

A Software Framework for Context-aware Secure Intelligent Applications of Distributed Systems

Soumoud Fkaier^{1,2,3} ^a, Mohamed Khalgui^{2,3} ^b and Georg Frey¹ ^c

¹Chair of Automation and Energy Systems, Saarland University, Saarbruecken, Germany

²Tunisia Polytechnic School, Carthage University, Tunis, Tunisia

³INSAT LISI Lab, Carthage University, Tunis, Tunisia

Keywords: Framework, Context-awareness, Reconfiguration, Intelligence, Coordination, Security.

Abstract: Future distributed reconfigurable systems need to provide smarter services. Therefore the used software need to include advanced mechanisms such as the context-awareness, artificial intelligence, collaboration between distributed parts of the system, as well as the secure data exchange. Most of the existing context-aware frameworks are restricted to a part of the mentioned scopes and are generally not suitable to reconfigurable systems. Hence, there is a need for a software engineering solution that reconciles all the said requirements. In this paper, we propose a software framework for developing collaborative, intelligent, and secure applications of distributed systems. This paper extends an existing framework with the mentioned features and shows its new structure and design. A software tool developing the proposed contributions is implemented using Java programming language. An example of microgrids software applications is used to show the suitability of the contributions.


1 INTRODUCTION


Smart behaviour is the main objective of current and future context-aware reconfigurable systems. These latter have to meet the challenge of providing more sophisticated behaviour. Therefore, its software applications need to be strong enough to offer such features. The advanced features can not only be reached by means of context-awareness computing, but also by means of different other concepts. Artificial intelligence concepts can play a major role to support smartness since they allow to develop knowledge reasoning for multiple use cases (e. g. the prediction, planning, learning). Besides, distribution paradigm is becoming more and more substantial and most of the current systems are considering it thanks to its assets when compared to the centralized one. In fact, centralized approach has the limitation of single-point-of-load and single-point-of-failure, which is less efficient for many cases. Furthermore, secure information exchange between the distributed parts is an undeniable condition for reliable system behaviour. In


addition to all that, common requirements such as the real-time as well as functional constraints, need also to be satisfied.

The development of context-awareness software applications of such systems is a complicated task to perform. For this, different context-awareness software frameworks have been proposed at the aim of facilitating their development. However, most of them handle limited features due to the complexity of their orchestration. It is hard to provide a generic software solution that can cover intelligence, coordination and security for different case studies in a same software support. In fact, a design challenge rises in the definition of the structure of a framework that needs to cover miscellaneous requirements. Few are the frameworks that consider the coordination requirement of distributed systems along with intelligence or security. The work reported in (Fkaier et al., 2016a) introduces an important framework that allows to implement context-aware reconfigurable applications running under real-time and functional constraints but despite its importance, it is still missing the ability to handle advanced intelligence issues, to operate smoothly in a distributed system, and to guarantee security of exchanged data.

To overcome the aforementioned limitations, the

^a  <https://orcid.org/0000-0003-2903-0943>

^b  <https://orcid.org/0000-0001-6311-3588>

^c  <https://orcid.org/0000-0001-8483-0295>

current paper proposes an extension to the framework reported in (Fkaier et al., 2016a). We mainly enhance the first and second layers (the Reconfiguration Layer and Context Control Layer) by adding new pools and refining the definition of existing ones. We propose a generic artificial intelligence mechanism that can be used for different purposes. We also introduce a coordination mechanism that helps to achieve coherent reconfigurations. Moreover, we define a module to be a container of security mechanisms. Hence, besides the existing techniques, the new enhanced framework allows to implement more features promoting applications smartness and autonomy. Furthermore, we implemented the proposed framework in a software tool using Java programming language.

In order to show the suitability of the new framework, an example of microgrids software applications is addressed. In fact, smart microgrids are a type of systems that are knowing an evolution through integrating the Information and Communication Technologies (ICT). Despite the numerous challenges identified in this system, most of works tackle the electric power level or the automation and control, and the software side is generally omitted although it is a key responsible for system smartness (Anvaari et al., 2012), (Ghosh et al., 2020).

The outline of this paper is organized as follows. Section 2 presents the state of the art. Section 3 introduces the definition of the new framework. Section 4 shows how the new concepts are suitable through a microgrids software application case study. Section 5 evaluates the performances of the contributions. Finally, Section 6 concludes the paper and opens perspectives.

2 STATE OF THE ART

In this section, we present in Section 2.1 an overview of the existing context-awareness frameworks. Then, we show in Section 2.2 the importance of context-awareness, intelligence, security, and coordination for smart grids.

2.1 Context-awareness Frameworks

Since its appearance, context-awareness computing has been an important enabler of systems smart behavior. However, alone, this concept cannot provide expected results especially with the never-ending expectations of systems autonomy. This is why, software frameworks have been intensively developed. The authors of (Sikder et al., 2019) have proposed a

framework for the detection of threats on smart devices, but despite its importance this work do neither consider the real-time reconfigurations nor the coordination between distributed agents. In (Aid and Rasoul, 2017) a context-aware framework for the disaster management is proposed. This work also does not treat distribution, security and real-time reconfigurations. The work reported in (Tang et al., 2014) presents a development environment that supports the pervasive applications. Distribution needs are taken into consideration by this work but requirements like secure coordination and real-time reconfiguration are not covered. The work reported in (Alhamid et al., 2016) proposes a collaborative context-aware framework for the development of applications in ambient intelligence environment. This framework also does not consider distribution, security, and reconfiguration. In (Bucchiarone et al., 2017) a context-aware framework for the development of processes in the internet of services is proposed. This latter does not handle the coordination and real-time reconfigurations. In (Fkaier et al., 2016a) the authors have proposed a context-awareness framework for the development of reconfigurable systems having real-time and functional constraints. The latter is an important support of the development of future smart systems, however it does not support the distribution, security, and advanced intelligence features. As it can be seen, most of the existing context-awareness frameworks are dedicated to specific features, and the majority do not tackle coordination and security.

2.2 Smart Grids and Intelligence, Security, Distribution

Smart grids are new electricity grids promoting the integration of ICTs into the traditional grid in order to enable enhanced functionalities through the exchange of data between the grid components (users, utility, and control infrastructure). These grids need to use advanced software concepts to achieve the required functionalities. In (Fkaier et al., 2020a) and (Fkaier et al., 2020b) context-awareness computing was used to enable the dynamic interaction with the changing environment in real-time. **Concerning the intelligence**, the authors of (Henri et al., 2020) have proposed a simulator of artificial intelligence concepts for microgrids. In (Khan et al., 2018) also discusses the adoption of artificial intelligence techniques in the microgrids development. The research reported in (Hubana, 2020) uses the artificial intelligence concepts for faults location and recovery in microgrids. **The security** have been an inherent requirement to all computerized systems. Ensuring se-

curity is an important condition to maintain a good reliability level. The work reported in (Elgamal et al., 2020) studies the microgrids profit maximisation all with sustaining its security. In (Hosseinimoghadam et al., 2020) the authors study the security of microgrids based on neuro-fuzzy inference model. The authors of (Dabbaghjamanesh et al., 2019) have used the blockchain technology to enhance the privacy of networked microgrids. The research in (Ferdowsi et al., 2019) has also studied the reconfiguration under security constraints. **With the increasing adoption of distributed paradigm** in the operational tasks of microgrids, coordination becomes necessary for the logic coherence. The authors of (Mbungu et al., 2019) have surveyed the optimal smart energy coordination for microgrids. In (El-Naily et al., 2020) a coordination approach is used to resolve the microgrids faults, while in (Wu et al., 2019) a distributed hierarchical coordination strategy for parallel inverters is proposed to improve the system flexibility. As it can be seen, most of the presented works about smart grids software are restricted to one or two of the system requirements. Software infrastructures that enables to develop multiple requirements using one software solutions are still needed.

3 ENHANCED FRAMEWORK

In this section, we introduce the enhanced framework, where Section 3.1 shows the extension of RL and Section 3.2 shows the extension of CCL.

The selected framework, was introduced in (Fkaier et al., 2016a) and its concepts are based on a meta-model reported in (Fkaier et al., 2017). A software tool implementing the framework semantics is reported in (Fkaier et al., 2016b).

The framework has a layered architecture composed of four layers: (i) Reconfiguration Layer (RL): responsible for the interaction of the framework logic with the environment through two modules of inputs and outputs, (ii) Context Control Layer (CCL): contains a set of pools where each of which is responsible for the checking of well-defined constraints, (iii) Services Layer (SL): holds the functionalities of applications in the form of services, and (iv) Communication Layer (CL): represents of the framework’s services.

The second layer, CCL, was composed of three pools: controller (C), functional (FP), and real-time (RTP) pools. The real-time pool provides the techniques to analyze the timing behavior of applications through checking the timing constraints, e. g., the operational deadlines. The functional pool offers the ability to check functional constraints related

to dependencies between the services (e.g. precedence constraints) and especially the coherence relationships (i.e., the inclusion/exclusion relationships). We come finally to the last pool, the controller, that ensures the orchestration of the operation of the pools and layers of the whole architecture, and holds the application’s logic.

This framework is proved to be efficient in the development of software applications of complex systems such as the airports baggage handling systems as reported in (Fkaier et al., 2016a). However, despite its undeniable assets, it still have some limitations especially for the development of distributed, intelligent, and secure systems. To satisfy these needs, we extend the definition of the framework concepts by updating the definition of the first and second layers (RL and CCL). We refine the mechanisms of the inputs pool by integrating a data processing mechanism using more sophisticated context-awareness reasoning technique of RL. We define also three new pools in the CCL: (1) intelligence pool, (2) coordination pool, and (3) security pool. The new schematic representation of the framework structure is depicted in Figure 1. In addition, the extension includes updating the controller capabilities in a way to support the interaction with the new pools. The internal dynamics of each new pool are defined in the following.

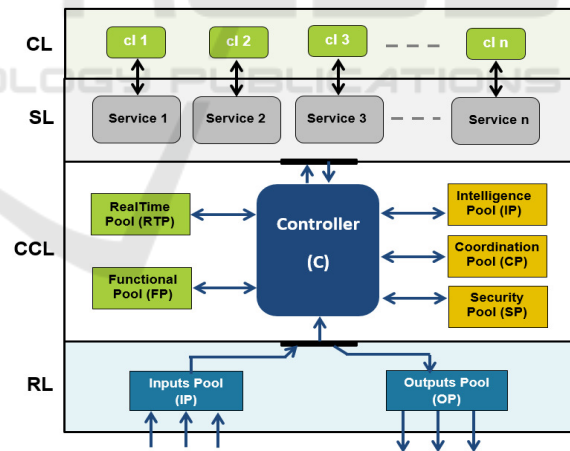


Figure 1: Framework architecture.

3.1 RL Extension

Input data are values coming from the sensors and measurements infrastructure, then structured thanks to a hardware control layer, and finally handed-over to the software layer, specifically to the IP of the framework. Figure 2 shows the abstraction of the information flow from field level until the Inputs Pool.

The Inputs Pool receives input data (denoted by

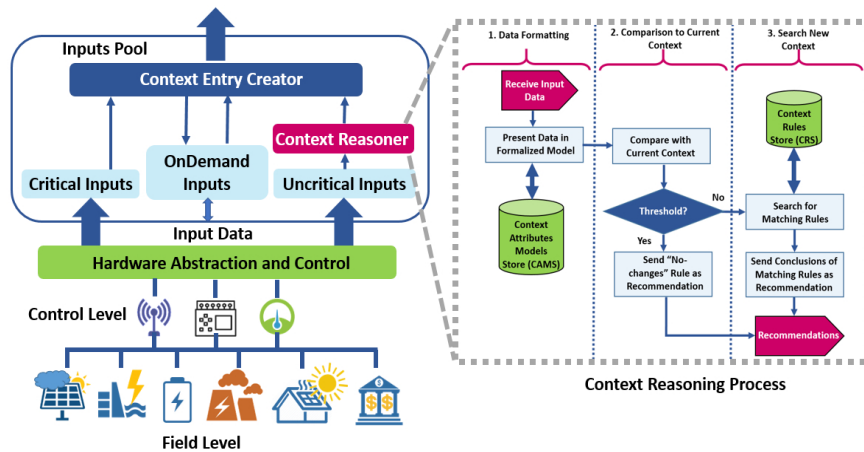


Figure 2: Schematic presentation of the inputs pool.

id), processes it, then creates a Context Entry CE and sends it to the controller pool of the upper layer (i.e., CCL). Input data are classified into two types: critical and uncritical. In order to specify which data are the critical and which are the uncritical, developers have to create the context ontology as mentioned in (Fkaier. et al., 2020a). The context ontology consists in a basis to generate Context Rows CR where $CR = \{i_1, i_2, \dots, i_n\}$ with i_j is a context item. A context item is defined as a tuple $i = \langle t, l, v \rangle$ where t is the type of the item expressed through its root in the ontology model (i.e., the ontology o_a , entity e_b , and attributes a_c), l is the label of the item, and v is the value of the item. If the input data are classified as critical, then the Context Entry Creator (see Figure 2) creates the context entry as

$$CE = \begin{cases} (ts||id) & \text{if } id \text{ is critical} \\ (ts||id||r) & \text{if } id \text{ is uncritical} \end{cases} \quad (1)$$

with ts is a time stamp and r is the set of recommendations resulted from the context reasoning. If the input data are classified as uncritical then a context awareness reasoning is performed according to reasoning method defined in (Fkaier. et al., 2020a). The pseudocode of the input module is then defined as mentioned in Algorithm 1.

3.2 CCL Extension

3.2.1 Controller

After receiving CE from the inputs pool, the controller parses it and decides whether a reconfiguration should take place. This logic is use-case-dependent and must be defined by the system owners. In case a reconfiguration is required, the controller starts to search for the appropriate configuration. For this, the

Algorithm 1: Inputs Pool Logic.

```

Inputs:  $id$ 
Outputs:  $CE$ 
 $CE = \emptyset;$ 
while (true)
     $id_{critical} = \text{ListenForCriticalData}();$ 
     $id = id_{critical};$ 
     $CE = \text{CreateCE}(ts, id);$ 
end
foreach (period) do
     $id = \text{ReadData}();$ 
     $r = \text{ContextReasoning}(id);$ 
     $CE = \text{CreateCE}(ts, id, r);$ 
end
if (input demand) then
     $id = \text{ReadData}();$ 
     $CE = \text{CreateCE}(ts, id);$ 
end
return  $CE;$ 
    
```

controller need to analyze the convenience of a certain configuration with regard to some constraints. Previously, only functional and timing constraints can be analyzed, but now more advanced requirements can be checked such as the intelligence, coordination with other peers in the system, and security of critical transactions. By the addition of the new pools, the controller code is also extended in order to support the new components. New communication requests and replies are defined and the structure of the controller code becomes as depicted in the UML (Unified Modeling Language) class diagram of Figure 3. The code is organized with respect to the separation of concerns as well as the single responsibility principles into three packages *Handling Layers*, *Using Pools*, and *Control Main Logic*: 1) The package *Handling Layers* contains the classes that enable the controller

to supervise the work of the RL, SL, and CL. 2) The package *Using Pools* contains the classes enabling the use of one or many pools (developers are free to use the pools they need according to the use case and not necessarily all pools). 3) The package *Control Main Logic* as its name indicates holds the main logic of the application.

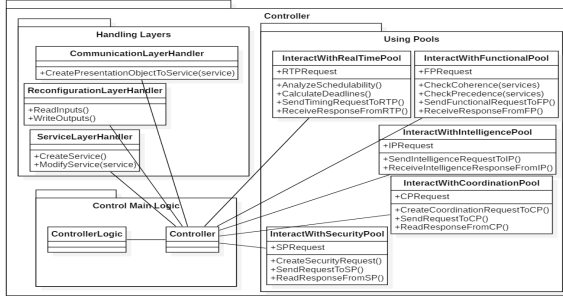


Figure 3: Controller class diagram.

3.2.2 Artificial Intelligence Pool (AIP)

The aim behind adding this pool in CCL is to make applications more intelligent. Artificial intelligence concepts can be used for many purposes such as the optimization tasks of a system, prediction tasks, etc. For this, we propose to create a generalized artificial intelligence technique that can be adapted/adjusted according to the use case. This pool analyzes an intelligence request created by the controller in order to process it using an Expert System (ExpSys). Generally, existing expert systems consist of a knowledge base and an inference engine, but for more efficiency in terms of computational power (used resources such as the memory, CPU time, energy, etc.) we add a system history that helps to avoid triggering known scenarios. Hence the ExpSys is defined by, $ExpSys = (I, K, H)$, where I denotes the inference engine, K denotes the knowledge base and H denotes the intelligence pool history (see Figure 4).

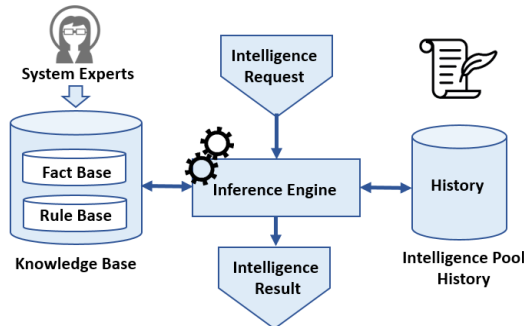


Figure 4: Intelligence Pool.

The knowledge base, K , accomplishes its task using two bases $K = (F, R)$: (1) the facts base, given by $F = \{f_j, j = (1, \dots, n)\}$, and it contains facts about the system where a fact generally reflects an evidence. (2) The rules base, given by $R = \{r_i, i = (1, \dots, m)\}$, storing rules which are logical forms enabling the deduction of new facts. The rules must be defined by human experts of a specific domain that muster the considered case and know how the system should behave. A rule has the form of $r_i = \text{“if premise then conclusion”}$. The system history, H , stores the history of the system behavior defined by the set: $\{h_q, q = (1, \dots, k)\}$, where h_q represents a previous demarche executed by the pool. The inference engine, I , uses K and H in order to deduct a new knowledge. The method of reasoning of I can be the forward or backward chaining.

3.2.3 Coordination Pool (CP)

In distributed approach, the operational tasks are performed based on geographically distributed peers. One important challenge that has often been faced is how to make a real coherent distributed and collaborative strategy.

The coordination between the distributed units is not an easy task to perform especially that the need to other peers can be manifested during the preparation of reconfigurations. This is why, we propose a coordination pool, denoted by CP , in the CCL that will be able to satisfy this requirement.

$$CM[i][j] = \begin{matrix} \text{system peers} \\ \left. \begin{matrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \dots & \dots & \dots & \dots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{matrix} \right\} \text{configurations} \end{matrix} \quad (2)$$

To accomplish its task, CP uses a matrix of configurations called Coordination Matrix (CM) of size (n, m) in which the columns represent all the existing peers in the system, and the lines represent the different configurations. Whenever a coordination with other system peers is needed, a specific peer must consult its CM and depending on the context decides which (re)configuration to process.

3.2.4 Security Pool (SP)

Coordination between the distributed peers of a system requires an exchange of data where some of them can be critical. Hence, securing them is important to guarantee successful applications deployment. In particular, it is required to ensure data integrity and confidentiality of some functionalities and/or reconfigurations. For these reasons, we add a security pool

to be a container of security mechanisms such as the algorithms necessary for encryption/decryption (e.g. RSA, ElGamal, ECIES), user access supervision, error detection software, blockchain-related algorithms (e.g. mining, consensus, validation). This pool provides the base to use such techniques and developers can extend it with any required technique. Hence, this pool is defined as $SP = \{st_1, \dots, st_n\}$, where st_i stands for a security technique.

4 CASE STUDY

This section presents an example of microgrids software application development using the framework. Section 4.1 presents the case study and Section 4.2 depicts the software development.

4.1 Case Study Presentation

Microgrids are electricity grids promoted with information systems to make smarter electricity generation, distribution, and consumption. Microgrids are also characterised by the integration of the renewable energy resources and the electricity storage systems as detailed in (Fkaier. et al., 2020a). Hence, complex and diversified functionalities need to be included in the software applications in order to provide efficient operation. In this paper, we demonstrate how the proposed framework facilitates the development of such functionalities. We consider the microgrids system depicted in Figure 5 where the proposed framework is used to the development of software applications of the software level. The considered grid consists of three microgrids mg_1, mg_2, mg_3 , where each has its loads, renewable sources, and storage system. In this paper, the configurable components set of a microgrid mg_i is defined as $C_{mg} = \{B, U, RES, ROS\}$, where B is the set of batteries, U is the connection to utility, RES is the set of renewable energy sources, and ROS is the set of remotely operating switches that connects to the other microgrids.

Every microgrid mg_i has a set of configurations, denoted by $CFG = \{cfg_1, cfg_2, \dots, cfg_n\}$, with n is the number of all configurations. A configuration cfg_j determines the combination of components arrangements (i.e., the mode of use). A cfg_j is composed of two main sections: (i) internal configuration, $cfg_{j_{int}}$: determining the way of use of the microgrid components such as the batteries, the switch of the utility, and the RES for internal functionalities, (ii) external configuration, $cfg_{j_{ext}}$: determining the way of use of the microgrid components for external functionalities. These external functionalities are used, in

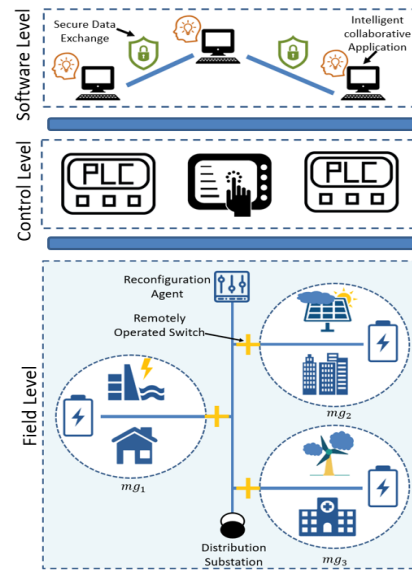


Figure 5: Considered microgrids system.

this paper, in the trading or emergency cases.

4.2 Software Applications Development

The software applications of the three microgrids mentioned in Figure 5 are implemented with the proposed framework. For simplicity reasons, let us consider only three services that are performed by the application as follows: trading electricity with other microgrids, storage management, and the renewable energy integration. Hence, the definition of the services layer is given by $SL = \{S_{Trading}, S_{BatteryScheduling}, S_{RESControl}\}$. Figure 6 shows the structure of the microgrid application according to the proposed framework.

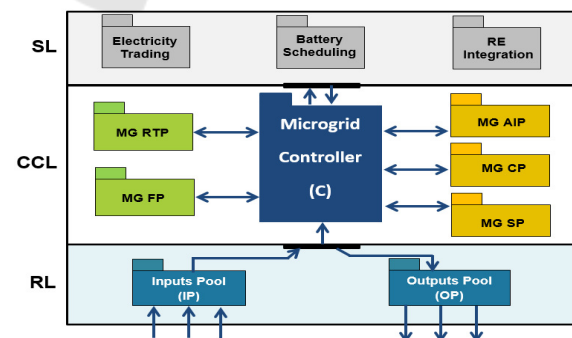


Figure 6: Microgrids application structure.

Before showing the details of the proposed scenarios, let us first define the settings of the framework that are parts of the defined pools, such as possible configurations CFG , context model with Ontology Web Language (OWL), context rules store CRS, rules base

RB of the AIP, the coordination matrix CM of the CP. Let us consider the possible configurations of mg_2 as shown in Table 1. This table is used by the controller module of CCL to determine reconfigurations and it is saved in an XML format.

Table 1: Configurations of mg_2 .

ID	Internal			External
	B	U	RES	ROS2
Cfg 1	MC	On	TA	PS
Cfg 2	HC	On	PA	TS
Cfg 3	MD	On	TA	PS
Cfg 4	HD	On	PA	TS

In this table MC: Medium Charging, HC: High Charging, MD: Medium Discharging, HD: High Discharging, TA: Total Activation, PA: Partial Activation, TS: Total Supply, PS: Partial Supply.

In this paper, we define the microgrid context model with OWL language as shown in Figure 7. The context modeling is created with the aim of analyzing real measurements that help to decide whether to charge or discharge batteries and also whether to turn on or off the connection to the main utility.

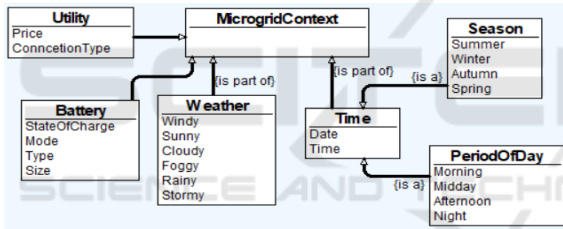


Figure 7: Microgrids context model with OWL language.

Using the context model of Figure 7, the context rules store is defined as depicted in Table 2. The role of the AIP is to decide whether it is required to initiate a trading session to buy electricity, to analyze more the recommendations of the RL especially in case of recommendation to switch off from the main grid. It is needed to avoid blackouts and to decide how to efficiently use the battery (when to charge/discharge, and with which intensity medium or high). The rules used by the AIP are given in Table 3.

Let us consider a time driven scenario to manifest the need to context-awareness, reconfiguration, AIP, CP, and SP. The inputs pool, as mentioned in the previous section, sends a CE in a periodic way in case of uncritical data and in sporadic way (i.e., event driven) in case of critical data. Initially, the application uses the configuration Cfg_2 mentioned in Table 1 which means that the microgrid is highly charging the batteries while it is not activating all RES. Also, it has the context row CR_0 mentioned in Table 4.

Scenario 1: At $t_1 = 12 : 00, 15^{th}, July$. The inputs pool collect data in order to create a new context row CR_1 . Let us assume the newly values are as mentioned in Table 5.

According to the context reasoning process shown in Figure 2, the difference between the current and new rows is calculate as follows: $Difference = \sum_{j=1}^n (|x_j - v_j| / [(x_j + v_j) / 2] \times 100) / n$, which gives 24,76%. Assuming that the fixed difference threshold is 10%, we need to proceed to the next step and trigger rules of CRS. This means that the context is changed, mainly timing ontology, and new recommendations need to be found to achieve better efficiency. From the CRS of Table 2, the conditions of R2 are met, then R2 is fired and recommendations to “Discharge batteries” is obtained. Therefore, the IP creates the following context entry $CE_1 = (12 : 00, 15/07 || PeriodOfDay = Midday, LevelOfCharge = 90\% || (Discharge batteries))$. The controller of CCL receives CE_1 and must now find whether this recommendation is efficient. Since such a decision is hard to take and human expertise is required to determine which can preserve batteries efficiency, the controller send a request to the AIP for analysis. The controller module, with the help of the IP, provides the information of actual value of loads (L) which is high since it is midday, the actual level of solar panels generation (RESG) which is also high, the actual market electricity price (MEP) which is medium, and the predicted solar generation of tomorrow (TSG) this is assumed to be low. Having these facts, the AIP parses the RB and collects triggerable rules. In this case, Rule 4 can be triggered, which results in “medium battery discharge”. Figure 8 depicts some of the methods developed as part of the AIP.

```

public List<String> updateFactBase(List<String> TR)
{
    List<String> updatedFB= new ArrayList<String>();
    for(String tr: TR)
    {
        if(!updatedFB.contains(getDeduction(tr))){
            updatedFB.add(getDeduction(tr));
        }
    }
    return updatedFB;
}
public String getDeduction(String rule)
{
    String result= "";
    int startPosition= rule.indexOf(",")+2;
    int endPosition=rule.indexOf(".");
    result=rule.substring(startPosition,endPosition);
    return result;
}

```

Figure 8: Some of the methods of AIP.

The result is given back to the controller and this latter searches in the possible configurations (see Table 1)

Table 2: Context Rules Store (CRS).

R1	if (Weather is Sunny) and (PeriodOfDay is Morning) and (Season is Summer) → charge batteries
R2	if (Weather is Sunny) and (PeriodOfDay is Midday) and (Season is Summer) → (discharge batteries)
R3	if (Weather is Cloudy) and (PeriodOfDay is Morning) and (Season is Summer) → switch on grid
R4	if (Weather is Cloudy) and (PeriodOfDay is Midday) and (Season is Summer) → discharge batteries

 Table 3: Rules Base (RB) for mg_2 .

Rule1	if (BC is high) and (L is low) and (RESG is high) and (MEP is high) and (TSG is high) → high battery charge
Rule2	if (BC is medium) and (L is low) and (RESG is low) and (MEP is high) and (TSG is low) → medium battery charge
Rule3	if (BC is high) and (L is medium) and (RESG is high) and (MEP is high) and (TSG is low) → high battery discharge
Rule4	if (BC is high) and (L is high) and (RESG is high) and (MEP is medium) and (TSG is low) → medium battery discharge

In this table BC: battery level of charge, L: loads, RESG: level of solar panels generation, MEP: market electricity price, TSG: tomorrow solar generation.

 Table 4: Initial context row CR_0 .

i_1	$\langle o_{utility}.a_{ConnectionMode}, OnMode, 1 \rangle$
i_2	$\langle o_{Battery}.a_{LevelOfCharge}, medium, 50\% \rangle$
i_3	$\langle o_{Weather}.a_{type}, Sunny, 2 \rangle$
i_4	$\langle o_{Time}.e_{Season}.a_{name}, Summer, 1 \rangle$
i_5	$\langle o_{Time}.e_{PeriodOfDay}.a_{name}, Morning, 1 \rangle$

 Table 5: Measured context row CR_1 .

i_1	$\langle o_{utility}.a_{ConnectionMode}, OnMode, 1 \rangle$
i_2	$\langle o_{Battery}.a_{LevelOfCharge}, high, 90\% \rangle$
i_3	$\langle o_{Weather}.a_{type}, Sunny, 2 \rangle$
i_4	$\langle o_{Time}.e_{Season}.a_{name}, Summer, 1 \rangle$
i_5	$\langle o_{Time}.e_{PeriodOfDay}.a_{name}, Midday, 2 \rangle$

which configuration matches the need. cfg_3 is the configuration that suits the requirements of the context. Hence, the controller starts to change from cfg_2 to cfg_3 by loading the proper methods developed as part of the service $s_{BatteryScheduling}$ which contains the right methods enabling the new battery and RES operation style. The execution of the changes must respect functional and timing constraints as detailed in (Fkaier et al., 2016a).

As it can be seen from Scenario 1, the enhanced inputs pools served to provide awareness about the system environment where a reconfiguration possibility is then analyzed by the controller module. Also, thanks to the AIP the battery scheduling measures are consolidated using the experts knowledge.

Scenario 2: Later on, at $t_2 = 16 : 00$, 15th, July. mg_2 receives an emergency call published by mg_3 after blackouts are occurring due to rocks and trees fall on electric lines, which requires some time to be repaired. The emergency call includes the information of the needed electricity quantity Q_{mg_3} and the required time of supply T_{mg_3} . In this case, a coordination session starts between the three microgrids based on the coordination matrix (see Table 6) as follows:

- mg_3 publishes an emergency request $R_{mg_3} = (Q_{mg_3}, T_{mg_3})$ to the rest of the grid.
- mg_1 (resp. mg_2) checks the current and planned resources, and decide if it is possible to help mg_3 .
- mg_1 (resp. mg_2) parses the coordination matrix CM and selects the possible configurations.
- mg_1 (resp. mg_2) sends the found possible configuration to mg_3 .
- mg_3 selects a configuration based on this logic:
 - if received configurations contain a configuration where only one microgrid can provide the electricity, then select this configuration. In case more than one configuration is found, then select the one related to the nearest microgrid geographically.
 - else select the shared configuration between mg_2 and mg_1 . If more than one configuration are shared, then select the one with nearest microgrids geographically.

$config_7$, $config_8$, and $config_9$ are the subject of discussion between mg_2 and mg_1 , where any decision is made based on the current configuration of a microgrid (i.e., cfg_4 for mg_2). Assuming that mg_1 can only provide a partial supply since it does not have big benefit from the water dams as RES in the summer, $config_8$ is the matching configuration. In case a partial supply is required by mg_1 and mg_3 , internal reconfigurations (see Table 1) need to be conducted as mentioned in Scenario 1. In order to transfer electricity to mg_3 , mg_1 needs to activate the remotely operating switches binding the two microgrids.

We can see from this scenario, that the use of the coordination pools has helped microgrids to resolve the fault through a collaboration based on known configurations.

Scenario 3: at $t_3 = 09 : 00$, 18th, July. mg_1 publishes a request R_{mg_1} to buy the quantity of electricity

Table 6: Coordination Matrix (CM).

	mg_1	mg_2	mg_3
$config_1$	F	TS	I
$config_2$	F	I	TS
$config_3$	F	PS	PS
$config_4$	PS	F	PS
$config_5$	TS	F	I
$config_6$	I	F	TS
$config_7$	PS	PS	F
$config_8$	I	TS	F
$config_9$	TS	I	F

In this table I: idle, F: Faulty, PS: Partial Supply, TS: Total Supply.

Q_{mg_1} during the period T_{mg_1} . Here in order to participate in the trading process, each microgrid should check its resources and find the proper configuration to deploy in case of its selection as a seller. Details of context analysis and intelligent storage handling should be conducted as demonstrated in the first scenario. In this scenario, we focus on another important side of microgrids communication, which is the security of exchanged data.

In order to properly conduct a trading process that preserves the data immutability and integrity, blockchain technology is used as a secure storage medium. Hence, each microgrid must exchange data through the blockchain (i.e., requests, bids, decisions are added to the chain). In order to be able to publish any transaction into the blockchain, a microgrid must create a block and add it to the blockchain. The techniques used to create and add blocks are defined in the security pool.

mg_2 creates the following bid $Bid_{1,mg_2} = (id_{bid_1} || price_{bid_1})$. Then, it creates a block having this bid as follows, first it must get the last block added to the blockchain, calculate the hash code of the new block based on the previous one, add the bid (the argument tx in Figure 9) and mine the block.

Hence, the security pool serves as a container of security methods necessary to maintain reliable and secure trading process between the distributed microgrids.

5 PERFORMANCE EVALUATION

Since the contribution of this paper is to add artificial intelligence, coordination and security pools to the control aspects of the logic of software applications, we focus on evaluating these aspects (Section 5.1). In Section 5.2, a comparison with related works is conducted.

```

public boolean addTransaction(Transaction tx) {
    if(tx == null) return false;
    if(!"0".equals(previousHash)) {
        if((tx.processTransaction() != true)) {
            return false;
        }
    }
    listTX.add(tx);
    return true;
}

public void mineBlock(int diff) {
    merkleRoot = StringUtil.getMerkleRoot(listTX);
    String goal = StringUtil.getDifficultyString(diff);
    while(!hash.substring(0, diff).equals(goal)) {
        nonce ++;
        hash = calculateHash();
    }
}

public String calculateHash() {
    return StringUtil.applySha256(
        previousHash +
        Long.toString(timestamp) +
        Integer.toString(nonce) +
        merkleRoot);
}

```

Figure 9: Some of the blockchain methods required in the trading.

5.1 Discussions

Coordination Pool: In order to show the efficiency of the coordination pool, let us consider an increasing number of peers (i.e., microgrids) in the system and see how the pool behaves.

Thanks to creating a predefined coordination matrix that specifies the possible configurations, it is not needed that peers exchange data with each other in order to find a valid system state. If each peer needs to negotiate with each other peer in the system at runtime, then there will be a continuously increasing coordination time and data with the increasing number of peers and number of configurations (see Figure 10).

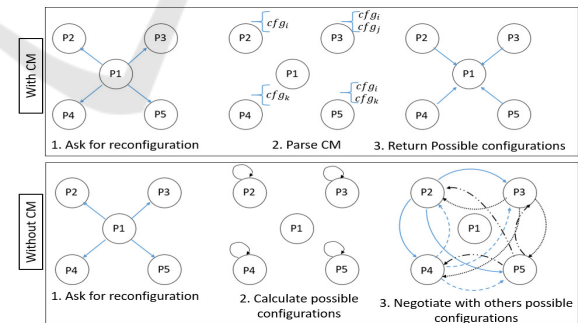


Figure 10: Number of transactions with CM versus without CM.

In case a coordination matrix is predefined, for n peers it is required to exchange $2 * (n - 1)$ transaction. However, in case peers need to find a configuration at run-time, the number of transaction can reach $(n - 1) + (n - 1) * (n - 2)$. Let us assume that in average, a transaction takes 2 minutes. Figure 11 shows the time taken to find the proper configuration in minutes.

Table 7: Comparison of our framework with other context-aware frameworks.

	Coordination	AI	Security	Reconfiguration
(Sikder et al., 2019)	-	-	✓	-
(Aid and Rassoul, 2017)	-	-	✓	-
(Bucchiarone et al., 2017)	-	✓	-	-
Our framework	✓	✓	✓	✓

In this table: ✓ means addressed, - means not addressed.

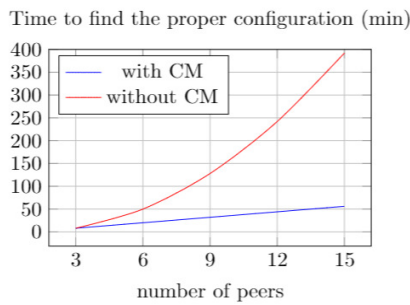


Figure 11: Time to find proper configuration when scaling the system.

Figure 11 depicts that in our approach, the coordination time is very slightly increasing by the increasing number of peers since they need only to parse the coordination matrix, select possible configurations and send it to the concerned peer.

Artificial Intelligence Pool: The artificial intelligence pool (AIP) builds upon an enhanced expert system. Adopting expert systems to take decision is very efficient thanks to its: (1) speed: automated rules reduce the complexity of work and find decisions for repetitive problems, (2) culmination: sum of diverse human expertise, (3) clearness: it provides reasonable explanations of the made decisions. However, despite these assets, intelligence techniques that builds upon a rule base have some drawbacks, mainly a maintenance one. In fact, rules need to be updated from time to time in order to upgrade the logic or to include new knowledge. Further, it is sometimes complex to extend existing rules, a lot of effort might be required.

5.2 Comparison to Related Frameworks

We finish this section with a comparative study in which we highlight the advantages of the proposed framework with respect to other works. We summarize the study in Table 7.

Compared with the existing context-awareness frameworks, the proposed one provides the opportunity to tackle numerous requirements of today’s smart systems. Particularly, the framework allows to develop security, coordination and intelligence logic. Most of the works address one specific feature not a

combination of them. More importantly, reconfiguration which a key enabler of system’s behavior adaptation is often not addressed. The frameworks presented in (Sikder et al., 2019) and (Aid and Rassoul, 2017) consider security but important requirements such as reconfiguration and coordination are not satisfied. The framework reported in (Bucchiarone et al., 2017) uses the artificial intelligence techniques to provide knowledge management, however it does not address security, coordination, nor reconfiguration.

6 CONCLUSIONS

This paper has introduced a new software framework that overcomes the limitations of the existing frameworks. The proposed framework enables the development of context-aware, distributed, intelligent, and secure software applications of reconfigurable systems.

The proposed framework introduces a solution to the coordination of peers in a distributed system. Hence, collaborative behavior can be achieved efficiently and seamlessly. It also introduces an enhanced expert system to be the promoter of intelligence operations. In order to keep a trustful data exchange for specific transactions, the framework includes a security pool having the role of holding the security techniques.

Thanks to the abstract and clear framework architecture, precisely the definition of the pools and layers, it becomes easy to arrange the requirements of applications in a loosely coupled way. The separation of the miscellaneous logic aspects into pools facilitates the applications development. More importantly, the concepts of the framework are implemented with Java programming language. The availability of the software tool of the framework helps to increase the developers productivity.

We have used the proposed framework to develop the application of microgrid’s system. The case study helped to show the suitability of the intelligent and collaborative requirements development as well as the secure data exchange in distributed systems.

In future works, we aim to apply the concepts of the framework to more case studies also we plan to

improve the quality of code of the software tool. Another concern is to integrate more advanced analytic techniques to cover more logic aspects.

REFERENCES

- Aid, A. and Rassoul, I. (2017). Context-aware framework to support situation-awareness for disaster management. *International Journal of Ad Hoc and Ubiquitous Computing*, 25(3):120–132.
- Alhamid, M. F., Rawashdeh, M., Dong, H., Hossain, M. A., Alelaiwi, A., and El Saddik, A. (2016). Recam: a collaborative context-aware framework for multimedia recommendations in an ambient intelligence environment. *Multimedia Systems*, 22(5):587–601.
- Anvaari, M., Cruzes, D. S., and Conradi, R. (2012). Smart grid software applications as an ultra-large-scale system: Challenges for evolution. In *2012 IEEE PES Innovative Smart Grid Technologies (ISGT)*, pages 1–6. IEEE.
- Bucchiarone, A., Marconi, A., Pistore, M., and Raik, H. (2017). A context-aware framework for dynamic composition of process fragments in the internet of services. *Journal of Internet Services and Applications*, 8(1):6.
- Dabbaghjamanesh, M., Wang, B., Mehraeen, S., Zhang, J., and Kavousi-Fard, A. (2019). Networked microgrid security and privacy enhancement by the blockchain-enabled internet of things approach. In *2019 IEEE Green Technologies Conference (GreenTech)*, pages 1–5. IEEE.
- El-Naily, N., Saad, S. M., and Mohamed, F. A. (2020). Novel approach for optimum coordination of overcurrent relays to enhance microgrid earth fault protection scheme. *Sustainable Cities and Society*, 54:102006.
- Elgamal, M., Korovkin, N., Elmitwally, A., and Chen, Z. (2020). Robust multi-agent system for efficient online energy management and security enforcement in a grid-connected microgrid with hybrid resources. *IET Generation, Transmission & Distribution*, 14(9):1726–1737.
- Ferdowsi, F., Dabbaghjamanesh, M., Mehraeen, S., and Rastegar, M. (2019). Optimal scheduling of reconfigurable hybrid ac/dc microgrid under dlr security constraint. In *2019 IEEE Green Technologies Conference (GreenTech)*, pages 1–5. IEEE.
- Fkaier, S., Khalgui, M., and Frey, G. (2020a). Hybrid context-awareness modelling and reasoning approach for microgrid’s intelligent control. In *Proceedings of the 15th International Conference on Software Technologies - Volume 1: ICSoft*, pages 116–127. INSTICC, SciTePress.
- Fkaier, S., Khalgui, M., and Frey, G. (2020b). Meta-model for control applications of microgrids. In *2020 6th IEEE International Energy Conference (ENERGY-Con)*, pages 945–950. IEEE.
- Fkaier, S., Romdhani, M., Khalgui, M., and Frey, G. (2016a). Enabling reconfiguration of adaptive control systems using real-time context-aware framework. In *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, pages 1–8. IEEE.
- Fkaier, S., Romdhani, M., Khalgui, M., and Frey, G. (2016b). R2tca: New tool for developing reconfigurable real-time context-aware framework—application to baggage handling systems. In *Proc. Int. Conf. Mobile Ubiquitous Comput., Syst., Services Technol.(UBICOMM)*, pages 113–119.
- Fkaier, S., Romdhani, M., Khalgui, M., and Frey, G. (2017). Context-awareness meta-model for reconfigurable control systems. In *Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE*, pages 226–234. INSTICC, SciTePress.
- Ghosh, P., Eisele, S., Dubey, A., Metelko, M., Madari, I., Volgyesi, P., and Karsai, G. (2020). Designing a decentralized fault-tolerant software framework for smart grids and its applications. *Journal of Systems Architecture*, 109:101759.
- Henri, G., Levent, T., Halev, A., Alami, R., and Cordier, P. (2020). pymgrid: An open-source python microgrid simulator for applied artificial intelligence research. *arXiv preprint arXiv:2011.08004*.
- Hosseinimoghadam, S. M. S., Dashtdar, M., Dashtdar, M., and Roghanian, H. (2020). Security control of islanded micro-grid based on adaptive neuro-fuzzy inference system. *Scientific Bulletin”: Series C Electrical Engineering and Computer Science*, (1):189–204.
- Hubana, T. (2020). Artificial intelligence based station protection concept for medium voltage microgrids. In *2020 19th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–6. IEEE.
- Khan, S., Paul, D., Momtahan, P., and Aloqaily, M. (2018). Artificial intelligence framework for smart city microgrids: State of the art, challenges, and opportunities. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 283–288. IEEE.
- Mbungu, N. T., Naidoo, R. M., Bansal, R. C., and Vahidinassab, V. (2019). Overview of the optimal smart energy coordination for microgrid applications. *IEEE Access*, 7:163063–163084.
- Sikder, A. K., Aksu, H., and Uluagac, A. S. (2019). A context-aware framework for detecting sensor-based threats on smart devices. *IEEE Transactions on Mobile Computing*, 19(2):245–261.
- Tang, L., Yu, Z., Wang, H., Zhou, X., and Duan, Z. (2014). Methodology and tools for pervasive application development. *International Journal of Distributed Sensor Networks*, 10(4):516432.
- Wu, Y., Guerrero, J. M., and Wu, Y. (2019). Distributed coordination control for suppressing circulating current in parallel inverters of islanded microgrid. *IET Generation, Transmission & Distribution*, 13(7):968–975.