")
        f.write(str(interval_number) + "\n")
     print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'
+ f"Will {opt_status} to {event['event_descriptor']['event_id']}. Informing the
VTN...")
    return opt_status

# When an event is updated (near, completed) the VEN must respond with optIn or
optOut.
async def update_event(event):
    status = event['event_descriptor']['event_status']
    ID = event['event_descriptor']['event_id']
    signal_number = len(event['event_signals'])
```

```
     print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'
+ 'An event update has been received')
     print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'
+ f"Event {ID} is now {status}")
    logger.info(f"An event update has been received. Event {ID} is now {status}")
    opt_status =  'optIn'
    # If the event contains one signal we read the first one, if it contains more
we read the second one
    if signal_number == 1:
        num = 0
    if signal_number > 1:
        num = 1
    interval_number = len(event['event_signals'][num]['intervals'])
    # We write the event data in the openADR recipe
    with open(dir + "ACTIVATIONS_OPENADR.DAT", "w") as f:
       f.write(str(event['event_signals'][num]['intervals'][0]['signal_payload'])
+ "\n")
        f.write(str(event['event_descriptor']['event_status']) + "\n")
        f.write(str(event['event_signals'][num]['signal_type']) + "\n")
         f.write(str(event['event_signals'][num]['intervals'][0]['dtstart']) + "\
n")
        f.write(str(event['event_signals'][num]['intervals'][0]['duration']) + "\
n")
        f.write(str(opt_status) + "\n")
        f.write(str(interval_number) + "\n")
    return opt_status

# Add reports
for resource in ven_resources:
    client.add_report(callback=partial(read_power,resource),
                      resource_id=resource,
                      measurement='power',
                      sampling_rate=report_rate,
                      unit='W')
     print('['+ str(datetime.now(timezone.utc).time().strftime('%H:%M:%S')) +']'
+  f"Adding  reporting  capabilities  for  {resource}.  Power  can  be  sent  every
{report_rate}")

#Add event handlers
client.add_handler('on_event', handle_event)
client.add_handler('on_update_event', update_event)

loop=asyncio.get_event_loop()
loop.create_task(client.run())
loop.run_forever()
```

ANNEX J. Gantt Diagram

Task name	Duration	OCTOBER 2020	NOVEMBER 2020	DECEMBER 2020	JANUARY 2021	FEBRUARY 2021	MARCH 2021	APRIL 2021	MAY 2021	JUNE 2021
Completion of the project	252									
1 **Literature review & other learning**	58									
1.1 Demand Response	35									
1.2 Communication protocols	22									
1.3 ITME software	15									
1.4 Python programming	14									
1.5 InfluxDB API	20									
1.6 SCADA	28									
2 **Laboratory preparation work**	198									
2.1 Scenario definition	8									
2.2 Profile selection	13									
2.3 InfluxDB API tests	20									
2.4 InfluxDB Python script	14									
2.5 Configuration on the SCADA	139									
2.6 DEMO screen	119									
2.7 Virtual HVAC definition	34									
2.8 Battery control definition	15									
3 **OpenADR Integration**	47									
3.1 openleadr exploration	47									
3.2 VEN,VTN definition (Python)	36									
4 **Experimental work**	35									
4.1 Scenario development	29									
4.2 openADR testing	1									
5 **Memory draft**	161									
5.1 Chapter 1 – Intro	70									
5.2 Chapter 2 – SmartLab	31									
5.3 Chapter 3 – Core	89									
5.4 Chapter 4 – Scenarios	60									
5.5 Chapter 5 – openADR	27									
5.6 Chapter 6 – Conclusions	8									
First draft										
Second draft										
Final document										

ETSEIB