A SECURE AUTOMATION SOLUTION TO PROVIDE FLEXIBILITY AT LOW-LEVEL GRID – MIDDLEWARE SERVICES

Razgar Ebrahimy
Denmarks Tekniske Universitet – Denmark
raze@dtu.dk

Mohsen Banaei
Denmarks Tekniske Universitet - Denmark
moban@dtu.dk

Henrik Madsen
Denmarks Tekniske Universitet - Denmark
hmad@dtu.dk

Jaime Chen Gallardo
Softcrits – Spain
jaime@softcrits.es

Manuel Diaz Rodríguez
Softcrits – Spain
mdr@softcrits.es

Juan Jacobo Peralta Escalante
Centro de Estudios de Materiales y Control de Obra, S.A – Spain
jacobo.peralta@cemosa.es

Krzysztof Piotrowski
Innovations for High Performance Microelectronics– Germany
piotrowski@ihp-microelectronics.com

ABSTRACT

The digitalisation of the grid at the DSO and consumer levels has created new challenges for the operators and stakeholders involved that require significant coordination and cooperation. More importantly, it requires secure, scalable and interoperable software automation solutions. These solutions should be comprehensive; in essence, that information and data can be used and traced from the low and medium-level transformers to the end user's smart meters and energy management systems. As part of the balance-plus (H2020) project, an interoperable and secure middleware framework has been developed and tested in three demo sites in Italy, France and Denmark. The framework provides greater observability, monitoring and control power to the stakeholders, such as DSOs and aggregators, to utilise the available flexibility from the end-users. The low-level grid is equipped with IoT devices and control systems from vendors and manufactures that are not always interoperable.

INTRODUCTION

Smart energy systems are highly interdependent both in terms of the technologies and the operations [1]. The digitalisation of the grid at the DSO and consumer levels has created new challenges for the operators and stakeholders involved that require significant coordination and cooperation [2]. Therefore, these issues were addressed in the ebalance-plus [3] project by developing a middleware framework with scalability and security principles at its heart. The framework's purpose is to facilitate digital interaction among many stakeholders with complete control over what data to share with others. In the meantime, to improve the resilience of the low-level grid and include the end users in the solution. The proposed framework has been implemented to be deployed on energy management devices in the end-users and on smart gateways at the transformer level. The underlying concept of the middleware is to ensure security and scalability where many stakeholders are involved in providing a service [4]. The ebalance-plus project focuses on unlocking flexibility from the low-level grid in a secure and scalable manner. So far, the framework has provided promising results in three demo sites where it has been deployed to serve as a mediator to offer flexibility and enable various stakeholders systems to communicate securely.

The current implementation is developed in JAVA and Python programming languages and is envisioned to enable the end user's energy management systems to securely participate in providing ancillary services to the grid. The framework also offers modularity and separation by design, where different applications and services can be coupled to provide a common service, such as flexibility or frequency control. The middleware also provides a secure interface between the host applications that can share data in secure and encrypted ways. Ebalance-plus project aims to resale the architecture design to help the community and DSOs to adapt the solution and use it.

EBALANCE-PLUS ARCITECURE

The ebalance-plus system is composed of units, called management units or MUs, that implement algorithms to forecast and manage the available flexibility to incentivise demand response programmes and increase the distribution grid capacity to avoid congestions and advise optimization strategies. Some of the concepts and design in the system architecture follows the Smart-Energy operating system [5] which is tried and tested in another EU projects. In the proposed architecture each MU is considered an autonomous system, such as an embedded PC, although other options are possible, such as virtualizing the MUs in containers or VMs. The units are organized following a tree hierarchy where each MU has a parent MU and possibly multiple children MUs except for the top-level grid management unit (TLGMU) which is the MU located at the root of the tree. Figure 1 illustrates the relationship between the different MUs (depicted as

rectangular dark blue boxes) in a scenario that has four main levels (enumerated from the top of the tree to bottom):

- **TLGMU:** top level grid management unit located, for example, in the cloud.
- **MVGMU:** medium voltage grid management unit located in the primary substation (PS).
- **LVGMU:** low voltage grid management unit located in the secondary substation (SS)**.**
- **CMU:** customer management unit located in the customer premises, buildings, etc.
- **DMU:** device management unit located inside external devices that communicate with the system.

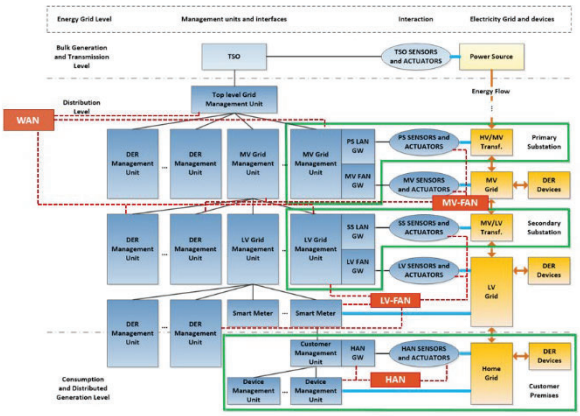

Figure 1 ebalance-plus architecture

In addition, distributed energy resource management units (DERMUs) can be deployed at any level to management DER devices located at different levels of the energy grid. Each unit must be able to exchange information with external devices and with each other to distribute the information generated in the system and transport it to the data consumers which are most often the ebalance-plus algorithms although it can also be GUI applications, SCADA/monitoring applications, etc. Also, data must be stored and consulted at any time prior to its generation. The software that provides such functionality is the **data exchange middleware**, which is a key concept in the architecture. Another key concept is the **adapter module** which assists the middleware in contacting external devices and retrieving the information from them or modifying available setpoints. For example, an adapter for a smart battery might be used to get information from the battery state and to change the charging schedule. This functionality is used by algorithms through the API provided by the data exchange middleware. From the point of view of the algorithm there is not external device, only pieces of data that can be written or set.

DATA EXCHANGE MIDDLEWARE DESIGN

The ebalance-plus system uses a middleware framework that allows the participants to communicate, exchange and store information. The framework stores data in tuple space structures which allows to implement a variety of logical structures that can contain all the information necessary to identify a value, its description, source, and time of creation. The tuple space is accessed by creating variables that can be written, read, or removed. Each variable is further divided into owner spaces. Operations on variables are checked against defined access control policies. By default, each owner can control only their own data. It is possible to grant or revoke permissions to change the default permissions. The variables concept implemented by the framework varies slightly from the general understanding of a variable. The first difference is that by default, each value written to the variable is stored as a separate entry. The second is that data stored in a single variable can have multiple owners. The third difference is that a variable can have multiple fields that are defined when creating the variable. For example, it is possible to create a weather variable that contains fields such as temperature, humidity, or wind. Then, owners can store and share their weather information as they please. The architecture of the framework follows a distributed approach. It allows the participants to store data close to the locations where the data is produced and/or where it might be consumed in order to be processed. The communication is secured through public key infrastructure (PKI).

The framework defines the Data Interface channel that is responsible for establishing a secure and transparent channel for communication between clients and a middleware server instance. The Data Interface channel is available as a library either in Java or Python. In the context of the middleware framework, clients that use the library to communicate and implement functionalities, are referred to as services (see Figure 2). The services use the Data Interface to perform actions on data (variables), permissions and use other system functionalities.

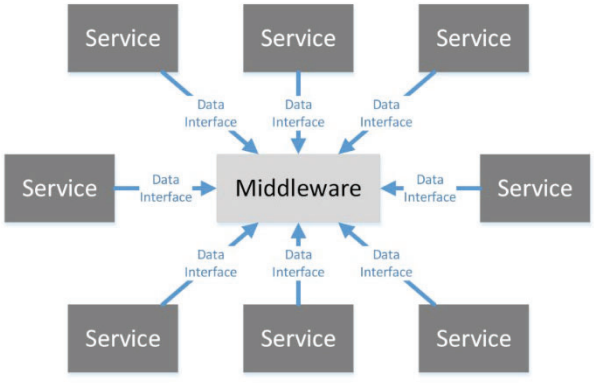

Figure 2 Data exchange middleware services

To protect against malicious services, the middleware platform implements a utility that is responsible for:

- **Bootstrapping secure environments for services.**
- **Running services with adequate privileges.**

The middleware servers can communicate directly or through a framework-provided proxy server (see Figure 3). If both servers are visible to each other, the communication can be direct. If the destination server is not visible to the source server, the communication must be executed through a proxy server.

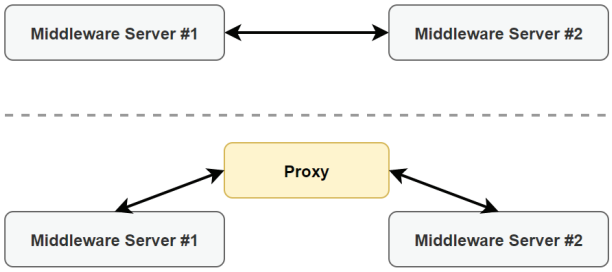

Figure 3 Communication between different instances of the middleware

When two middleware servers communicate, a single channel is opened. In this case, only the destination server must be visible. The response is sent through the same channel as the request.

TOP LEVEL SECURE ARCHITECTURE

The top level of the hierarchy is indeed structured as depicted in Figure 4. The SuperUnit is the MU on the highest level of a given deployment. It may be the TLGMU, but in a setup with many DSO grids connected, it will be an even higher-level MU that is above the TLGMUs. Another additional element is the discovery server that is responsible for maintaining the list of registered stakeholders and MUs together their respective certificates, which it also issues. By that it is has the role of being the certification authority (CA) in the ebalance-plus system. The last element is the proxy server that is responsible for buffering requests targeted at management units located behind network elements, like routers or firewalls that prevent these MUs to be directly addressed from outside. Such MUs need then to collect their requests from the proxy server.

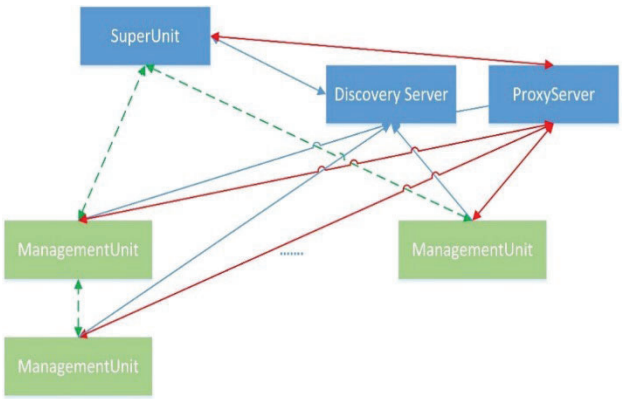

Figure 4 Top level system architecture

SECURE IDENTIFICATION

The hardware, software components and stakeholders introduced in the previous section need a secure way to identify themselves. The platform uses the concept of public key infrastructure (PKI) to introduce security and privacy in communication between participating stakeholders. The public key infrastructure ensures the confidentiality, integrity and authenticity of the messages exchanged on the middleware platform.

The PKI is based on public key encryption which is a cryptographic system based on mathematical problems in which each participant has two keys – public and private. The public key can be distributed publicly and freely to everyone willing to communicate with the participant, while the private key has to be kept as secret as possible by the client, because it can be used to decrypt a message that was previously encrypted with a corresponding public key of the same participant.

Here we deliberately do not specify the security levels in terms of key length or the technology (cryptography) used, as these are parameters that shall be specified for a deployment on the basis of the requirements (standards and values considered secure. The PKI introduces a concept of certificate that most importantly contains the owner information and the public key used in the encryption process. Each certificate has a corresponding private key that is kept separate from the certificate and can be used to decrypt any message that was encrypted with the corresponding public key. Each actor participating in the communication is required to have a valid certificate that was issued (generated and signed) by a trusted entity. The actors include middleware servers, services or middleware proxy servers. The certificate can be used to prove:

a. Identity of the actor participating in the communication,
b. ownership of the public key.

To be able to issue certificates for the entities, a trusted entity has to be established. The technical name of such entity in the PKI is Certification Authority (CA). The certification authority is responsible for:

a. verifying the identity of participating actors,
b. issuing and storing the certificates for verified actors,
c. maintaining a list of revoked certificates,
d. providing a mechanism for checking revoked certificates.

The certification authority has its own certificate with a separate corresponding private key that is kept secret. Each actor has a local copy of the certification authority certificate (authority identification + authority public key).

When an actor is willing to communicate on the middleware platform, she must create a certificate signing request (CSR) which (among others) contains the actor's certificate (identity and public key). The request is forwarded to the certification authority. The

authority performs a verification process to confirm the identity of the requesting actor. When the verification process is successful, the certification authority uses its private key to sign the certificate sent in the CSR. A signed certificate is returned to the requesting actor. Disclosing private keys of both parties (even to each other) at any step of the certification process would break the security enforced by the public key cryptography. Therefore it has to be clearly stated that the private keys of both the requesting actor and the certificate authority are not shared at any step of the certification process and:

- the certification authority uses its private key only to sign certificates,
- the actor uses its private key only to decrypt messages that were encrypted with the corresponding public key.

Without knowing the certification authority's private key, it is computationally extremely expensive (for most of actors/scenarios considered impossible) to create a valid signed certificate (in acceptable period of time) with altered owner information. The actors can use this fact to validate the authenticity of other actors. When attempting to communicate, both actors must provide their signed certificates, of which the signature can be verified. Any modification attempt of data signed in the certificate will result in signature mismatch which will alert the other actor and the connection can be rejected. After the signature verification, both actors can communicate with the certification authority to check if the certificate is not revoked.

When the private key of an actor's signed certificate is compromised or the actor is behaving maliciously, the certificate can be revoked which equals to blacklisting the certificate by the certification authority. A revoked certificate is no longer considered trusted. Based on the circumstances, the certificate can be reissued or cooperation with the malicious actor can be terminated. The certification authority has no power to physically take away the certificate from the actor, therefore other actors have to check whether their communication participant is identifying itself with a revoked certificate.

The implementation of the security features is modular and configurable. It allows to quickly update the underlying mechanisms or chosen cipher suite in case that the currently chosen option is no longer considered secure.

In the ebalance-plus system the following certificate classes are issued by the certification authority:

- middleware instance / management unit certificate,
- stakeholder / user certificate,
- certificate to prove the identity of a service running on behalf of a given stakeholder

We do not certify the services individually, because the stakeholder and the service identities together define the compound identity that defines what the given instance of a service can access (do) and how.

SECURE BOOTSTRAP AND RUNTIME

The services that make use of the functionalities provided by the middleware platform are generic applications. By default, there are no limitations (other than limitations enforced by the operating system) in actions they may perform. In cases where a single device runs a middleware server and services that belong to a single stakeholder or services that communicate with the middleware server are run on physically separate hardware from the server and each other, the threat is minimal. However, in cases where a single device hosts the middleware server and services that belong to different stakeholders, there is a risk of malicious services that have the possibility to obtain confidential information such as private keys, databases, passwords or source code of services that belong to competing stakeholders.

To protect against malicious services, the middleware platform implements a utility that is responsible for:

a. bootstrapping secure environments for services,
b. running services with adequate privileges.

The utility creates a directory for each stakeholder (Figure 5). Each service that runs on behalf of a stakeholder has a corresponding sub-directory inside the stakeholder's directory. The service directory is a secure space for the service to store any files necessary to run the service, including the executable file, certificates, private keys, databases, or passwords. Based on the scenario, a service can have access to all service folders of its stakeholder or only to its own folder. Access to folders of other stakeholders is denied.

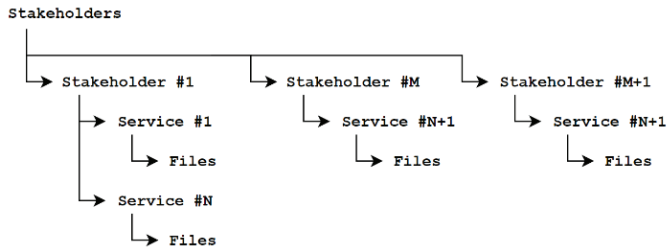

Figure 5 Stakeholder and service separation

The first layer of security is the file system permissions that provide protection from the outside. For the Java services, the second layer of security is Java's Security Manager (SM) that provides protection from the inside. The SM allows to define policies for applications (services) and allows to protect any meaningful resource and enforce limited or specific usage (Figure 6). A similar approach will be defined for Python services.

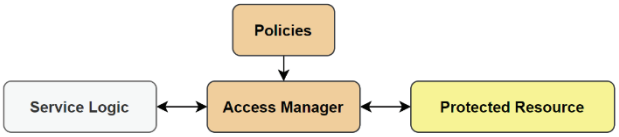

Figure 6 Relation between a service, Access Manager, policies and protected resources

The bootstrapping utility is also used to run the services. After the secure environment for a service is set up and the corresponding policies are created, the utility runs the service executable with the Security Manager enabled that controls the service behaviour in accordance to defined policies. The services are never executed without the assistance of the bootstrapping utility. Doing so could compromise the system security and render the implemented measures impractical.

INTEGRATION METHODOLOGY

The services executed on the management units communicate with each other through the middleware. Each level implements its level-specific functionalities that are built on the middleware platform and features it provides. The evaluation of the middleware platform presented in this document verifies the correctness and evaluates the performance of each functionality offered by the platform within defined certain scenarios that are present in the example ebalance-plus architecture.

The tests have been carried out by creating Java services that implement the defined scenarios and output their results of operations, timestamps and/or delta times (depending on the test) to evaluate the correctness and security, or extract data for further processing (performance evaluation).

During the development phase of the middleware, the underlying data storage (database) used by the middleware to save data, was identified to be a significant bottleneck if the data access is slow. Thus, on the embedded PC hardware platforms, the performance evaluation considers two storage options – the usual µSD card and the much faster USB stick. These two were chosen because of the significant difference in the read and write speeds and to investigate the data access speed influence. Each performance-related test is executed on each storage option and the results are compared.

Each test is documented within its test card (see Table 1) that contains the definition of the test scenario, the initial state requirements and documents the obtained result along with the description of actions that happened during the test execution.

Table 1Test definition template

Test #*N* – title…	**Short description …**
Definition	Aim of the test, detailed description and expected results.
Requirements	Any requirements that must be satisfied before executing the test.
Device(s)	Any devices that take part in the test.
Result	SUCCESS / FAILURE
Result description	The final result of the test, detailed description of the evaluation.
Comments (optional)	If applicable, comments to the test or the result.

CONCLUSION

Smart grids are complex system in which a high number of heterogeneous devices need to exchange information. To simplify the way these devices, communicate a data exchange middleware has been designed in the ebalance-plus project. The middleware provides a data-centric high-level programming abstraction based on simple operations that is used by all actors in the system to hide the complexity of the underlying architecture and hardware devices. To integrate external devices/systems within the middleware an adapter module is used. An adapter module provides an abstraction over each specific external device and oversees transforming requests/responses from/to the middleware/devices. The purpose of this work has been to present the design and evaluation of the middleware used in the ebalance-plus platform. As part of the evaluation, tests have been performed and their execution in order to evaluate the middleware framework that is used as the communication platform in the ebalance-plus project. The correctness and security tests show that the middleware behaves as defined and in a secure manner.

ACKNOWLEDGEMENT

This research was supported by the European Commission through the H2020, project ebalance-project grant number 864283.

REFERENCES

[1] Ebrahimy, R., & Pourmirza, Z. (2017, February). Cyber-interdependency in Smart Energy Systems. In *ICISSP* (pp. 529-537)

[2] Savin, V. D. (2022). Cybersecurity Threats and Vulnerabilities in Energy Transition to Smart Electricity Grids. In *Navigating Through the Crisis: Business, Technological and Ethical Considerations: The 2020 Annual Griffiths School of Management and IT Conference (GSMAC) Vol 2 11* (pp. 71-83). Springer International Publishing.

[3] EbalancePlus project. (2023, January 21). Récupéré sur https://www.ebalanceplus.eu/project/

[4] Alfalouji Q, Schranz T, Kümpel A, Schraven M, Storek T, Gross S, Monti A, Müller D, Schweiger G. IoT Middleware Platforms for Smart Energy Systems: An Empirical Expert Survey. *Buildings*. 2022; 12(5):526. https://doi.org/10.3390/buildings12050526

[5] Madina, C. *et al.* (2020). Technologies and Protocols: The Experience of the Three SmartNet Pilots. In: Migliavacca, G. (eds) TSO-DSO Interactions and Ancillary Services in Electricity Transmission and Distribution Networks. Springer, Cham. https://doi.org/10.1007/978-3-030-29203-4_6