A new approach for distributed deployment of centralized algorithms in smart grid

Thi-Thanh-Quynh Nguyen, Vincent Debusschere, Christophe Bobineau, Antoine Labonne, Cédric Boudinet, Quang Huy Giap, Nouredine Hadjsaid

A new approach for the distributed deployment of centralized algorithms in smart grids

T.T.Q. Nguyen[1, 2], V. Debusschere[1], Ch. Bobineau[2], A. Labonne[1], C. Boudinet[1], Q.H. Giap[3], and N. HadjSaid[1]

[1]Univ. Grenoble Alpes, CNRS, Grenoble INP*, G2ELab, 38000 Grenoble, France
[2]Univ. Grenoble Alpes, CNRS, Grenoble INP*, LIG, 38000 Grenoble, France
[3]Da Nang University of Technology, 550000 Da Nang, Vietnam
thi-thanh-quynh.nguyen@g2elab.grenoble-inp.fr

***Abstract*—Distributed algorithms are regularly used to deploy distributed paradigms in smart grids. Computations are executed without any centralized server even in a limited bandwidth network. However, there are still some drawbacks in the proof of convergence as well as a lack of programming abstraction. In this paper, we propose a new approach to the distributed paradigm by using distributed programming in a data manipulation language called Smartlog. A distributed programming methodology will automatically divide a centralized Smartlog program into multiple rules while ensuring the correct transformation of the centralized computations. An application of voltage control in a distribution grid with high penetration of PV systems is used as an illustration of this approach. Two implementations are compared: centralized programming and distributed programming. Both are deployed in a real-time simulation with OPAL-RT and a network of Raspberry Pis. The response time of each implementation is analyzed to evaluate their respective performance, showing that the Smartlog language is particularly appropriate to smart grids in terms of compacity and simplicity.**

***Index Terms*—Distributed programming, distributed database, over-voltage regulation, smart grids, Smartlog.**

I. Introduction

Many applications of control and management in historical grids present algorithms that can be easily deployed in a centralized manner. However, along with the development of smart grids, a centralized implementation can show shortcomings, such as high computation and communication costs, low fault tolerance capability, etc. [1]. Besides, with the current development of smart grids infrastructures, computing units are almost dispersed in the network. These available resources can participate in the management of the power system. With the size of this system increasing, the distributed implementation of algorithms becomes a viable alternative to centralized ones. Indeed, it can deal with more resilience issues while ensuring a proper replacement the conventional centralized controls [2].

However, there are still some drawbacks to distributed implementations that restrain their practical deployment. Firstly, there is a lack of programming abstraction [3]. For instance, consensus algorithms [4] (e.g., Metropolis, Finite-time Average Consensus algorithms, Maximum Degree Weight) present a convergence that depends on the network configuration. Secondly, there is a lack of convergence proof when the system scales up as well as a low convergence speed. For instance, the ADMM algorithm can be used for the distributed optimal power flow problem [5] but the convergence speed goes up to several thousand iterations for a small grid. The ALADDIN Algorithm is a possibility to improve the convergence rate of the ADMM method [6], however not proven in the case where the number of nodes increases. The slow convergence speed and the neglected communication delay lead to slow control and management of the power system. In a context where the speed of variation of the electrical quantities increases due to renewable energy integration, some bounds of the system stability could be more frequently violated and the power quality could also decrease. Moreover, the control and the management of smart grids in real-time require to deal immediately with changes in the power system.

For those reasons, in this paper, another approach is considered, that combine the advantages of both the centralized and decentralized implementations. This allows taking advantage of two paradigms while ensuring the response to any change of system, as fast as possible. Most of the imperative languages such as Java or Python support distributed programming in centralized memories. Nevertheless, that is not appropriate for a distributed paradigm in smart grids. Thus, in this paper, we approach the distributed programming with a declarative language, called *Smartlog* [7].

In [7], each smart device plays the role of a node in the IP network of the smart grids. Each node is a rule-based system with a local database and can operate simultaneously as a client and a server. Based on that architecture, a high-level programming of the data manipulation language Smartlog is developed to support the proposed distributed programming. A methodology, proposed in this paper, helps splitting a centralized program into distributed rules which are executed in multiple nodes of the grid.

The authors would like to thank the French Embassy in Vietnam and the Foundation Grenoble INP for funding the PhD leading to the presented results.

*Institute of Engineering Univ. Grenoble Alpes.

II. Distributed rule-based system in smart grids

A. Network architecture

Smart grids are considered as heterogeneous. It is composed of a lot of smart devices which are in charge of collecting data and establishing communications among them. Each smart device can be in charge of computations and communications at each node of the grid. Fig. 1 presents their typical architecture.

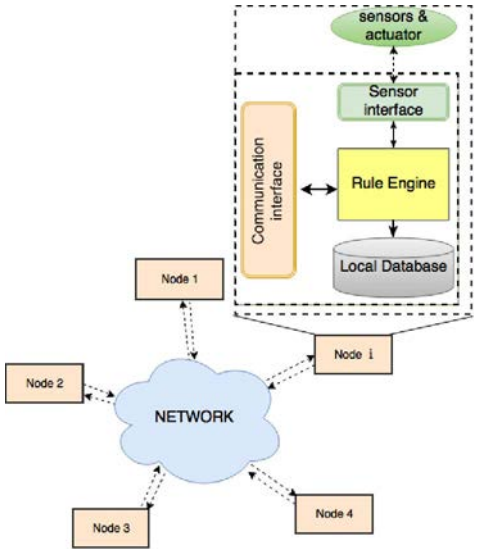

Fig. 1. The architecture of each node in the considered smart grid.

There are four main components in the architecture of each node:

- The `Local database` : stores the node's information in the infrastructure of the grid, such as parameters, the measured data which are collected from local sensors or directly from the network.
- The `Sensor interface`: is set up to collect data from sensors, to store these data into the local database and to transfer the output values to the regulator in order to control the active electrical components.
- The `Rule engine`: is the most critical part of the architecture because it supplies an environment to execute the declarative programs. In our implementation, the PostgreSQL trigger is used as a rule engine.
- The `Communication interface`: is in charge of the interaction between nodes over the communication network. Received data, after unpacking, will be stored in the local database. The communication is developed based on the TCP/IP protocol and the transferred data are in the form of a JSON Object.

Each participating node in the grid possesses the same architecture. This constructs an IP network in which each node can operate simultaneously as both *client* and *server*. That increases the robustness and reduces the damage when there is a single point of failure in the system [8].

The distributed programming is developed based on the available infrastructure of information network such as WiFi, 3G, 4G and is deploy in the test-cases of an isolated microgrid. Thereby, we assume that each node in the network is fully capable of communicating with all others.

B. The Smartlog language

1) The structure of a Smartlog program: The structure of a Smartlog program is presented in Listing 1 [7]:

```
Program(NameOfProgram){
Data_types{//define the schematic of stored data
}
Initial_data{//declare the data initialization
}
Module(data_type 1){//rules
}
Module(data_type 2){//rules
}...}
```

Listing 1. The structure of a smartlog program.

2) Rule syntax: The `Rule` is in charge of defining the action which is executed in each node. The syntax of a rule in Smartlog is the same as in other logic programming languages:

Head : Body [terminator]

Where *Body* = $\mathbf{B_1}, ..., \mathbf{B_n}$, the body part of the rule, is a conjunction of terms. Each term in the body part B_i can be a relational atom $R(r_1, r_2, ..., r_n)$, a condition term or an assignment term.

3) Operators for the head part: The `Head` part holds the variables with assigned values in the body part and defines the execution of the rule. If all the terms of the body are approved, the execution of the head part (H) is launched.

By default, the execution of the head part is a *storing mode*, which means that the results will be stored in its local database.

$$H \text{ :- } B_1, B_2, ..., B_n$$

The execution in *sending-mode* is expressed with:

$$\text{^ H :- } B_1, B_2, ..., B_n$$

In this case, a destination's address should be marked with the symbol @ in front of the *address variable*.

C. How Smartlog supports distributed programming

Rules in a Smartlog program are grouped into many modules. Each module defines all actions of the system with the modification of a specific *data_type*. Measured data can trigger calculations in a local database. Meanwhile, immediate data is used to support data sharing. These data, transferred between nodes, are in the form of *data_type*. The data reception allows triggering consecutive calculations in other nodes. For example, consider two modules in two programs as follows:

```
Module(A){
^TmpC(i, v, c) :- A(i, v, c), B(i,@j); }
```

Listing 2. A Module A in node i.

```
Module(TmpC){
C(i, v, c) :- TmpC(i, v, c); }
```

Listing 3. A Module $TmpC$ in node j.

When the rule of node i in Listing 2 is executed, an atom named `TmpC` is sent to j. The module in Listing 3 will continue to perform its actions after receiving `TmpC`. With this mechanism, Smartlog can support fully distributed programming.

III. Rule distribution methodology

In order to perform the same computation as a centralized program, the main problem of distributed programming is to separate rules from the centralized program and allocate them to nodes, when data are distributed on different machines. The methodology of rule distribution (DSLP method) is proposed to tackle this question. The general algorithm of the methodology is summarized in Algorithm 1:

Algorithm 1 Distributed programming methodology (DSLP).

```
1: procedure DISTRIBUTEPROGRAM(program)
2:     Describe the data distribution in the network
3:     for rule in program do
4:         Rewrite the rule according to data fragmentation
5:         for rule in the rewritten rules do
6:             Evaluate all possibility of data communication
7:             Choose the best decision of communication
8:             Distribute the rule based on the best decision
9:     Generate distributed programs
```

The methodology of distributed programming is not presented in detail in this paper, for a reason of space. Instead, we focus on evaluating its applications. A centralized algorithm for a common issue in smart grid: over-voltage regulation.

IV. Application

A. A centralized algorithm in smart grid

The rise of the percentage of renewable energy penetration, especially Photovoltaic (PV) sources in traditional power grids risks power imbalances as well as quality loss of the energy. One of the most critical issues, when this energy integrated into the grid, is over-voltage. Many solutions have proposed to address this problem, and one of the practical approaches is Adaptive Active Power Capping (AAPC) [9]. The principle of the method is presented in Fig. 2.

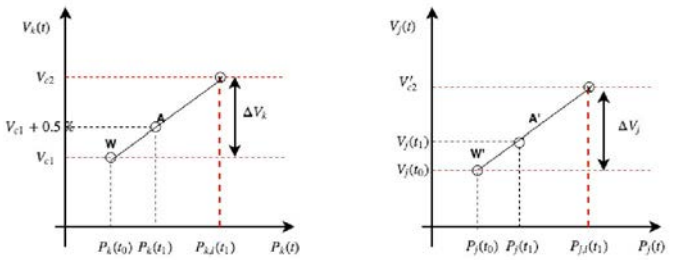

Fig. 2. Principle of Active Power capping method.

V_{C_1} and V_{C_2} are two bounds of the regressive method, with $[V_{C_1} - V_{C_2}] = [1.042 - 1.058](pu)$ [9]. Over-voltages occur in one or more nodes in the grid. The node presenting a voltage value over V_{c1} is called a critical node (CRI). In the AAPC method, computations are performed based on the critical nodes. We use the linear regression method to predict the upper power limitation (PLI) P_l^j of each PV source.

$$P_l^j(t_1) = P^j(t_0) + (V_{c2}^{'} - V^j(t_o))/\xi^j \tag{1}$$

with ξ is a linear coefficient (SLOPE) calculated by the ratio of voltage variation (VVAR) and power variation (PVAR):

$$\xi^j = \frac{V^j(t_1) - V^j(t_0)}{P^j(t_1) - P^j(t_0)} \tag{2}$$

If each PV output power production is below its upper threshold, then the over-voltage in the grid is correclty controlled. P_j^{ref} is the generated power threshold (PREF) of the j^{th} PV and $P_m^j(t_2)$ is the maximum PV output power at time t_2 based on the MPPT control. The upper threshold of the j^{th} node is defined as:

$$P_{ref}^j = min(P_l^j(t_1), P_m^j(t_2)) \tag{3}$$

Each PV node has the same responsibility to participate in maintaining the acceptable grid operation. Thus, the power supplied or curtailed of each PV has to be based on a fair sharing. The power of each PV is then its limited power (PLI) or maximal power (PMA) as follows:

$$P_a^j = \frac{\sum_{j=1}^{j=n} P_{ref}^j(t_1)}{\sum_{j=1}^{j=n} P_m^j(t_2)} \times P_m^j(t_2) = \eta \times P_m^j(t_2) \tag{4}$$

with η the power curtailment coefficient (PCUR).

B. Setup and Smartlog implementation

The PREDIS distribution grid [10] is used as a test object in our case. This grid comprises 14 nodes, with five distributed sources, three asynchronous machines and static loads. This grid is simulated in MATLAB/SIMULINK and executed in OPAL/RT Real-time simulation. Each Raspberry Pi plays the role of a local computing unit which is installed near a distributed source (e.g., PV).

The description of the data schemes used in this application is presented in Table I.

TABLE I
Data scheme for the AAPC algorithm.

Atom	Scheme description
Measure(i,t,vi,p,pmi)	Measure(ID key, Times, VOLT, POW, PMA)
Slope(i,j,s,dv)	Slope(CRI-ID key, ID key, SLOPE, VVAR)
Warning(i, t)	Warning(ID key, TIMES)
Alert(i, t, co)	Alert(ID key, TIMES, proportion).
WarningMeasure(i,j,v,p)	WarningMeasure(CRI-ID key, ID, VOLT, POW).
Curtail(i, pe)	Curtail(ID key, PCUR)
Actuator(i, pe)	Actuator(ID key, PERFORMANCE).
Plimit(i, j, pli)	Plimit(CRI-ID key, ID, PREF).

The rules are expressed in a centralized Smartlog program as presented in Listing 1.

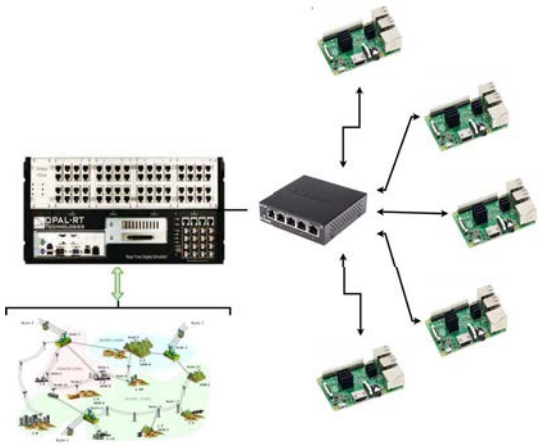

Fig. 3. Architecture of the real-time platform.

```
Module(Measure){
  Warning(i,t) :- Measure(i,t,vi,p,pmi),
       ~WarningMeasure(i,i,_,_), vi>=1.042;
  :- Measure(i,t,vi,p,pmi),
       !WarningMeasure(i,i,_,_), vi <1.042;
  Alert(i,t,co) :- Measure(i,t,vi,p,pmi),
       WarningMeasure(i,i,_,_), vi>=1.047, co :=
       (1.058-vi)/(vi-1.042);}
Module(Warning){
  WarningMeasure (i,j,vj,pj) :- Warning(i,t),
       Measure(j,t,vj,pj,_);}
Module(Alert){
  Slope(i,j,s,dv) :- Alert(i,t,co),
       Measure(j,t,vj,pj,pmj),
       WarningMeasure(i,j,vo,po), pj<>po, s:=
       (vj-vo) /(pj-po), s<>0, dv := (co+1)*(vj-vo);
  Curtail(i,pe) :- Alert(i,t,_), Measure(j,t, _,_,
       pmj), Plimit(i,j,pli), pli := least(pli,pmj),
       prs := sum(pli), pms := sum(pmj), pe:=
       prs/pms;}
Module(Slope){
  Plimit(i,j,pli) :- Slope(i,j,s,dv),
       WarningMeasure(i,j,_,po), pli := po+ dv/s;}
Module(Curtail){
  Actuator(j,pe) :- Curtail(i,pe), Actuator(j,_,_);}
```

Listing 4. Set of rules for the AAPC implementation in Smartlog.

An experiment was performed with three implementations of the same model and algorithms: a centralized java programming, a centralized Smartlog programming, and a distributed Smartlog programming.

1) Data distribution: In the distributed implementation, we suppose that each distributed PV source corresponds to a Raspberry Pi which is in charge of a local computing unit. Five Raspberry Pis were used in the experiment and eight *data_types*. These *data_types* are fragmented horizontally for each Raspberry.

A centralized Smartlog program and its data distribution are the input of the distributed programming methodology. Five distributed programs (corresponding to five Raspberry Pis) are its output.

2) Data sets: The data sets are generated automatically during the real-time simulation. The corresponding data in the simulated grid will be transferred from OPAL-RT to each Raspberry Pi.

3) Scenario: In the voltage control problem, we only consider the change of power injected at each node. In reality, the produced solar power depends mainly on climate and clouds. In the short time of simulation (the experiment is performed during 10 minutes), we assumed the load to be constant, with changes in the PV power. Fig. 4 shows the behaviors of the AAPC method as well as the response time of the implementation. The 7^{th} node of the grid is designed to be the one with the highest voltage value during the simulation.

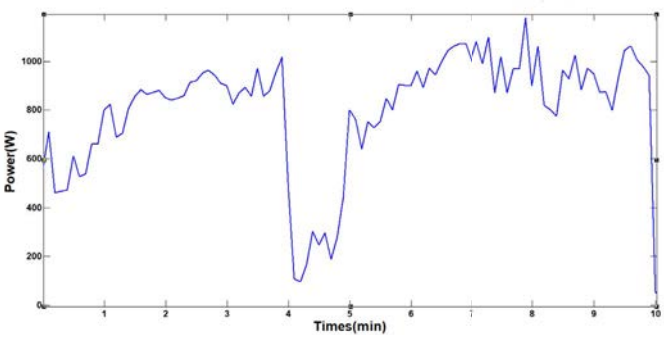

Fig. 4. The PV generation curve over 10 minutes.

V. Result and Discussion

A. The correctness

In Fig. 5, the response of voltage in the 7^{th} node is shown in the two Smartlog implementations (centralized and distributed). The voltage is controlled to not cross the upper bound (1.058 p.u.). The results being identical, the correctness of DSLP method is confirmed.

B. The performances

We use the response time to evaluate and compare the performance of each implementation. The response time should be considered within a given interval.

At the second minute of the simulation, the active power of the PV increases linearly and causes an over-voltage in the grid for the first time. The AAPC method is activated to restrain the percentage of power production. Continuously, when the grid is in over-voltage, the power curtailment inversely follows the rise of active power. That means the curtailment reduces linearly when the power production increases and exceeds the power upper bound.

In practice, the response time is defined as the interval between two consecutive reactions of the PV's actuator when the grid operates in over-voltage. Based on that, we estimate the response time of each deployment in the experiment, presented in Fig. 6.

We call t_i is the average processing time of each rule, t_c the average delay time for each communication and N is the number of computing units of the grid. The response time at a node is estimated as $T_{res} = T_{comp} + T_{comm}$. With R_j the

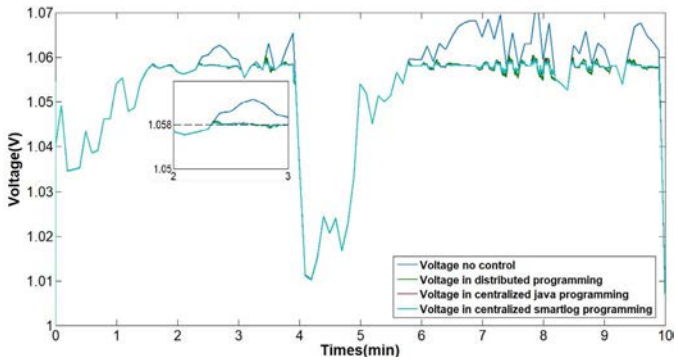

Fig. 5. Voltage response at 2^{th} PV node with three implementations of the AAPC algorithm.

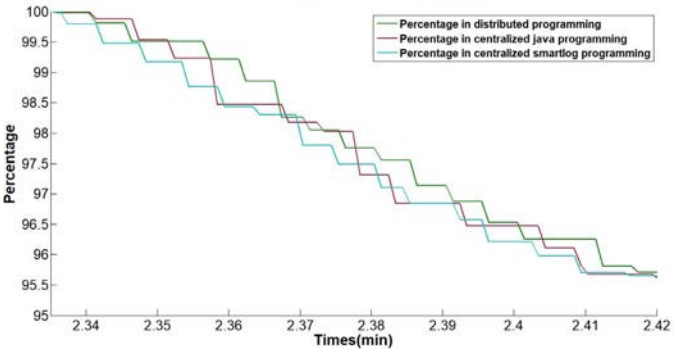

Fig. 6. Output of voltage regulation for a given time interval.

number of executed rules in the j^{th} node, the analysis of the response time in this experiment is shown in Table II.

TABLE II
RESPONSE TIME FOR THE TWO SMARTLOG IMPLEMENTATIONS.

Programming	T_{comp}	T_{comm}
Centralized	$\sum_{j\in N}\sum_{i\in R_j} t_i$	0
Distributed	$\max(\sum_{i\in R_2} t_i, \sum_{j\in R_3} t_j)$	Nt_c

Although the computation load is shared over the network, the response time still depends on the communication time that depends on the characteristic of the grid. If the communication time is significant, then the response time with a distributed programming may be longer than the one of a centralized programming.

In the experiment, the statistic of the average response times for the three implementations are shown in Fig. 7.

The response time of the distributed programming is better than the response time of the centralized programming: $\max(\sum_{i\in R_2} t_i, \sum_{j\in R_3} t_j) + Nt_c < \sum_{j\in N}\sum_{i\in R_j} t_i$. In the real-time platform, the communication time among Raspberry Pis is indeed small. Besides, the query time in a distributed database is faster than in a centralized database. Moreover, the computing units operate in parallel and share the computing load, which apparently reduces the response time.

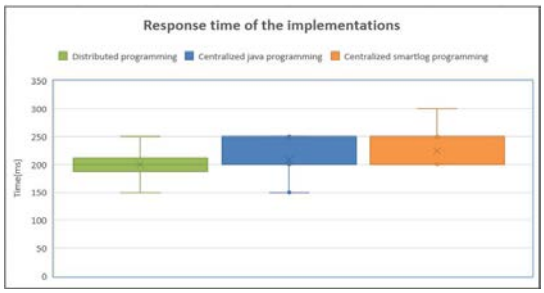

Fig. 7. The average response time of three implementations of the AAPC algorithm.

VI. CONCLUSION

In this paper, we proposed an approach to deploy a distributed implementation for smart grid applications. This approach not only overcomes the limitation of the distributed algorithms such as the lack of programming abstraction and the proof of convergence but also provides a simple implementation and a better response time than a centralized implementation.

This is illustrated with an application of over-voltage regulation in a real-time OPAL-RT simulation and a network of Raspberry Pis, with the simple procedure of "programming in centralized manner and executing in a distributed manner". This approach can deal with the scalability of the number of nodes in the network, which validity is the the prospective research of this work.

REFERENCES

[1] N. Hatziargyriou, *Microgrids: Architectures and Control*. Elsevier, 01 2014.
[2] Q. Shafiee, J. C. Vasquez, and J. M. Guerrero, "Distributed secondary control for islanded MicroGrids - A networked control systems approach," in *38th Annual Conference on IEEE Industrial Electronics Society (IECON)*, Oct. 2012, pp. 5637–5642.
[3] P. J. Marron and D. Minder, *Embedded WiSeNts research roadmap*. Logos-Verlag, 2006.
[4] Y. Xu and Z. Li, "Distributed optimal resource management based on the consensus algorithm in a microgrid," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 4, pp. 2584–2592, April 2015.
[5] S. Magnússon, P. C. Weeraddana, and C. Fischione, "A distributed approach for the optimal power-flow problem based on ADMM and sequential convex approximations," *IEEE Transactions on Control of Network Systems*, vol. 2, no. 3, pp. 238–253, 2015.
[6] A. Engelmann, T. Mühlpfordt, Y. Jiang, B. Houska, and T. Faulwasser, "Distributed ac optimal power flow using ALADIN," *FAC-PapersOnLine*, vol. 50, no. 1, pp. 5536–5541, 2017.
[7] T.-T.-Q. Nguyen, C. Bobineau, V. Debusschere, Q.-H. Giap, and N. Hadjsaid, "Using declarative programming for network data management in smart grids," in *22th International Database Engineering & Applications Symposium*, ser. IDEAS 2018. New York, NY, USA: ACM, 2018, pp. 292–296.
[8] J. Li, "On peer-to-peer (p2p) content delivery," *Peer-to-Peer Networking and Applications*, vol. 1, no. 1, pp. 45–63, 2008.
[9] S. Alyami, Y. Wang, C. Wang, J. Zhao, and B. Zhao, "Adaptive real power capping method for fair overvoltage regulation of distribution networks with high penetration of pv systems," *IEEE Transactions on Smart Grid*, vol. 5, no. 6, pp. 2729–2738, 2014.
[10] M. C. Alvarez-Herault, A. Labonne, S. TourÃl', T. Braconnier, V. Debusschere, R. Caire, and N. Hadjsaid, "An original smart-grids test bed to teach feeder automation functions in a distribution grid," *IEEE Transactions on Power Systems*, vol. 33, no. 1, pp. 373–385, Jan 2018.