This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Subramanya, Rakshith; Sierla, Seppo; Yli-Ojanperä, Matti; Makkonen, Henri; Pourakbari Kasmaei, Mahdi; Vyatkin, Valeriy
Interfacing Third Party Cloud Services to a Virtual Power Plant

Published in:
2021 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)

DOI:
[10.1109/ISGTEurope52324.2021.9640200](https://doi.org/10.1109/ISGTEurope52324.2021.9640200)

Published: 01/01/2021

Document Version
Peer reviewed version

Please cite the original version:
Subramanya, R., Sierla, S., Yli-Ojanperä, M., Makkonen, H., Pourakbari Kasmaei, M., & Vyatkin, V. (2021). Interfacing Third Party Cloud Services to a Virtual Power Plant. In *2021 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe)* IEEE. <https://doi.org/10.1109/ISGTEurope52324.2021.9640200>

© 2021 IEEE. This is the author's version of an article that has been published by IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Interfacing Third Party Cloud Services to a Virtual Power Plant

Rakshith Subramanya
*Department of Electrical Engineering
and Automation
Aalto University
Espoo, Finland*
rakshith.subramanya@aalto.fi

Seppo Sierla
*Department of Electrical Engineering
and Automation
Aalto University
Espoo, Finland*
seppo.sierla@aalto.fi

Matti Yli-Ojanperä
*Department of Electrical Engineering
and Automation
Aalto University
Espoo, Finland*
matti.yli-ojanpera@aalto.fi

Henri Makkonen
*Siemens Osakeyhtiö
Espoo, Finland*
henri.makkonen@siemens.com

Mahdi Pourakbari-Kasmaei
*Department of Electrical Engineering
and Automation
Aalto University
Espoo, Finland*
Mahdi.Pourakbari@aalto.fi

Valeriy Vyatkin
*Dep. of Electrical Engineering and
Automation, Dep. of Computer Science,
Electrical, and Space Engineering and
Computer Technologies Laboratory
Aalto University, Luleå University of
Technology and ITMO University
Espoo, Finland, Luleå, Sweden and St.
Petersburg, Russia*
valeriy.vyatkin@aalto.fi

Abstract— Cloud computing is a megatrend in industry digitalization. The virtual power plant (VPP) is especially suited for cloud deployment since it can manage energy resources that are not co-located. However, there is a lack of research in VPP cloudification, mainly related to multi-tenancy, a key technology behind the efficiency benefits of cloud computing. Cloudification is complicated as power systems and distributed energy resources frequently employ interfaces and protocols from an earlier era. In particular, IEC 60870-5-104 is a well-established standard for telecontrol in electrical engineering and power system automation applications. This paper proposes a new perspective on VPP interoperability via the application of cloud computing. IEC 60870-5-104 enabled systems are integrated into third-party systems in the cloud. Examples of such third-party systems are the Internet of Things-enabled distributed energy resources or electricity market information systems. This paper proposes a new problem definition of VPP-connected systems through establishing the said third party systems as tenants in a multi-tenant architecture. To define a general industry standards-based system architecture, both the software design methods and cloud computing architectural concepts are applied. An implementation is presented that is ready to integrate into systems in the field.

Keywords— *virtual power plant, IEC 104, REST API, software as a service, cloud computing, smart grid*

I. INTRODUCTION

Virtual power plants (VPP) have emerged as a key operating concept in smart grids [1]. VPPs manage distributed energy resources such as building automation loads, battery storage, electric vehicle (EV) chargers, and photovoltaic units, so an information architecture must interface with the control systems controlling such resources and interface the VPP to market information systems [2]. A recent trend is the cloudification of such control and information systems, motivating research on cloud-based VPP architectures [3]. This results in the following research problem: how can third-party cloud services be interfaced with a VPP?

Two potential interfacing technologies in the smart grid domain such as IEC 60870-5-104, commonly referred to as IEC 104 [4], and web services [5], were applied for the specific purpose of interfacing a VPP to third party systems. However, these works did not address the possibility of the systems being available through the cloud. Despite the challenges, business realities are forcing companies to adopt the SaaS (Software as a Service) model [6], meaning that the software is hosted in the cloud and licensed on a subscription basis. More recently, the SaaS paradigm has started gaining traction in the smart grid domain, necessitating the redesign of information system architectures for the purpose of cloud deployment [7]. This paper aims to design and validate a SaaS architecture for interfacing third-party cloud services to a VPP, exploiting IEC 104 and REST (Representational State Transfer) APIs (Application Programming Interfaces).

The adoption of the SaaS paradigm in the smart grid domain is being driven by big data applications [8], the flexibility and extensibility of web deployed applications [9], and the desire to minimize end-users' need for IT infrastructure [10]. Cloud and SaaS technology is also emerging in systems that VPPs interface to, including renewable energy management systems (EMS), home EMS, heating, ventilation and air conditioning (HVAC), EV charging [11]–[13], vehicular clouds [14] and vehicular networks [15]. However, there are several research gaps on 1) how a VPP could interface to such systems, despite an emerging body of research on cloud-deployed VPPs [3], [16], and 2) the application of multi-tenancy in this domain, although it is a key technique responsible for much of the cost savings in cloud computing [17].

IEC 60870 part 5 is one of the IEC60870 standards that defines systems used for telecontrol (SCADA) in electrical engineering and power system automation applications. Part 5 provides a communication profile for sending basic telecontrol messages between two systems. IEC 60870-5-101 is the companion standard for basic telecontrol tasks [18]. IEC 60870-5-104, commonly referred to as IEC 104, is the companion standard for IEC 60870-5-101 over TCP/IP [19].

This research was supported by Business Finland grant 7439/31/2018.

Fig. 1. Architecture concept

IEC 104 is commonly used in power systems automation applications [20], and is identified as a key standard supported by legacy systems, needs to be identified as the cloudification of smart grid information systems progresses [21].

Web services have established themselves as a key technology for interfacing Web applications, including cloud-deployed applications, with REST API emerging as the preferred distributed-system framework for developers [22]. Currently, the main application for REST APIs in the smart grid domain is related to communication with simulation packages [23] and accessing smart meter data, but REST API has also been identified as a key interfacing technique in a cloud-based smart grid architecture [24]. The possible use cases for integrating third party SaaS applications to a VPP are varied. Artificial intelligence can perform advanced analytics, and to this end data needs to be retrieved from the VPP and the results of the analytics need to be returned to the VPP [25], [26]. The need to interface such systems to a VPP is explicitly identified and [27] uses REST API to implement such an integration with a commercial VPP. Various IoT enabled distributed energy resources, such as intelligent outdoor [28], [29] and indoor lighting network [13], could be connected to a VPP for the purpose of exploiting them on various electricity markets. The IoT architecture design of a VPP managing distributed energy resources is explicitly discussed in [30], which takes into account the need to avoid hard real time control of the resources over the public Internet.

In summary, there is a trend for the cloud and SaaS technology adoption in the smart grid domain in general and especially in the systems that a VPP interfaces to, referred to as third-party systems in this article. Although IEC 104 and REST API are potential interfacing technologies, there is a lack of research about how these interfacing technologies could be used in a cloud-based architecture that allows a VPP to interface to relevant third-party systems in the cloud.

Therefore, the main contributions of this paper are as follows:

- A new perspective on VPP interoperability is proposed through the application of cloud computing.
- IEC 104 is used to integrate legacy systems into cloud-deployed systems.
- In order to exploit the full potential of cloud computing, multi-tenancy is a key technology. A new problem definition of VPP-connected systems is formulated by defining such systems as tenants.
- Software design methods and cloud computing architectural concepts are applied to define an industry

standards-based system architecture that is general in the sense that it does not limit itself to any specific type of third-party system. An implementation is presented that is ready to integrate into systems in the field.

II. PROPOSED ARCHITECTURE

As stated in section I, this paper designs and validated a SaaS architecture for interfacing third-party cloud services to a VPP, exploiting IEC 104 and REST APIs. Fig. 1 shows the proposed architecture, in which VPPs communicating with the IEC 104 protocol are able to access multiple SaaS clients via a single instance of a REST API server. These SaaS clients are connected to assets like photovoltaic panels and electric vehicle charging stations via the REST API. The same REST API server is connected to multiple IEC 104 servers, and each IEC 104 server is connected to the IEC 104 client of a VPP. Each IEC 104 server and client pair communicates through IEC 104 protocol.

A key technique for realizing the benefits of cloud technology is multitenancy. Multi-tenancy means that a single instance of the software serves multiple clients or tenants. In our case, a single instance of the REST API server serves multiple tenants. One IEC104 client, one IEC104 server, and one REST API client make up one tenant. The tenant has specific data, configuration, and user management, but all of the tenants use the same REST API server instance. In a SaaS architecture for interfacing third-party cloud services to a VPP, multi-tenancy will enable fast scaling, efficient resource distribution, and easier third-party integration. However, tenant isolation or tenant partitioning is one of the SaaS multi-tenant data architecture's primary aspects for data isolation. In the current context, messages sent from an IEC 104 server should only reach its corresponding SaaS client. For instance, in Fig. 1, control data transmitted from IEC 104 server 1 should only go to SaaS client 1.

The IEC 104 frame format is defined as in Table I. The IEC 104 features APCI (Application Protocol Control Information), ASDU (Application Service Data Unit), and APDU (Application Protocol Data Unit) for classifying the end station and its segments. For example, in the case of a VPP, the ASDU contains a Cause of Transmission (COT) data unit identifier, which indicates the cause of the data transmission from a VPP to be cyclic or spontaneous. The Information Object Address (IOA) field provides the information object element's address and is transmitted in each ASDU, including information about that object's input or output. Different IOA mappings have been incorporated to

monitor and control the ASDUs, and this is explained further in Section 5.

tenant is passed in the URL of the REST API call and it is used to specify the tenant to the REST API, as shown in Fig. 2.

Fig 2. UML Sequence diagram specifying interactions between the entities in Fig. 1

Fig. 2 is a UML (Unified Modeling Language) sequence diagram that specifies the proposed protocol between the entities in Fig. 1. When the IEC 104 server is started, it waits for an interrogation command from the IEC 104 client. Upon receiving it, the server sends GET a request to the RestAPI to retrieve all monitored ASDUs. If the request is successful, it sends a confirmation of the interrogation command to the client. After that, the server enters a loop for sending all the received monitored ASDUs to the client one at a time. After each successfully sent ASDU, the server sends a POST confirmation to the RestAPI. After that, the server enters a loop, in which it sends GET requests to retrieve unconfirmed monitor ASDUs. It sends all the received unconfirmed ASDUs to the client one at a time and after each successfully sent ASDU it will POST confirmation to the RestAPI. The server waits 1 second before sending the next GET request. At the same time, the IEC 104 server listens to incoming commands from the IEC 104 client. Upon receiving a command, the IEC 104 will POST that command to the RestAPI, and after successful response, it will send confirmation of the ASDU to the IEC 104 client. If the connection to the client is interrupted, the server will exit the loop and wait for a new connection and interrogation command. If the connection to the RestAPI is interrupted, the server will close the connection to the client and will refuse new client connections until the connection to the RestAPI is OK. This architecture supports multi-tenancy via multiple IEC 104 server connections.

Fig. 3 shows the interaction between the entities within a multi-tenant architecture implemented based on Fig. 1. In this example, we have considered three IEC 104 clients, three IEC 104 servers, one instance of the REST API server, and three REST API clients. The communication flow between the entities is the same as shown in Fig. 2, but we have created three tenants using the same REST API. The REST API stores each tenant's data into a dedicated directory. The name of the

Each pair of REST API client and IEC 104 server belong to the same tenant. For example, REST API client 1 and IEC 104 server 1 are parts of tenant 1. In the UML activity diagram in Fig. 3, the process of initiating a Single command is shown.

TABLE I. IEC 104 FRAME FORMAT

byte/ bit	7	6	5	4	3	2	1	0				
0	Start byte (0x68)									APCI		APDU
1	Length of the APDU (max. 253)											
2	Control field 1											
3	Control field 2											
4	Control field 3											
5	Control field 4											
6	Type identification									ASDU		
7	SQ	Number of objects										
8	T	P/N	Cause of transmission (COT)									
9	Originator address (ORG)											
10	ASDU address fields											
11	(2 octets)											
12	Information object address fields (IOA) (3 octets)											
13												
14												
15	Object Information											
..												
...												
n-1												
n												

For example, when IEC 104 client 1 initiates such a command SCcli1, it sends the command to REST API client 1 via IEC 104 server 1 and the REST API server. Notably, the REST API separates the messages sent from several tenants and keeps them in their respective channels. For a SaaS model, this is important for achieving the cost savings of multi-tenancy while ensuring the secure management of each tenant's data.

III. IMPLEMENTATION

The REST API is designed to store ASDUs (Application Service Data Units) in JSON (JavaScript Object Notation)

Fig. 3. UML activity diagram specifying multi-tenant interactions

files. The file locations are tenant (IEC104-Server) specific. The REST API is protected using Basic Authentication. For each user, there is a specified list of tenants that the user is authorized to access. Each file contains all the ASDUs with the specific common address. The ASDUs are stored in a dictionary where the key is the IOA (Information object address) and value is an object that has:

- The ASDU
- Direction, monitor or control.
- Confirmed, a Boolean status whether the ASDU has been read and transmitted forward.
- timestamp_added, timestamp when the ASDU was added.
- timestamp_confirmed when the ASDU was confirmed.

This REST API the 'RestAPI' in the center of Fig. 1. It is used together with IEC104-servers and SaaS-clients as illustrated in Fig. 1. The IEC104 servers (1) receive control values from IEC104-client and POST those to the API and (2) GET monitor values from the API and send those to IEC104-client. The SaaS-clients (1) GET control values from the API and transmit those to the SaaS and (2) receive monitor values from the SaaS and POST those to the API. The API can be used to integrate multiple cloud services to one or many IEC104-clients.

The implementation supports the IEC 104 ASDUs listed in Table II. Below is an example JSON for the ASDU M_ME_NB_1. The ASDU structure follows the IEC 104 frame format defined in Table I.

TABLE II. ASDUS (APPLICATION SERVICE DATA UNITS)

Type	Dec	Description
M_ME_NB_1	11	Measured value scaled value
M_SP_TB_1	30	Single-point information with time tag CP56Time2a
M_ME_TE_1	35	Measured value, scaled value with time tag CP56Time2a
M_ME_TF_1	36	Measured value, short floating point value with time tag CP56Time2a
C_SC_NA_1	45	Single command
C_SE_NB_1	49	Set point command, scaled value
C_SE_NC_1	50	Set point command, short floating point value
C_SC_TA_1	58	Single command with time tag CP56Time2a

C_SE_TB_1	62	Set point command, scaled value with time tag P56Time2a
-----------	----	---

```
{
  "cause_of_transmission": "SPONTANEOUS",
  "common_address": 1,
  "information_objects": [
    {
      "elements": [
        {
          "quality": {
            "blocked": false, "invalid": false,
            "notTopical": false, "overflow": false, "substituted": false
          },
          "value": 10
        }
      ],
      "ioa": 1
    }
  ],
  "is_sequence_of_elements": false, "negative_confirm": false,
  "originator_address": 0, "test": false, "type_id": "M_ME_NB_1"
}
```

The REST API is configured with the following information.

- The hosts IP address and the port.
 - tenants: file_path, path to where the JSON files will be saved (The directory for each tenant must be different, or the files will be overwritten by other tenants.)
- Users: (Username, Password, List of tenants the user has access to)

The 'IEC 104 Server' in Fig. 1 is designed to work as an adapter between 'RestAPI' and 'IEC 104 Client'. It is implemented using j60870 library.

The proposed architecture in Fig. 1 is tested in multi-tenant mode. Three instances of REST API clients, one REST API server, three IEC 104 servers, and three IEC 104 client instances are created. The REST API server is designed using the uvicon ASGI server and the REST API client is designed using Python modules. IEC 104 server uses the j60870 library, and IEC 104 client is a standalone executable program. Except for the IEC 104 clients, all other instances are running within Docker containers. A Container is software that packages code and all its dependencies, so it runs quickly and reliably from one computing environment to another. A Docker container is a package of software that includes everything needed to run an application. Docker Compose is used for

defining and running multi-container Docker instances. From the SaaS perspective, this containerization is crucial for the scalability and security perspective.

command for scale value, it sends the new control ASDU to the REST API as a POST request. Additionally, the IEC 104 server sends activation confirmation and activation

The screenshot shows a simulator window for IEC 104 clients. On the left, a tree view shows three client instances: IEC104_CLIENT_1, IEC104_CLIENT_2, and IEC104_CLIENT_3, each with its own Data_Objects and Traffic logs. The main area contains a 'Start Communication' button, a 'Stop Communication' button, and a 'Server Status - Connected' indicator. Below these are three tables, each representing a client's data. Each table has columns for S.No, Server IP Address, Port, Common Address, Event Report Type Id, IOA, Value, Quality Bits, Time Stamp, and IEC870 COT Cause.

S.No	Server IP Address	Port	Common Address	Event Report Type Id	IOA	Value	Quality Bits	Time Stamp	IEC870 COT Cause
1	10.0.2.2	5001	1	M_SP_TB_1	100	0	GD	0: 0: 0: 0 - 1/1/2020 report	SPONT
2	10.0.2.2	5001	1	M_ME_NB_1	200	0	GD	9:39: 3:774 - 12/ 4/2021 assu	SPONT
3	10.0.2.2	5001	1	M_ME_TF_1	300	0.000	GD	0: 0: 0: 0 - 1/1/2020 report	SPONT
4	10.0.2.2	5001	1	M_ME_TE_1	400	0	GD	0: 0: 0: 0 - 1/1/2020 report	SPONT

S.No	Server IP Address	Port	Common Address	Event Report Type Id	IOA	Value	Quality Bits	Time Stamp	IEC870 COT Cause
1	10.0.2.2	5002	1	M_SP_TB_1	100	0	GD	0: 0: 0: 0 - 1/1/2020 report	SPONT
2	10.0.2.2	5002	1	M_ME_NB_1	200	0	GD	9:39:25:797 - 12/ 4/2021 assu	SPONT
3	10.0.2.2	5002	1	M_ME_TF_1	300	0.000	GD	0: 0: 0: 0 - 1/1/2020 report	SPONT
4	10.0.2.2	5002	1	M_ME_TE_1	400	0	GD	0: 0: 0: 0 - 1/1/2020 report	SPONT

S.No	Server IP Address	Port	Common Address	Event Report Type Id	IOA	Value	Quality Bits	Time Stamp	IEC870 COT Cause
1	10.0.2.2	5003	1	M_SP_TB_1	100	0	GD	0: 0: 0: 0 - 1/1/2020 report	SPONT
2	10.0.2.2	5003	1	M_ME_NB_1	200	0	GD	9:39:27: 69 - 12/ 4/2021 assu	SPONT
3	10.0.2.2	5003	1	M_ME_TF_1	300	0.000	GD	0: 0: 0: 0 - 1/1/2020 report	SPONT
4	10.0.2.2	5003	1	M_ME_TE_1	400	0	GD	0: 0: 0: 0 - 1/1/2020 report	SPONT

Fig. 4. IEC 104 client window

Scalability, quick setup, and deployment are some significant advantages of using SaaS architecture that could be easily achieved with Docker. SaaS applications are already installed with their dependencies and are usually configured in the cloud. So, adding a new tenant should be faster. SaaS is easy to scale, that is, depending on the number of tenants, the instances could be increased or decreased quickly. As per the need, the docker instances could be scaled up or down horizontally (by replicating the instances). With this setup, connecting a new energy resource via a SaaS client or a VPP client is simple. For instance, in this multi-tenant model, the communication starts once the IEC 104 client 1 sends the commands to the IEC 104 server 1. This message will then be sent to the corresponding REST API client 1, a part of SaaS, via the REST API server. To simulate the IEC 104 client messages from a VPP, an IEC 104 client simulator is used, and three IEC 104 client instances are created within the simulator.

IV. RESULTS

The REST API server and three instances of IEC 104 servers are started, and the IEC 104 clients are started via the IEC 104 client simulator. The IP address and port number of the three IEC 104 servers are loaded to the client simulator to establish the connection with the IEC 104 servers. Once the communication between the IEC 104 servers and clients is established, the REST API server starts polling the data. Now we can send the point commands from the IEC 104 simulator. Our REST API client supports Set Point Scale, Set Point Float and Single Command. To monitor and control the ASDUs, IOA mappings have been incorporated as shown in Table 3. In the REST API client, we have defined four monitor ASDUs, one for each supported monitor ASDU type. For the purpose of this test, the logic of the REST API client is configured so that commands to *IOA 1100* set the value of *IOA 100*. The commands to *IOA 1200*, *IOA 1300*, and *IOA 1400* set the values of internal variables that are used as set points for *IOA 200*, *IOA 300*, and *IOA 400*, respectively. If the value of *IOA 100* is true, the REST API client will change the values of *IOA 200-400* towards the set point assigned as shown in *ASDU Control* in Table 3, at a rate of one per second.

First, the Set point for scale value is set from the IEC 104 Client, and once the IEC 104 server receives the set point

termination messages back to the IEC 104 client. Like Set Point for Scale value is set, the Set Point for Float is also set from the IEC 104 client. However, no data is showing in the IEC 104 client yet as the REST API client is not running.

TABLE III. IOA MAPPINGS

ASDU Monitor		ASDU Control	
IOA 100	to	IOA 1100 to C_SC_NA_1	(This changes value of IOA 100)
IOA 200	to	IOA 1200 to C_SE_NB_1	(Set point for IOA 200)
IOA 300	to	IOA 1300 to C_SE_NC_1	(Set point for IOA 300)
IOA 400	to	IOA 1400 to C_SE_TB_1	(Set point for IOA 400)

Once the REST API client has started, it sends the ASDU's to the REST API server via the POST method. The IEC 104 server receives the ASDUs from the REST API server via the GET method and sends the ASDUs to the IEC 104 client and the data can now be seen in the IEC 104 client. The REST API client will poll for new commands from the REST API server, and the REST API client would push any new data values to the IEC 104 client.

The communication log of the REST API server is shown below. The logs show POST request to the REST API server from one of the IEC 104 server. Here, request's *ioa* value is 200, and *common_address* is 1 as shown in Fig. 2.

```
restapi_1 | 2021-04-12 09:39:03 [INFO] - "POST /tenant/tenant1/asdu/1/200/confirm HTTP/1.1" 200
```

The log for one of the REST API clients is as follows. REST API client and REST API server exchanges control and monitor data, as shown in Fig. 3. The REST API client receives the data via the REST API server, which is of IEC 104 format shown in Table 1. For example, the ASDU COT identifier value is SPONTANEOUS in all the REST API client logs below. Other identifiers such as Originator address, IOA, and object information can also be seen in the response.

REST-API Client1 log:

```
2021-04-12 09:39:03 [DEBUG] __main__: response.json:
{ "type_id": "M_ME_NB_1", "is_sequence_of_elements":
false,
"cause_of_transmission": "SPONTANEOUS", "test": false,
```

```
"negative_confirm": false, "originator_address": 0,
"common_address": 1, "information_objects":
[{"ioa": 200, "elements":
  [{"quality":
    {"blocked": false, "invalid": false, "notTopical": false,
    "overflow": false, "substituted": false
    }, "value": 0 } ] ] }
```

The events captured at IEC 104 server shows sending confirmation to the REST API server and sending the ASDUs as explained in Fig. 2.

```
iec104-server1_1 | Apr 12, 2021 9:39:03 AM
org.aalto.IEC104.IEC104Server confirmASdu
iec104-server1_1 | INFO: Sending confirmation of sent
monitor ASDU to RESTAPI
```

```
iec104-server1_1 | Apr 12, 2021 9:39:03 AM
org.aalto.IEC104.IEC104Server sendASdu
iec104-server1_1 | INFO: Sending ASDU
```

Now the IEC 104 clients capture the values from their respective IEC 104 servers and displays, see Fig. 4. Once the *Single command* is triggered from the IEC client 1, The IEC 104 clients show the updated ASDU values.

V. CONCLUSION

This article has proposed a new perspective to VPP interoperability through the application of cloud computing. IEC 104 is used to integrate legacy systems. An IEC 104 client, an IEC 104 server and a REST API client form a tenant. Multi-tenancy has been exploited to serve multiple tenants with a single REST API server instance, effectively exploiting the computational efficiency benefits of cloud computing while ensuring the secure treatment of the data of each tenant. The design has been specified in UML. An implementation with three tenants has been presented. Test data has been used in this implementation, which is ready to integrate into systems in the field in further work.

References

- [1] C. Wietfeld, A. A. Cardenas, H. H. Chen, P. Popovski, and V. W. S. Wong, "Smart Grids," *IEEE Wireless Communications*, vol. 24, no. 2, Institute of Electrical and Electronics Engineers Inc., pp. 8–9, Apr. 01, 2017, doi: 10.1109/MWC.2017.7909091.
- [2] L. Hernandez *et al.*, "A multi-agent system architecture for smart grid management and forecasting of energy demand in virtual power plants," *IEEE Commun. Mag.*, vol. 51, no. 1, 2013, doi: 10.1109/MCOM.2013.6400446.
- [3] K. Yang, S. Zhang, Y. Li, and Y. Sun, "Edge-cloud collaboration architecture of virtual power plant for large-scale EV integration," 2020, doi: 10.1109/EI250167.2020.9347131.
- [4] M. Kolenc, N. Suljanovic, P. Nemcek, and M. Zajc, "Monitoring communication QoS parameters of distributed energy resources," 2016, doi: 10.1109/ENERGYCON.2016.7513900.
- [5] P. B. Andersen, B. Poulsen, C. Træholt, and J. Østergaard, "Using service oriented architecture in a generic virtual power plant," 2009, doi: 10.1109/ITNG.2009.192.
- [6] D. S. Linthicum, "A Guide to Cloud-Enabling Your Software," *IEEE Cloud Comput.*, vol. 3, no. 2, 2016, doi: 10.1109/MCC.2016.46.
- [7] N. Daleckovic, S. Vukmirovic, I. Kovacevic, and J. Stankovski, "Providing flexible software as a service for smart grid by relying on big data platforms," 2017, doi: 10.1109/ISNCC.2017.8072030.
- [8] P. Scarabaggio, S. Grammatico, R. Carli, and M. Dotoli, "Distributed Demand Side Management With Stochastic Wind Power Forecasting," *IEEE Trans. Control Syst. Technol.*, 2021, doi: 10.1109/TCST.2021.3056751.
- [9] M. Shah, R. Vaishnav, N. Rajagopal, and K. V. Prasad, "Delivering Power system decision support tools over the web," 2010, doi: 10.1109/PES.2010.5590154.
- [10] J. Englander, L. Wenger, Y. Kledzinski, T. Schmid, V. Zeller, and V. Stich, "A methodology to realize energy-related use cases for a cloud-based smart energy management," 2019, doi: 10.1109/SmartCloud.2019.00013.
- [11] K. H. Tseng, C. L. Hsieh, and Y. F. Lien, "The design and implementation of a Cloud Renewable Energy Management System," 2015, doi: 10.1109/ICNSC.2015.7116098.
- [12] A. Ahmed, M. M. Khan, and W. Ahmed, "Cloud based network management and control for building automation," 2017, doi: 10.1109/INMIC.2016.7840123.
- [13] R. Baniya, M. Maksimainen, S. Sierla, C. Pang, C. W. Yang, and V. Vyatkin, "Smart indoor lighting control: Power, illuminance, and colour quality," 2014, doi: 10.1109/ISIE.2014.6864878.
- [14] M. Tajmohammadi, S. M. Mazinani, M. Nikooghadam, and Z. Al-Hamdawee, "LSPP: Lightweight and Secure Payment Protocol for Dynamic Wireless Charging of Electric Vehicles in Vehicular Cloud," *IEEE Access*, vol. 7, 2019, doi: 10.1109/ACCESS.2019.2946241.
- [15] H. A. Khattak, H. Farman, B. Jan, and I. Ud Din, "Toward Integrating Vehicular Clouds with IoT for Smart City Services," *IEEE Netw.*, vol. 33, no. 2, 2019, doi: 10.1109/MNET.2019.1800236.
- [16] M. Kenzhina, I. Kalysh, I. Ukaegbu, and S. K. Nunna, "Virtual Power Plant in Industry 4.0: The Strategic Planning of Emerging Virtual Power Plant in Kazakhstan," in *International Conference on Advanced Communication Technology, ICACT*, 2019, vol. 2019-February, doi: 10.23919/ICAICT.2019.8701989.
- [17] Y. Wang, Q. He, D. Ye, and Y. Yang, "Formulating Criticality-Based Cost-Effective Fault Tolerance Strategies for Multi-Tenant Service-Based Systems," *IEEE Trans. Softw. Eng.*, vol. 44, no. 3, 2018, doi: 10.1109/TSE.2017.2681667.
- [18] R. Czechowski, P. Wicher, and B. Wiecha, "Cyber security in communication of SCADA systems using IEC 61850," 2015, doi: 10.1109/MEPS.2015.7477223.
- [19] V. Skoko, B. Atlagic, and N. Isakov, "Comparative realization of IEC 60870-5 industrial protocol standards," 2015, doi: 10.1109/TELFOR.2014.7034572.
- [20] G. Han, B. Xu, and J. Suonan, "IEC 61850-based feeder terminal unit modeling and mapping to IEC 60870-5-104," *IEEE Trans. Power Deliv.*, vol. 27, no. 4, 2012, doi: 10.1109/TPWRD.2012.2209685.
- [21] A. Garcia-Domínguez, "Enabling SCADA cluster and cloud for smart grid using hierarchical multicast; The PTMF framework," in *Proceedings of the IEEE International Conference on Industrial Technology*, 2015, vol. 2015-June, no. June, doi: 10.1109/ICIT.2015.7125102.
- [22] L. Garber, "The lowly API is ready to step front and center," *Computer (Long Beach, Calif.)*, vol. 46, no. 8, 2013, doi: 10.1109/MC.2013.290.
- [23] K. Anderson, J. Du, A. Narayan, and A. El Gamal, "GridSpice: A distributed simulation platform for the smart grid," *IEEE Trans. Ind. Informatics*, vol. 10, no. 4, 2014, doi: 10.1109/TII.2014.2332115.
- [24] A. Meloni and L. Atzori, "A cloud-based and RESTful Internet of Things platform to foster Smart Grid technologies integration and reusability," 2016, doi: 10.1109/ICCW.2016.7503818.
- [25] T. Kempitiya, S. Sierla, D. De Silva, M. Yli-Ojanperä, D. Alahakoon, and V. Vyatkin, "An Artificial Intelligence framework for bidding optimization with uncertainty in multiple frequency reserve markets," *Appl. Energy*, vol. 280, 2020, doi: 10.1016/j.apenergy.2020.115918.
- [26] D. De Silva, S. Sierla, D. Alahakoon, E. Osipov, X. Yu, and V. Vyatkin, "Toward Intelligent Industrial Informatics: A Review of Current Developments and Future Directions of Artificial Intelligence in Industrial Applications," *IEEE Industrial Electronics Magazine*, vol. 14, no. 2, 2020, doi: 10.1109/MIE.2019.2952165.
- [27] R. Subramanya, M. Yli-Ojanperä, S. Sierla, T. Hölttä, J. Valtakari, and V. Vyatkin, "A Virtual Power Plant Solution for Aggregating Photovoltaic Systems and Other Distributed Energy Resources for Northern European Primary Frequency Reserves," *Energies*, vol. 14, no. 5, 2021, doi: 10.3390/en14051242.
- [28] E. Nefedov *et al.*, "Energy efficient traffic-based street lighting automation," 2014, doi: 10.1109/ISIE.2014.6864874.
- [29] C. W. Yang, E. Nefedov, S. Sierla, and P. Flikkema, "Vehicle and pedestrian aware street lighting automation," 2015, doi: 10.1109/INDIN.2015.7281917.
- [30] C. Giovanelli, O. Kilkki, S. Sierla, I. Seilonen, and V. Vyatkin, "Task Allocation Algorithm for Energy Resources Providing Frequency Containment Reserves," *IEEE Trans. Ind. Informatics*, vol. 15, no. 2, 2019, doi: 10.1109/TII.2018.2821676.