International Journal of Control and Automation
Vol.7, No.1 (2014), pp.153-170
http://dx.doi.org/10.14257/ijca.2014.7.1.13

A Novel Hierarchical Multi-Agent Architecture for Automatic Restoration of Smart Grids

Zhongyuan Jiang[1], Mohamed Khalgui[2], Olfa Mosbahi[3] and Aymen Jaouadi[3]

[1]*School of Electro-Mechanical Engineering, Xidian University, Xi'an 710071, China*
[2]*Institute of Computer Science, Martin Luther University, Halle, Germany*
[3]*Institute of Computer Science, Tunis El-Manar University, Tunis, Tunisia*

jiangzy0708@gmail.com

Abstract

This paper proposes a novel hierarchical multi-agent architecture for the automatic fault detection and restoration of smart grids based on traditional power transmission systems that are composed of a lot of electric substations (ESs). Each agent is associated with an ES to control the ES. If a fault occurs in the input lines of an ES, the agent corresponding to the ES can automatically detect the fault and choose a solution from other agents. The ES of the chosen agent can supply power to the faulty ES by closing a corresponding electric switch and then the fault is restored. Furthermore, a coordination protocol is proposed to allow the fault restoration. Moreover, the agents are formally modeled by timed automata and the coordination protocol is verified by the model checker UPPAAL. Finally, the contribution of this paper is applied to a real-world system.

Keywords: *Smart Grids, Multi-Agents, Fault Restoration, Timed Automata, Verification*

1. Introduction

A traditional power transmission system (TPTS) usually meets a lot of sudden events [1]. Large area blackouts (LABs) are the most severe accidents, which are usually caused by serious internal or external damages, such as the interruption of transmission lines. It is estimated that the yearly economic loss in U.S. is $25~$180 billion because of the LABs and power quality disturbances [2]. In 2003, the U.S.-Canadian LAB affected approximately 50 million people [3]. In order to improve the reliability of TPTSs such that the TPTSs become more intelligent, smart grids are developed [4, 5].

A smart grid is an intelligent energy and information switching system. Self-healing is one of the most important characteristics for smart grids [6]. If a fault occurs, it must be automatically detected and restored to avoid a LAB. The traditional methods to deal with faults include three steps: fault detection, location, and restoration. In [7], a method that is used to detect and localize faults in smart grids is presented based on a designed Petri net model. The faults can be computed by means of incidence matrix operations. However, the Petri net model and fault computation are complex and inefficient for a fault because of the scale of large smart grids. The fault restoration is also neglected.

A multi-agent system is a combination of several networked software agents working in collaboration to achieve an overall goal [8]. It is an ideal method to use multi-agent systems to implement smart grids based on TPTSs [9]. Barn *et al*., [10] describe how a multi-agent

ISSN: 2005-4297 IJCA

system can be used to implement the power support. Catterson *et al*., [11] identify five key agents within a smart grid, which offer templates for the behavior and interaction of agents.

Momoh [12] presents a multi-agent architecture to perform the fault detection and restoration for Navy ship. The architecture consists of three kinds of agents: a number of bus agents, a number of load agents, and a negotiating agent. It can effectively restore the faults but only for a simplified ship system. In [4], the authors discuss the implementation of a multi-agent system that is used to perform the fault restoration of a smart grid. The multi-agent system can promote the emergency supply for a smart grid. Xu and Liu [13] propose a multi-agent architecture to implement the fault restoration in micro-grids. Each bus in a micro-grid is assigned with a node agent and each agent makes synchronized fault restoration decisions. The authors of [14] report a principle for a multi-agent coordination to achieve the fault restoration of a micro-grid via wireless communication. The existing architectures are useful but complex and clumsy due to a large number of agents in large scale smart grids. Moreover, they do not differentiate agents for high, medium, and low voltage lines.

Vyatkin *et al*., [15, 16] formulate a multi-agent architecture to achieve fault restoration with a number of bus agents and a single facilitator agent. A bus agent tries to find sub-optimal target configuration after a fault occurs by coordinating with other peers. The models of agents are performed by the form of international standards IEC 61850/61499. However, the developed coordination protocols are useful but do not consider the efficiency. The optimal numbers of communication lines and exchanged messages among agents are not considered. Especially, the synchronous communication in large scale smart grids is neglected. Finally, the existing studies consider neither any formalization nor verification of their architectures.

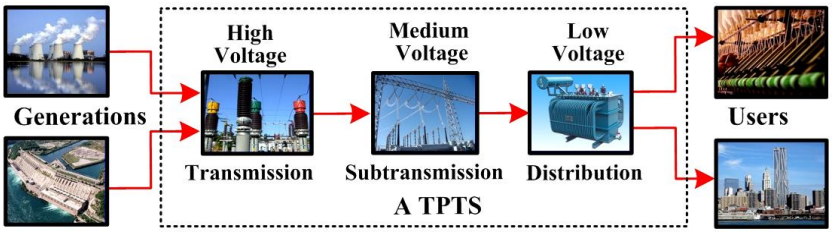

Figure 1. A TTPS

In this paper, a TPTS is divided into three layers: transmission, subtransmission, and distribution, as shown in Figure 1. The three layers are composed of high, medium, and low voltage electric substations (ESs), respectively [17]. Each ES has input and output lines. In order to perform the emergency power supply, each ES is connected with adjacent ESs via emergency lines and electric switches that are opened in initial states. If a fault occurs in the input lines of an ES, a connected ES of the faulty ES can provide power by closing a corresponding electric switch. Figure 2 shows a part of the TPTS of West Lake/North Urban Centers in Tunis (Tunisia), which is a typical TPTS. Some emergency lines and related electric switches are also shown in Figure 2. For example, the two ESs "Tunisia Leasing" and "Export House" are connected with an ES "INSAT1" since the available load (it can be supplied to other ESs to restore faults) of "INSAT1" is 5100KW and the rated loads (it is the minimum loads for normal operation) of "Tunisia Leasing" and "Export House" are 2400KW and 2670KW, respectively. Therefore, if the input lines of the two ESs "Tunisia Leasing" and "Export House" simultaneously break down, the ES "INSAT1" can provide power to the two faulty ESs since $5100 > (2400 + 2670) = 5070$. Then, the two electric switches "Act5" and "Act6" will be temporarily closed.

Moreover, this paper proposes a hierarchical multi-agent architecture to perform the automatic fault detection and restoration. Each agent is associated with an ES. If a fault occurs in the input lines of the ES, the associated agent of the ES will automatically detect the fault and look for a connected ES to restore the fault. In the architecture, the agents are classified into three categories: High-Agents, Medium-Agents, and Low-Agents according to the three layers of ESs. All High-Agents belong to the topmost layer and each High-Agent is in the upper layer of a set of Medium-Agents. Similarly, each Medium-Agent is in the upper layer of a set of Low-Agents. All Low-Agents belong to the bottommost layer.

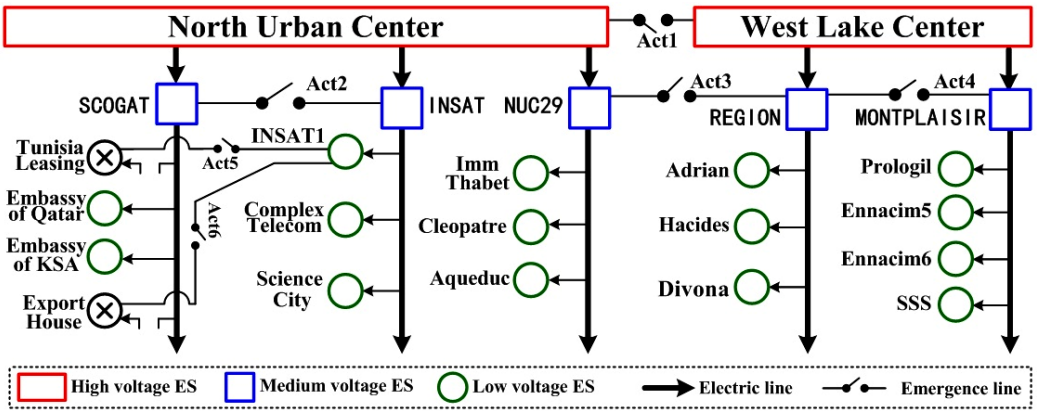

Figure 2. A part of West Lake/North Urban Center in Tunis

Furthermore, all agents are coordinated by a proposed coordination protocol that provides a way to perform fault restoration and information switching. In the worst case, comparing with the developed protocols in [15] and [16] and a general coordination protocol (each agent directly communicate with its solutions), the number of communication lines is reduced about 67% in large scale smart grids.

To formally describe and verify the properties of the whole agent-based architecture and the reliability of automatic fault restoration, timed automata [18] are used to model the three types of agents and UPPAAL that is a model checker [19] is applied to verify the correctness of their reactions by performing the coordination protocol with a case study.

The rest of this paper is structured as follows. Section 2 proposes a hierarchical multi-agent architecture and Section 3 presents a coordination protocol. The simulation and verification with timed automata and UPPAAL are illustrated in Section 4. Finally, Section 5 concludes this paper.

2. Hierarchical Multi-agent Architecture

In this section, a hierarchical multi-agent architecture is proposed to automatically detect and restore the faults, which is composed of three types of agents.

2.1. Fault detection and restoration

A TPTS is composed of a lot of ESs that can be hierarchically divided into three layers: high, medium, and low voltage ESs. Each ES has the input and output lines. In order to automatically detect and restore the faults that occur in the input lines of ESs, each ES is associated with an agent that is composed of three units: a monitor, a restorer, and a database. The three units are illustrated as follows.

- Monitor: It has an electric current sensor that is linked with the input lines of this ES. If the input lines break down, the electric current sensor is triggered since the electric current is disappeared. Then, a fault is detected. The monitor sends a request message to the restorer.
- Restorer: If it receives a request message, it tries to choose a different agent. The ES of the chosen agent can provide power to restore the fault.
- Database: It is used to record the information, such as the input load, output load, available load, and other information.

Therefore, if a fault occurs in the input lines of an ES, it can be automatically detected by the associated agent of the ES. The associated agent tries to find a different agent to restore the fault. All information is saved in the internal database of the associated agent. Specially, if faults occur on the electric current sensors, the faults can be detected by enhancement of a parity space approach [20, 21].

All agents are classified to three categories: High-Agents (HAs), Medium-Agents (MAs), and Low-Agents (LAs) according to the three layers of ESs. The HAs, MAs, and LAs are associated with the high, medium, and low voltage ESs, respectively. Let N be a set of ESs and A be a set of agents. Two functions are defined as follows.

- $Agent: N \to A$ is a function. Let $Agent(e)$ denote the associated agent a of e, where $e \in N$ and $a \in A$.
- $Agent^{-1}: A \to N$ is a reverse function of $Agent$. Let $Agent^{-1}(a)$ denote the ES e that is associated with a, where $e \in N$ and $a \in A$.

2.2. Parameters of ESs and agents

Let a be an agent. a contains the following four parameters.

1) **ID:** It is the unique address composed of a prefix, i.e., "HA", "MA", or "LA", and serial numbers, such as "HA1", "MA11", and "LA111". If a fault is detected by the internal monitor of an agent, then the fault is located simultaneously since the ID of the agent is unique.

2) **Supply Load (SL):** It is the maximum permitted input load of $Agent^{-1}(a)$, denoted as $a_{(SL)}$, which is provided from an upstream ES.

3) **Rated Load (RL):** It is the minimum load for the normal operation of $Agent^{-1}(a)$, denoted as $a_{(RL)}$, which is transmitted to the downstream ESs.

4) **Available Load (AL):** It is the available load of $Agent^{-1}(a)$, denoted as $a_{(AL)}$. It can be supplied to other ESs to restore faults and avoid LABs.

Then, the power balance of $Agent^{-1}(a)$ is defined as follows.

$$a_{(AL)} = a_{(SL)} - a_{(RL)} \quad \textbf{(1)}$$

2.3. Solution and actuator

To avoid LABs in a TPTS, any ES e may be previously connected with some other ESs via emergency lines and electric switches that are controlled by actuators (software controllers). In initial states, all actuators are opened. The associated agents of the connected ESs are called solutions of $Agent(e)$. If a fault occurs in the input lines of e, one of the solutions of $Agent(e)$ will be selected. The ES of the selected solution can provide power to e by closing an actuator.

Let a be an agent, a fault occur in the input lines of $Agent^{-1}(a)$, and s be a solution of a. s is said to be an available solution of a if

$$s_{(AL)} \geq a_{(RL)} \tag{2}$$

Eq. (2) means that the ES of an available solution has enough power to restore the fault. In other words, the solutions for any agent is previously computed and stored in the database according to Eqs. (1) and (2) since the emergency lines must be previously designed and laid out in the TPTSs.

Let a_1, a_2, $\cdots$, and a_m be m agents. If there are m faults that occur in the input lines of $Agent^{-1}(a_1)$, $Agent^{-1}(a_2)$, $\cdots$, and $Agent^{-1}(a_m)$, respectively, and s is their same solution, the m faults can be restored by s if

$$s_{(AL)} \geq \sum_{i=1}^{m} a_{i(RL)} \tag{3}$$

2.4. Hierarchical multi-agent architecture

A hierarchical multi-agent architecture based on the TPTSs can be defined as $L = ((N_H \cup N_M \cup N_L), (A_H \cup A_M \cup A_L), E, C, F)$, where N_H, N_M, N_L, A_H, A_M, and A_L are finite and non-empty with $(N_H \cup N_M \cup N_L) \cap (A_H \cup A_M \cup A_L) = \varnothing$,

- N_H, N_M, N_L are the sets of high, medium, and low voltage ESs, respectively,
- A_H, A_M, A_L are the sets of HAs, MAs, and LAs, respectively,
- $E \subseteq (N_H \times N_M) \cup (N_M \times N_L)$ is an electric current relation, graphically represented by arcs with single arrows,
- $C \subseteq (A_H \times A_M) \cup (A_M \times A_L) \cup (A_H \times A_H) \cup (A_M \times A_M) \cup (A_L \times A_L)$ is a communication relation, graphically represented by dotted arcs with double arrows, and
- $F \subseteq (N_H \times N_H) \cup (N_M \times N_M) \cup (N_L \times N_L)$ is a solution relation, graphically represented by arcs with actuators.

For example, a hierarchical multi-agent architecture is shown in Figure 3, which is based on the part of Figure 2. It contains two HAs ("North Urban Center" and "West Lake Center" that are associated with HA1 and HA2, respectively), three MAs ("SCOGAT", "INSAT", and "Region" that are associated with MA11, MA12, and MA21, respectively), and nine LAs (the downstream ESs of the three MAs). HA1 and MA11 are an electric current relation. HA1 and HA2 are a communication relation.

LA111 and LA121 are a solution relation. If a fault is detected by LA111, its solution LA121 will be required to close Act5 to restore the fault and vice versa.

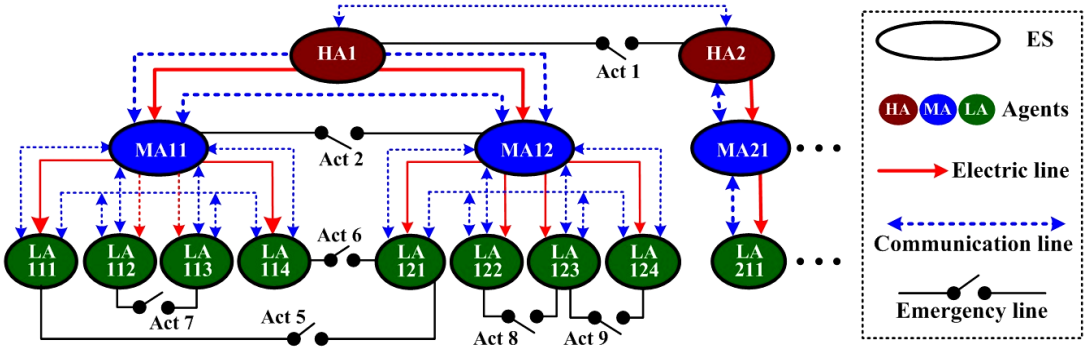

Figure 3. A hierarchical multi-agent architecture

The relation between any two agents in architecture L is defined as follows.

- **Parent/Descendant:** Let $a_1, a_2 \in (A_H \cup A_M \cup A_L)$ be two agents. The electric current is streamed from $Agent^{-1}(a_1)$ to $Agent^{-1}(a_2)$ directly. Then, a_1 is called the parent of a_2 and a_2 is called a descendant of a_1. For example, HA1 is the parent of MA11 in Figure 3.
- **Root/Leaf:** Let $a_1, a_2, a_3 \in (A_H \cup A_M \cup A_L)$ be three agents, a_2 be the parent of a_1, and a_3 be the parent of a_2. Then, a_3 is called the root of a_1 and a_1 is called a leaf of a_3. In Figure 3, HA1 is the root of LA111 and LA111 is a leaf of HA1.
- **Brother:** Any two agents that have a same parent are called brothers. Specially, any two HAs are brothers. In Figure 3, LA111 and LA114 are brothers.
- **Neighbor:** Any two LAs that have a same root and different parents are called neighbors. Any two MAs that have different parents are also called neighbors. All HAs do not have neighbors. All neighbors belong to a same HA. For example, LA111 and LA121 are neighbors and belong to HA1 in Figure 3.
- Friend: Any two LAs that have different roots are called friends. For example, LA111 and LA211 are friends in Figure 3. Only LAs have friends and all friends belong to different HAs.

In order to reduce the communication lines, the communication rules in the proposed architecture L are defined as follows.

- $\forall a \in A_H$, a can directly communicate with all other HAs (its brothers) and its descendants;
- $\forall a \in A_M$, a can directly communicate with its brothers, descendants, and parent;
- $\forall a \in A_L$, a can directly communicate with its brothers and its parent.

In Figure 3, HA1 can directly communicate with HA2 and MA11. MA11 can directly communicate with HA1, MA12, and LA111. LA111 can directly communicate with MA11 and LA112.

3. Coordination protocol

$\forall a \in (A_H \cup A_M \cup A_L)$, all solution of a are divided into brother, neighbor, and friend solutions according to the relations of brothers, neighbors, and friends, respectively. All the solutions of a are saved in the databases as follows.

- Its all brother solutions are saved in its database as a list.
- If $a \in A_H$, a does not have any neighbor solution. Otherwise, its all neighbor solutions are saved in its parent's database as a list. The goal is to reduce the number of exchanged messages since all communication paths between a and its neighbor solutions should contain their parents that can be considered as management departments. If they ask for power supplying, then it is necessary to interconnect with each other.

If $a \in A_H$, its all friend solutions are saved in its root's database as a list. Otherwise, it does not have any friend solution. The goal is similar with neighbor solutions since they also have different roots that can be considered as the higher management departments.

3.1. Coordination protocol

In order to coordinate agents and solutions to perform fault restoration and information switching, a coordination protocol is proposed. First, three types of messages that are used to switch information among agents are illustrated as follows.

1) **Request(FID):** This message is only used to inform FID, the parent of FID, or the root of FID that there is a fault in the ES of FID, where FID is an agent. The receiver of this message will try to find a brother, a neighbor, or a friend solution for FID.
2) **Close(MsgID, FIDS, SID):** It is used to inform a solution to close the corresponding actuators. Then, the ES of the solution can supply power to the faulty ESs. MsgID is the sender (an agent) of this close message. FIDS is a set of agents. $\forall f_i \in$ FIDS, there is a fault that occurs in the input lines of $Agent^{-1}(f_i)$. Moreover, all agents in FIDS have a same solution SID.
3) **Return(FIDS, SLS):** It is used to reply the restoration results for the agents in FIDS. SLS is a set of the corresponding restoration results. $\forall f_i \in$ FIDS, $d_i \in$ SLS is the restoration result of f_i. If $d_i \neq 0$, f_i is restored. Otherwise, f_i is not restored.

Coordination protocol: The coordination protocol is composed of two algorithms. Let $a \in (A_H \cup A_M \cup A_L)$ be an agent. If a receives a request message, Algorithm 1 is executed to find a solution and send a close message to the solution. If a receives a close message, Algorithm 2 is executed to open related actuators and reply a return message. If the input lines of a faulty ES are repaired and the repaired ES can supply power normally as the initial state, the agent of the repaired ES informs its available solution to open the related actuator to stop the emergency power supply.

Algorithm 1 If current agent a simultaneously receives $m(m \geq 1)$ request(f_i) messages, a tries to find solutions and send close messages to the solutions to restore the m faults, where $f_i \in (A_H \cup A_M \cup A_L)$ is a (the request message is sent by the monitor of a), a descendant of a , or a leaf of a , $i = 1, 2, \cdots, m$.

1: **for** each agent f_i of the m messages **do**
2: a tries to find a solution s_i for f_i (*Perhaps there are some agents that have a same solution*);
3: **if** a cannot find a solution for f_i **then**
4: **if** $a \in (A_M \cup A_L)$ **then**
5: a sends a request(f_i) message to its parent (*the same protocol is executed by the parent*);
6: **else**
7: a asks for a manual restoration for f_i ;
8: **end if**
9: **end if**
10: **end for**
//The following lines 12 and 13 are a message manager (MM). It is used to combine //several close messages that are sent to a same solution to a close message.
11: **for** each solution s_i of the m agents **do**
12: a creates a agent set **FIDS**;
13: a puts $k(1 \leq k \leq m)$ agents into the FIDS according to FIFO strategy if the k agents have a same solution s_i ;
14: a sends a **close(** a **, FIDS,** s_i **)** message to s_i through the root or the parent of s_i ;
15: a is waiting for a return message within a given time (*if the time is up, a will resend the close message*);
//The following lines 16~24 are used to handle the return message according to //the parameters of FIDS and SLS.
16: **if** a receives a **return(FIDS, SLS)** message **then**
17: **for** each agent $f_i \in$ FIDS and $d_i \in$ SLS **do**
18: **if** $d_i == 0$ **then**
19: f_i is not restored. a will try to find another solution for f_i ;
20: **else**
21: f_i is restored;
22: **end if**
23: **end for**
24: **end if**
25: **end for**

Algorithm 2 If current agent a receives a close(MID, FIDS, SID) message, it tries to close the related actuators and reply a return message.

1: a puts this close message into an FIFO line that may contain other close messages;
2: a takes a **close(MID, FIDS, SID)** message;
3: **if** $a \neq$ SID **do**
4: a is not the solution and a delivers the close message to a related descendant according to SID (*the same protocol is executed by the descendant*);
5: **else**
6: a is the solution and a creates a set **SLS** that is used to record the corresponding restoration results of the agents in FIDS;
7: **for** each f_i in FIDS **do**
8: **if** ($a_{(AL)} \geq f_{i(RL)}$) and (a successfully closes the actuator according to f_i) **then**
9: $a_{(AL)} = a_{(AL)} - f_{i(RL)}$;
10: $d_i = f_{i(RL)}$; //*Fault is restored*.
11: **else**
12: $d_i = 0$; //*Fault is not restored*;
13: **end if**
14: a puts d_i into SLS;
15: **end for**
16: a replies a **return(FIDS, SLS)** message to MID according to the close message;
17: **end if**

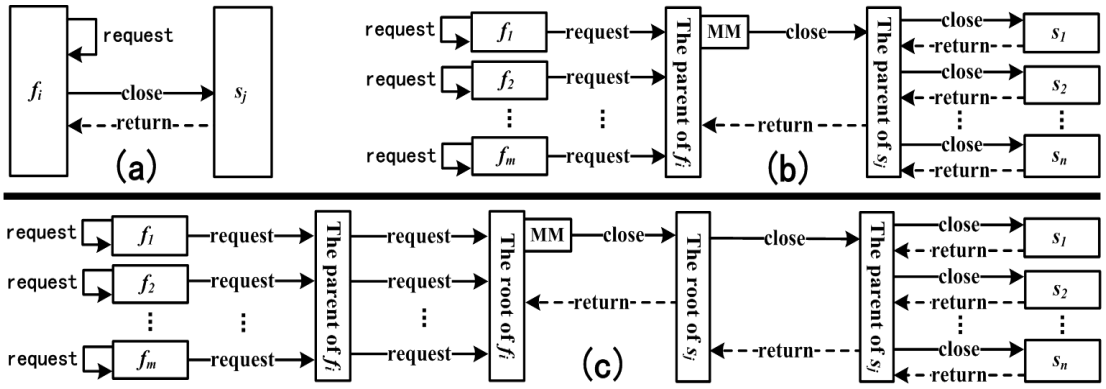

Figure 4. Message exchange processes between agents and their (a) brother, (b) neighbor, and (c) friend solutions

In this coordination protocol, if a receives a request message, then a tries to directly contact its brother solutions. The message exchange processes are shown in Figure 4(a). Moreover, if the fault cannot be restored by its brother solutions and $a \in (A_M \cup A_L)$ is true, then a delivers the request message to its parent since the neighbor solutions of a are recorded in its parent. The parent tries to contact the neighbor solutions of a. The message

exchange processes are shown in Figure 4(b). Furthermore, if the fault cannot be restored by the neighbor solutions of a and $a \in A_L$ is true, then the parent delivers the request message to the root of a since the friend solutions of a are recorded in the root. The root tries to contact the friend solutions of a. The message exchange processes are shown in Figure 4(c). Finally, if all solutions of a cannot restore the fault, then a manual restoration should be applied. This coordination protocol is independently executed by any agent.

Steps 12 and 13 in Algorithm 1 are a message manager (MM). If there are $k(k \geq 1)$ close messages that should be simultaneously sent to a same solution, then the k close messages will be merged into a close message. The merged close message with attached information (it records the information of the k close messages) is sent to the solution. The detailed processes are illustrated in Example 1.

3.2. Example 1

In Figure 2, the two faulty ESs "Tunisia Leasing" and "Export House" are associated with LA111 and LA114, respectively, and the ES "INSAT1" is associated with LA121 that is a neighbor solution for LA111 and LA114, as shown in Figure 3.

If two faults are detected by the two agents LA111 and LA114, the two agents simultaneously send two request messages, request(LA111) and request(LA114), to their parent MA11 since they do not have brother solutions. After MA11 receives the two request messages, it tries to find a neighbor solution for the two faults, respectively, in which they have a same neighbor solution LA121. Then, MA11 prepares to simultaneously send two close messages to LA121. However, the two close messages are merged into a close message, close(MA11, {LA111, LA114}, LA121), since they have a same destination. Therefore, MA11 only sends the merged close message to LA121 through MA12. After LA121 receives the close message, it tries to close the two actuators Act5 and Act6 according to the second parameter ({LA111, LA114}) in the merged close message. Therefore, LA121 replies a return message, return({LA111, LA114}, {2400, 2670}), to MA11 through MA12. Finally, the two faults are restored since $d_1 = 2400 > 0$ *and* $d_2 = 2670 > 0$.

In this example, two close and two return messages are optimized to a close and a return messages because of the message manager.

3.3. Complexity analysis

Let f_1, f_2, $\cdots$, and $f_m \in (A_H \cup A_M \cup A_L)$ and s_1, s_2, $\cdots$, and $s_n \in (A_H \cup A_M \cup A_L)$ be m and n different brothers, respectively, and $\forall f_i (i = 1, 2, \cdots, m)$, s_1, s_2, $\cdots$, and s_n are the solutions of f_i. If m faults simultaneously occur in the ESs of f_1, f_2, $\cdots$, and f_m, then f_1, f_2, $\cdots$, and f_m simultaneously contact s_j to restore the faults, where j sequentially takes $1, 2, \cdots$, and n until the faults are restored since s_1, s_2, $\cdots$, and s_n are restored in a database as a list. In the worst case, the m faults are simultaneously restored by s_n. Let $\beta_{NCL}^{m \sim n}$ be the number of communication lines and $\beta_{NEM}^{m \sim n}$ be the number of exchanged messages between the m faulty agents and their n solutions. Then, $\beta_{NCL}^{m \sim n}$ and $\beta_{NEM}^{m \sim n}$ for the three types of solutions, i.e., brother, neighbor, and friend solutions, are computed as follows.

Case 1: $\forall f_i (i = 1, 2, \cdots, m)$, if s_1, s_2, $\cdots$, and s_n are brother solutions of f_i, f_i directly contacts s_j with close and return messages to restore the faults, as shown in Figure 4(a). Therefore, we have

$$\beta_{NCL}^{m\sim n} = m \times n = mn \text{ and } \beta_{NEM}^{m\sim n} = m \times (2 \times n) = 2mn$$

Case 2: $\forall f_i (i = 1, 2, \cdots, m)$, if s_1, s_2, $\cdots$, and s_n are brother solutions of f_i, f_i sends a request message to its parent. Therefore, the parent will receive m request messages and there are m close messages that should be sent by the parent. However, the parent merges the m close messages into a close message and sends this merged close message to s_j. Finally, s_j replys a return message to the parent, as shown in Figure 4(b). We have

$$\beta_{NCL}^{m\sim n} = m + 1 + n = m + n + 1 \text{ and } \beta_{NEM}^{m\sim n} = m + (4 \times n) = m + 4n$$

Case 3: $\forall f_i (i = 1, 2, \cdots, m)$, if s_1, s_2, $\cdots$, and s_n are brother solutions of f_i, f_i sends a request message to their parent and the parent delivers the request message to the root of f_i. Therefore, the root receives m request messages and only a merged close message is sent to s_j to restore the faults, as shown in Figure 4(c). We have

$$\beta_{NCL}^{m\sim n} = m + 1 + 1 + 1 + n = m + n + 3 \text{ and } \beta_{NEM}^{m\sim n} = m + m + (6 \times n) = 2m + 6n$$

In a general coordination protocol, each agent directly contacts a solution with a close message and the solution directly replies a return message to the agent. We have $\beta_{NCL}^{1\sim 1} = 1$ and $\beta_{NEM}^{1\sim 1} = 2$. If the m faults and the n solutions are considered by this protocol, we have $\beta_{NCL}^{m\sim n} = mn$ and $\beta_{NEM}^{m\sim n} = 2mn$. In [15] and [16], we have $\beta_{NCL}^{1\sim 1} = 2$ and $\beta_{NEM}^{1\sim 1} = 5$. For the m faults and the n solutions, we have

$$\beta_{NCL}^{m\sim n} = 2mn \text{ and } \beta_{NEM}^{m\sim n} = 5mn$$

Assume that $m = n$. The advantage of the proposed coordination protocol is shown in Figure 5. We can see that the $\beta_{NCL}^{m\sim n}$ and $\beta_{NEM}^{m\sim n}$ in the protocol proposed in this paper are smaller than the $\beta_{NCL}^{m\sim n}$ and $\beta_{NEM}^{m\sim n}$ in other protocols.

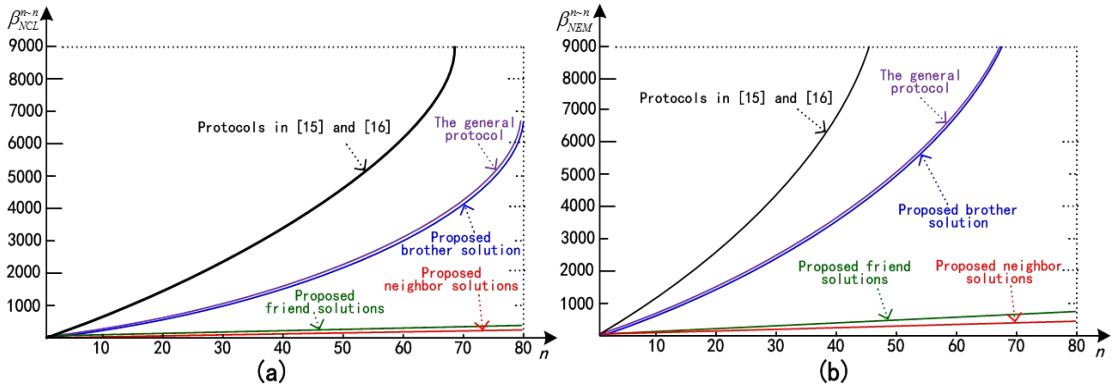

Figure 5. Comparison: (a) $\beta_{NCL}^{m\sim n}$ and (b) $\beta_{NEM}^{m\sim n}$

Suppose that there be two faults that simultaneously occur in the input lines of two ESs, the two agents of the two faulty ESs simultaneously compete a same solution to restore the faults. If the two agents are brothers, they are putted into a set FIDS according to first in first out (FIFO) strategy (step 13 in Algorithm 1). Otherwise, they send two request messages to the same solution and the two request messages are putted into an FIFO line (step 1 in Algorithm 2). Therefore, the two faults will be handled according to the FIFO strategy by the same solution. If the two faults asynchronously occur or they have different solutions, the two agents communicate with their solutions, independently.

4. Verification with Timed Automata

To verify the automatic restoration characteristic of the proposed architecture, the states and behavior of agents are formally described by timed automata [18]. Moreover, the correctness of the coordination protocol is verified by UPPAAL [19].

4.1. Timed automata model of agents

The automata formalism [22] is extended to timed automata by Alur and Dill in [18] with a finite set of real-valued clocks. UPPAAL [19] is an integrated tool for modeling, validation, and verification of complex systems based on timed automata. UPPAAL is applied to a number of industrial case studies, such as web service, power down protocol, and mutual exclusion protocol. The details of timed automata and UPPAAL can be found in [18] and [19].

The timed automata model of the agents is shown in Figure 6. It contains three messages (declared as "chan Request[150], Close[150], Return[150];"), eleven states (illustrated in Table 1), and two integer variables *pIndex* and *ActNum* (*pIndex* is the number of received request messages and *ActNum* is the number of agents that should be restored by the same solution in a close message).

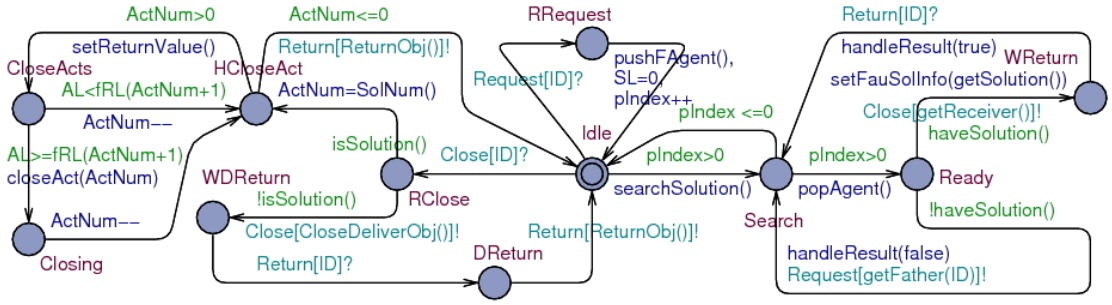

Figure 6. The timed automata model of agents

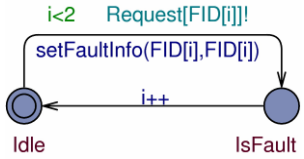

Figure 7. A fault injector

At state "*Idle*" in Figure 6, the current agent can receive two messages (a request message and a close message). At state "*WReturn"*, it can receive a return message. Moreover, in order

to inject faults to simulate and verify the coordination protocol, an additional model is given, as shown in Figure 7, where i is an index (in initial state, $i=0$) and FID is an array used to record the IDs of agents (their associated ESs are injected faults). For example, "const int FID[2]={111,114};" means that LA111 and LA114 will be injected a fault, respectively.

Table 1. The illustration of all states

States	Illustration
Idle	Inactivity state.
RRequest	It receives a request message.
Search	It tries to look for a solution for any agent.
Ready	It prepares to handle an agent. If the agent has a solution, it sends a close message to the solution. Otherwise, it sends a request message to its parent.
WReturn	A close message is sent. It is waiting for a return message.
RClose	It receives a close message.
HCloseAct	This agent is the solution and it prepares to close related actuators.
CloseActs	It is comparing the AL of current agent with the RL of an agent in the received close message.
Closing	It is closing an actuator at this time.
WDReturn	This agent is not the solution. It delivers the received close message to a corresponding descendant and it is waiting for a return message.
DReturn	It receives a return message from the descendant.

4.2. Example 2

In this subsection, Example 1 is simulated with the simulator of UPPAAL. First, a fault injector and five agents are declared with the designed timed automata models as follows.

faultInjector=FaultInjector(); *MA11=Agent(11,8000,5000,3000);*
MA12=Agent(12,30000,15300, 14700); *LA111=Agent(111,5400,2400, 3000);*
LA114=Agent(114,5170,2670, 2500); *LA121=Agent(121,8300,3200, 5100);*

In Figure 7, if the state of faultInjector is changed from "Idle" to "IsFault", then a request message is sent and LA111 receives the request message (LA111.pIndex=1). After the state is changed from "IsFault" to "Idle", we have $i=1$. *If the state of faultInjector is changed again since* $i=1<2$, *then LA114 will receive a request message (LA114.pIndex=1). Therefore, the states of LA111 and LA114 are changed from "Idle" to "RRequest" and changed back to "Idle". The two agents try to find a brother solution by calling the function "searchSolution()". Their states are changed to "Search", respectively. However, they do not have any brother solution at state "Ready". Then, they send a request message to their same parent MA11, respectively. Their states are changed back to "Idle" through "Search".*

After MA11 receives the two request messages, we have MA11.pIndex=2. Its state is changed from "Idle" to "Search" since MA11.pIndex>0. It calls the function "searchSolution()" to look for a neighbor solution LA121 for the two agents. MA11 merges the information of LA111 and LA114 into a close message since they have a same solution. Then, MA11 sends this close message to LA121 through MA12.

The function "isSolution()" returns true after LA121 receives the close message. The state is changed to "HCloseAct" and LA121.ActNum=2 (LA111 and LA114). LA121 tries to close the two actuators Act5 and Act6 since $5100>(2400+2670)$. *Moreover, LA121 assigns* $SLS_{LA111}=2400$ *and* $SLS_{LA114}=2670$ *and replies a return message to MA11 through MA12. Finally, the two faults are restored by LA121 since* $SLS_{LA111}=2400>0$ *and* $SLS_{LA114}=2670>0$.

According to the simulation, LA111 and LA114 can be restored by their neighbor solution LA121. It includes $11\times5=55$ states, where 22 states are reachable.

4.3. Verification with UPPAAL

UPPAAL is a toolbox for the verification of real-time systems with its verifier via the following properties [19].

- *E* < > *p*: There exists a path such that p eventually holds.
- *A [ ] p*: For all paths, p always holds.
- *E [ ] p*: There exists a path where p always holds.
- *A* < > *p*: For all paths, p will eventually hold.
- $p \rightarrow q$: Whenever p holds, q will eventually hold.

Note that p and q are state formulas, such as LA111.RRequest and LA111.pIndex=2. Therefore, the correctness of the coordination protocol is verified by using UPPAAL based on Example 2 via the following properties.

P1: *E* <> LA111.RRequest and LA114.RRequest. There exists a path such that the two states of LA111 and LA114 are "RRequest". If this property is satisfied, then two faults occur in the ESs of LA111 and LA114 and the two agents receive a request message, respectively.

P2: *E* <> LA114.RRrequest and LA114.WReturn. There exists a path such that the state of LA114 is "RRequest" and that of LA114 is "WReturn". If this property is satisfied, it represents that LA114 has a brother solution.

P3: *E* <> LA111.RRrequest and LA111.WReturn. There exists a path such that the state of LA111 is "RRequest" and that of LA111 is "WReturn". If this property is satisfied, it means that LA111 has a brother solution.

P4: *E* <> LA121.AL ≥ LA111.RL+LA114.RL. There exists a path such that $LA121_{(AL)} \geq LA111_{(RL)} + LA114_{(RL)}$. If this property is satisfied, it represents that LA121 is an available solution for LA111 and LA114 by Equation 3.

P5: *E* <> LA111.Search and LA114.Search and MA11.Search and MA11.pIndex==2. There exists a path such that the states of LA111, LA114, and MA11 are "Search" and the value of MA11.pIndex is two. If this property is satisfied, it means that two faults occur in the ESs of LA111 and LA114, respectively, and MA11 receives two request messages from LA111 and LA114.

P6: *E* < > MA11.pIndex==2 and MA11.WReturn and LA121.HCloseAct and LA121.ActNum==2. There exists a path such that the value of MA11.pIndex is two, the state of MA11 is "WReturn", the state of LA121 is "HCloseAct", and the value of LA121.ActNum is two. If this property is satisfied, it represents that a close message is sent by MA11 after MA11 receives two request messages. Moreover, LA121 receives a close message (two actuators should be closed in the close message).

P7: *E* <> LA121.HCloseAct and LA121.ActNum==2 and LA121.AL ≥ LA111.RL+ LA114.RL. There exists a path such that the state of LA121 is "HCloseAct", the value of LA121.ActNum is two, and $LA121_{(AL)} \geq LA111_{(RL)} + LA114_{(RL)}$ is true. If this property is satisfied, it represents that LA121 receives a close message, two faults need to be restored by LA121, and LA121 can simultaneously restore the two faults.

The seven properties are verified by the verifier of UPPAAL and the results are shown in Figure 8. According to the results, we can conclude:

- **P1 is satisfied:** Two faults occur in the ESs of LA111 and LA114.
- **P2 and P3 are not satisfied:** LA111 and LA114 do not have brother solutions.
- **P4 is satisfied:** LA121 can restore the two faults.
- **P5 is satisfied:** LA111 and LA114 send a request message to MA11, respectively.
- **P6 is satisfied:** MA11 receives two request messages and it sends a close message to LA121 with the attached information (LA111 and LA114).
- **P7 is satisfied:** LA121 receives a close message with the attached information (LA111 and LA114) and the two faults can be restored by LA121.

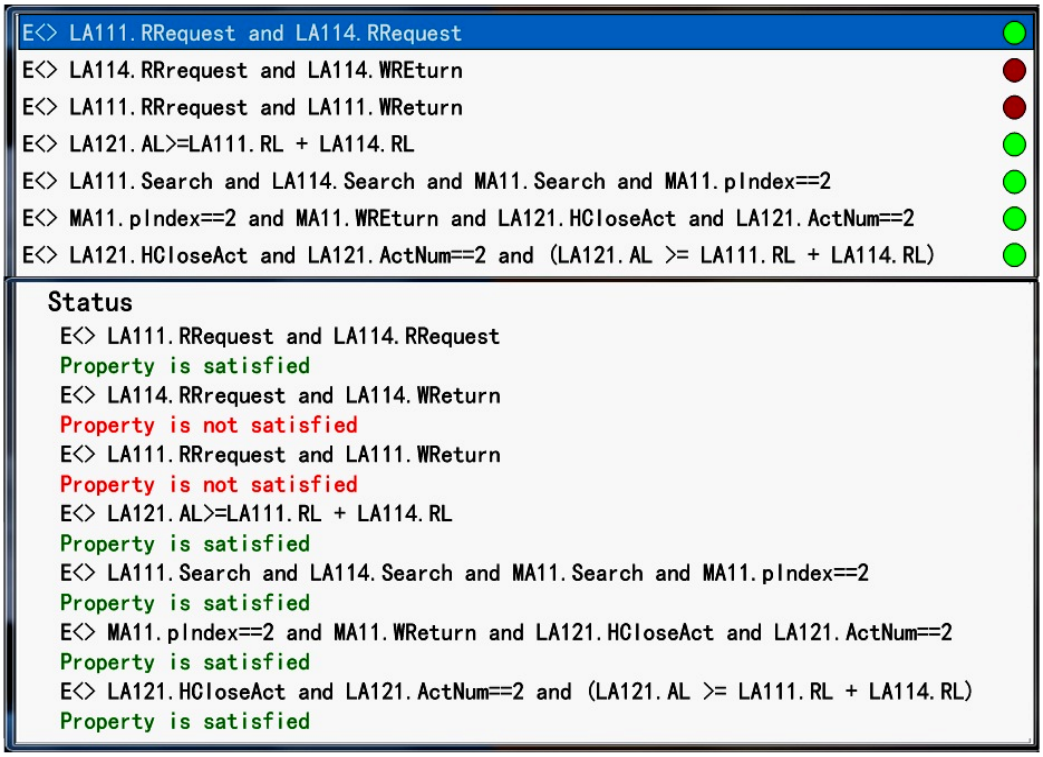

Figure 8. Properties and the results of the verification

Therefore, the two faults that occur in the ESs of LA111 and LA114 can be restored by their neighbor solution LA121.

5. Conclusion

This paper proposes a novel hierarchical multi-agent architecture for the automatic fault detection, location and restoration of smart grids based on IEC 61499/61850, which is composed of three types of agents. The input lines of high, medium, and low voltage ESs are supervised by the associated HAs, MAs, and LAs, respectively. The proposed coordination protocol can coordinate the agents to perform the automatic restoration. Moreover, the

solutions are classified into three types (brother, neighbor, and friend solutions). The properties of agents are simulated with timed automata and the correctness of coordination protocol is verified by UPPAAL through a case study. The numbers of communication lines and exchanged messages are optimized comparing with the existing coordination protocols.

In the future work, the proposed hierarchical multi-agent architecture and protocol should be implemented by PLCs (Programmable Logic Controllers). The goal is to construct a virtual small scale smart grid to simulate and verify the correctness of the proposed architecture and coordination protocol.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant No. 61074035 and 61104110, the Fundamental Research Funds for the Central Universities under Grant No. JY10000904001 and K50510040012, the National Research Foundation for the Doctoral Program of Higher Education, the Ministry of Education, P. R. China, under Grant No. 20090203110009.

References

[1] V. Behjat and A. Vahedi, "Numerical modelling of transformers interturn faults and characteris-ing the faulty transformer behaviour under various faults and operating conditions", IET Electric Power Applications, vol. 5, no. 5, **(2010)**, pp. 415-431.
[2] U. S. DOE, "Grid 2030: A national vision for electricity's second 100 years", United States Department of Energy, **(2003)**.
[3] N. B. Bhatt, "August 14, 2003, U.S.-Canada blackout", in Proceedings of the IEEE PES General Meeting, Denver, CO., **(2004)**.
[4] M. Pipattanasomporn, H. Feroze and S. Rahman, "Multi-agent systems in a distributed smart grid: Design and implementation", In Proceedings of IEEE PES 2009 Power Systems Conference and Exposition (PSCE'09), WA, USA: IEEE/PES, **(2009)**, pp. 1-8.
[5] L. Changsoo, J. Daewon and L. Keun-Wang, "Design and implementation of small scale electric power management system", International Journal of Control and Automation, vol. 6, no. 3, **(2013)**, pp. 375-382.
[6] H. M. Liu, X. Y. Chen, K. Yu and Y. H. Hou, "The control and analysis of self-healing urban power grid", IEEE Transactions on Smart Grid, vol. 3, no. 3, **(2012)**, pp. 1119-1129.
[7] V. Calderaro, C. N. Hadjicostis, A. Piccolo and P. Siano, "Failure identification in smart grids based on petri net modeling", IEEE Transactions on Industrial Electronics, vol. 58, no. 10, **(2011)**, pp. 4613-4623.
[8] L. F. Wang, Z. Wang and R. Yang, "Intelligent multiagent control system for energy and comfort management in smart and sustainable buildings", IEEE Transactions on Smart Grid, vol. 3, no. 2, **(2012)**, pp. 605-617.
[9] T. Logenthiran, D. Srinivasan, A. M. Khambadkone and H. N. Aung, "Multiagent system for real-time operation of a microgrid in real-time digital simulator", IEEE Transactions on Smart Grid, vol. 3, no. 2, **(2012)**, pp. 925-933.
[10] M. E. Baran and I. M. El-Markabi, "A multiagent-based dispatching scheme for distributed generators for voltage support on distribution feeders", IEEE Transactions on Power Systems, vol. 22, no. 1, **(2007)**, pp. 52-59.
[11] V. M. Catterson, E. M. Davidson and S. D. J. McArthur, "Embedded intelligence for electrical network operation and control", IEEE Intelligent Systems, vol. 26, no. 2, **(2011)**, pp. 38-45.
[12] J. A. Momoh, "Navy ship power system restoration using multi-agent approach", In Proceedings IEEE Power Engineering Society General Meeting, Montreal, Quebec, Canada, **(2006)**.
[13] Y. L. Xu and W. X. Liu, "Novel multiagent based load restoration algorithm for microgrids", IEEE Transactions on Smart Grid, vol. 2, no. 1, **(2011)**, pp. 152-161.
[14] H. Liang, B. J. Choi, W. H. Zhuang, X. M. Shen, A. S. A. Awad and A. Abdr, "Multiagent coordination in microgrids via wireless networks", IEEE Wireless Communications, vol. 19, no. 3, **(2012)**, pp. 14-22.
[15] V. Vyatkin, G. Zhabelova, N. Higgins, K. Schwarz and N. K. C. Nair, "Towards intelligent smart grid devices with iec 61850 interoperability and iec 61499 open control architecture", in Proceedings of IEEE Conference on Transmission and Distribution, New Orleans, USA, **(2010).**
[16] G. Zhabelova and V. Vyatkin, "Multiagent smart grid automation architecture based on iec 61850/61499 intelligent logical nodes", IEEE Transactions on Industry Electronics, vol. 59, no. 5, **(2012)**, pp. 2351-2362.

[17] F. Z. Peng, "A generalized multilevel inverter topology with self voltage balancing", IEEE Transactions on Industry Applications, vol. 37, no. 2, **(2001)**, pp. 611-618.
[18] R. Alur and D. L. Dill, "A theory of timed automata", Theoretical Computer Science, vol. 126, no 2, **(1994)**, pp. 183-235.
[19] G. Behrmann, A. David and K. Larsen, "A tutorial on uppaal", Springer Formal Methods for the Design of Real-time Systems, Lecture Notes in Computer Science, Berlin: Springer, (**2004**).
[20] H. Berrir and I. Slama-Belkhodja, "Enhanced parity equations method for sensor fault detection in electrical drives", in Conference on Control and Fault-Tolerant Systems, Nice, France, **(2010)**.
[21] I. Slama-Belkhodja, J. Arbi, S. Skander-Mustapha, M. J.-B Ghorbal, M. Bejaoui and I. Bahri, "Reconfiguration strategies for doubly fed induction generators based wind system", in Proceedings of 6th International Multi-Conference on Systems, Signals and Devices (SSD'09), Tunisia, Tunisian, **(2009)**, pp. 1-16.
[22] J. E. Hopcroft, R. Motwani and J. D. Ullman, "Introduction to automata theory, languages, and computation", Addison-wesley, Cambridge, MA., **(1979)**.

Authors

Zhongyuan Jiang received the B.S. and the M.S. degrees in Computer Science and Technology from Xihua University, Chengdu, China, in 2004 and 2008, respectively. He is currently a Ph.D. student in School of Electro-Mechanical Engineering, Xidian University, Xi'an, China, from 2011. His research interests include supervisor control theory and fault detection and restoration of smart grids and discrete event systems.

Mohamed Khalgui received the B.S. degree in computer science from Tunis University, Tunisia, in 2001, the M.S. degree in telecommunication and services from Henri PoincaréUniversity, France, in 2003. He did research in computer science at INRIA Institute to obtain the Ph.D. degree from the French Polytechnic Institute of Lorraine, France, in 2007. Thanks to a Humboldt grant, he did research activities in computer science at Martin Luther University, Germany, to get the Habilitation Diploma in IT, in 2012. Mr. Khalgui is currently a senior researcher at ITIA-CNR Institute, Italy. He is active in several European projects and in other interesting international collaborations.

Olfa Mosbahi received the B.S. degree in computer Science and the M.S. degree from Tunis University in 1999 and 2002, respectively. She did research in Computer Science at INRIA Institute to receive the Ph.D. degree at the French Polytechnic Institute of Lorraine, France, in 2008. She is an assistant professor in Computer Science at Carthage University, Tunisia. She was a part time researcher at INRIA, France and a temporary lecturer at Nancy II University. She was also a researcher at Martin Luther University, Germany. She is active in several European Projects and also in other interesting international collaborations.

Aymen Jaouadi received the B.S. and the M.S degrees in Computer Science and Information System from Tunis El Manar University, Tunis, Tunisia, in 2010 and 2012, respectively. He is currently a Ph.D. student in Tunis El Manar University in collaboration with the laboratory LISI-INSAT since 2013. His research interests include Concurrent Fault Management in the electric grids and new solutions for Home Automation Systems and Renewable Energy in the Green Low Power Systems.