Reconfigurable Wireless Sensor Networks Simulator (RWSNSim): A New Discrete-event Simulator

Hanene Rouainia[1][a], Hanen Grichi[2,3][b], Laid Kahloul[4][c] and Mohamed Khalgui[3,5,*][d]

[1]*Faculty of Sciences of Tunis, El-Manar University, Tunis, Tunisia*

[2]*Faculty of Sciences of Bizerte (FSB), University of Carthage, Bizerte, Tunisia*

[3]*School of Electrical and Information Engineering, Jinan University, Zhuhai, China*

[4]*LINFI Laboratory, Computer Science Department, Biskra University, Biskra, Algeria*

[5]*INSAT Institute, University of Carthage, Tunis, Tunisia*

hanene.rouainia@fst.utm.tn, {hanen.grichi, laid.k.b, khalgui.mohamed}@gmail.com

Keywords: Wireless Sensor Network, Reconfigurable Wireless Sensor Network, RWSNSim, Simulator, Sensor Node, Sink Node, Agent.

Abstract: Reconfigurable wireless sensor networks become an important area in research and industrial communities. With their development and spread, many problems and solutions have emerged. Network simulators have become an essential necessity to study the impact of these solutions on networks in order to avoid huge costs in terms of money, time, and effort if applied on the ground. In this paper, we propose a new discrete-event simulator for WSNs and RWSNs called *RWSNSim*. We present its description, modeling, and provided services. Finally, to demonstrate the efficiency of *RWSNSim*, we simulate a case study and detail the simulator functioning steps.

1 INTRODUCTION

With the spread of microelectromechanical systems (MEMS) technology, wireless sensor networks (WSNs) have gained worldwide attention. WSNs deploy a set of multi-functional devices known as sensor nodes (SNs). They have considerable characteristics such as small size, low cost and computing resources, and limited processing. Sensor nodes can sense both physical and chemical measurements in the surrounding environment to process the sensing data, communicate with each other wirelessly, and work cooperatively (Agrawal, 2017), (Rouainia et al., 2022). WSNs can be used in a variety of areas like medical, environmental monitoring, military, and smart homes (Khriji et al., 2018), (Vijayalakshmi and Muruganand, 2018).

We have several challenges in WSNs like lack of energy problem which occurs because the WSNs work under many types of renewable energy resources which are not frequently available, real-time problem which means that in some applications, the transmitted messages over the network must respect their deadlines, and packet dropping which may be caused by software or hardware failures as mentionned by several researchers in literature (Rouainia et al., 2020), (Rouainia et al., 2022), (Hafidi et al., 2020).

Reconfigurable wireless sensor networks (RWSNs) are wireless sensor networks with the possibility to execute reconfiguration scenarios such as mobility and resizing. The existence of additional specific devices (i.e., mobile sensor nodes, mobile sink nodes, and software and hardware agents) allows RWSNs to execute reconfiguration scenarios. These reconfiguration scenarios are proposed as solutions to the mentioned WSNs problems in our previous works (Rouainia et al., 2020), (Rouainia et al., 2022).

Since the expensive cost, effort, time, and complexity implicated in the construction and the implementation of RWSNs, the developers prefer to get an overview about feasibility and behaviour of RWSNs before hardware implementation. Indeed, the analysis and evaluation of the proposed solutions and techniques through real experiments are not feasible, complex, and very expensive in terms of time, effort, and cost. As result, to keep up with these challenges,

[a] https://orcid.org/0000-0001-7544-988X
[b] https://orcid.org/0000-0002-4601-3574
[c] https://orcid.org/0000-0002-9739-7715
[d] https://orcid.org/0000-0001-6311-3588
[*]Member IEEE

several simulation tools (Nayyar and Singh, 2015) are proposed to test and analyse the performance of the proposed techniques, protocols, and solutions. They have many advantages like low cost, easy development, giving real-time results, and detecting the positive and negative effects on the entire network.

There are two types of simulations surrounding WSNs and RWSNs: trace-driven simulation and discrete-event simulation. Trace-driven simulation is an important approach in many simulation applications, especially in real-time applications. It enables fast design evaluation by considering system models which are derived from a sequence of observations made on a real system. It allows users to get in-depth details of the simulation model. But it contains several drawbacks like increasing the complexity of the simulation. On the other hand, discrete-event simulation is used to model real-world systems that can be divided into several logically separate processes that autonomously progress through time. This type of simulation is mostly used in WSNs and RWSNs due to its ease in simulating various tasks running on different sensor nodes, sinks, and agents (Nayyar and Singh, 2015), (Rouainia et al., 2022).

We can evaluate and compare the simulators of WSNs and RWSNs using a set of parameters including the following: the type of simulator which is classified into three categories: generic, code level, and firmware simulator, the license which can be commercial or open-source, the platform which is the operating system on which the simulator operates such as Windows, Linux, or both, and WSN platforms which are defined in terms of sensors types and platforms which can be simulated by the simulator (Nayyar and Singh, 2015).

In this paper, we propose a new simulator *RWSNSim* to construct WSNs and RWSNs, save them in a database, use two routing protocol (LEACH and WBM-TEEN), plot the simulation graph, present an execution report for each monitoring time, draw the resulting line charts after the simulation, and compare between the different networks and simulations (RWSNSim, 2022). It permits also to apply the methodology proposed in (Rouainia et al., 2022) which is a new energy efficient and fault tolerant methodology based on a multi-agent architecture in RWSNs using mobile sink nodes, mobility, resizing, and test packet technique. The proposed simulator is considered as a discrete-event simulator.

The rest of the paper is organized as follows. Section 2 presents the related works. The new simulator is developed in Section 3. Section 4 exposes a case study executed by *RWSNSim*. Finally, the conclusion is drawn in Section 5.

2 RELATED WORKS

Several WSNs simulation tools have been proposed by academic and commercial communities (Nayyar and Singh, 2015), (Rajan et al., 2015). In this section, we will discuss some of the most important work in this field.

RWiN-Environment (RWiN, 2016) is a graphical tool developed to evaluate the services of the RWiN-Methodology which is proposed to analyse, construct, develop, and verify an RWSN system in order to reduce the consumed energy by the network (Grichi et al., 2016a), (Grichi et al., 2016b).

Network Simulator-3 (NS-3) (NS3, 2022) is a discrete-event simulator for Internet systems. It was launched in June 2008 as an open-source project. It is free, targeted primarily at research and educational uses, licensed under the GNU GPLv2 license, and maintained by a worldwide community. Network Simulator-3 is written in C++ language and python. The simulations executed in NS-3 can be implemented using pure C++ with optional python bindings. It can work in various operating system platforms like Linux and Windows via Cygwin. The latest version of NS-3 is NS-3.35 which is released in October 1, 2021. It provides several improvements compared to previous versions like IPv6 support for NixVectorRouting and a group mobility helper.

OMNet++ (OMNeTPP, 2019) is a powerful object-oriented discrete-event simulator. It can be used for the simulation of computer networks, distributed and parallel systems, modeling of multiprocessors, and performance evaluation of complex software systems. It was launched in September 1997 and has a large number of users in academic, educational, and research-oriented commercial institutions. Indeed, OMNeT++ is not a simulator, but it permits writing simulation scenarios using several frameworks and tools. It is considered as an extensible, modular, and component-based open-architecture simulation framework implemented using C++ programming language. It provides an extensive graphical user interface (GUI) and intelligence support. OMNET++ distributions are available for different operating systems like Windows, Linux, and MAC OS X. The latest version of OMNeT++ is 5.7 which is released in October 6, 2021 and is intended to be the last release of the 5.x series.

JavaSim (J-Sim) simulator (JSim, 2022) is an object-oriented simulation package based upon C++SIM started in year 1997. It is considered as a general purpose simulator used by many commercial and academic organizations. J-Sim is a platform open source, free, extensible, neutral, and reusable because

Table 1: Comparison of some simulation tools.

Name	Type	Program. Language	License	GUI	Scalability	Portability	Designed for (WSNs or RWSNs)
RWiN-Environment	Discrete event	Java	Open source	Excellent	Medium	Yes	RWSNs
NS-3	Discrete event	C++, Python	Open source	Limited	Large	Yes	WSNs
OMNet++	Discrete event	C++	Open source	Excellent	Large	Yes	WSNs
JavaSim	Discrete event	Java	Open source	Medium	Medium	Yes	WSNs
GloMoSiM	Discrete event	C	Open source	Limited	Large	Yes	WSNs
RWSNSim	Discrete event	Java	Open source	Excellent	Medium	Yes	WSNs, RWSNs

it has been developed using Java, based on ACA (Autonomous Component Architecture), and offers discrete-event process-based simulation. The use of a script interface in J-Sim permits executing several script-based languages such as Python and Perl. The current version of JavaSim is 2.2.0 GA which is released in January 4, 2020. J-Sim offers a framework for WSN simulation using INET, ACA, and Wireless Protocol stack.

Global Mobile Information System Simulator (GloMoSiM) (GloMoSim, 2020) is a discrete-event scalable simulation software used for large-scale wireless and wired network systems and developed by Parallel Computing Lab at UCLA. It is designed solely for wireless networks using parallel discrete-event simulation provided by Parsec. It compiles the simulation of protocols using Parsec compiler. GloMoSiM supports three communication protocols: traditional Internet protocols, multi-hop wireless communication, and direct satellite communication. It permits to simulate networks with thousands of heterogeneous nodes and communication links.

Finally, the proposed simulator *RWSNSim* (RWSNSim, 2022) is considered as the first simulation tool designed for both WSNs and RWSNs. Table 1 presents a comparison of the mentioned simulation tools according to general parameters.

Through table 1, we conclude that the strengths of *RWSNSim* are summarized in the following: license, graphical user interface (GUI), portability and its ability to simulate both WSNs and RWSNs.

3 RWSNSim: RECONFIGURABLE WIRELESS SENSOR NETWORKS SIMULATOR

In this section, we describe the *RWSNSim* simulator and present its provided services. We also model the architecture of WSNs and RWSNs using class diagrams and validate them. Furthermore, we model the used databases to store the information of WSNs and RWSNs using entity relationship diagrams.

3.1 Description

RWSNSim is a descrete-event network simulator written in Java. It is developed using *Java* for wireless sensor networks and reconfigurable wireless sensor networks. It is an open source, free, reusable and extensible simulator. It is designed for research and educational use. *RWSNSim* is capable of simulating networks that contain hundreds of sensor nodes, sink nodes, and zone agents. It supports several operating systems such as Windows and UNIX.

The proposed simulator is a desktop simulator that makes it possible to construct WSNs and RWSNs and simulate them using two routing protocols (LEACH and WBM-TEEN). It can also apply the proposed methodology in (Rouainia et al., 2022) which is a new energy efficient and fault tolerant methodology in RWSNs. This methodology based on a multi-agent architecture and consists of the implementation of a set of techniques such as the use of mobile sink nodes, mobility, resizing of zones, and test packet technique.

RWSNSim provides several services during the simulation process, including the following:

- Creating WSNs and RWSNs and saving them in a database using *Java*, *MySQL* or *hsqldb* according to the user choice.
- Providing two routing protocols: LEACH and WBM-TEEN which are considered as energy efficient protocols.
- Executing the simulation graph of each network using *jgraph*, *jgraphx*, and *jgrapht* libraries.
- Displaying the execution report for each monitoring time.
- Extracting the simulation results and representing them in form of line charts using *jfreechart* library.
- Comparing the different networks and simulations.

3.2 WSNs and RWSNs Modeling

In the following, we describe the target sensor networks used by *RWSNSim* simulator. Then, we present the architecture of WSNs and RWSNs with their components and relations between them using class diagrams which are used to develop the *RWSNSim* simulator. We also validate the used class diagrams by a natural language and with ROCL constraints. This last one is an extension of OCL which improves the specification and validation of constraints related to different execution scenarios of systems (Grichi et al., 2015).

3.2.1 The Target Sensor Networks

(a) *Wireless Sensor Networks (WSNs):*

With *RWSNSim*, each wireless sensor network composed of:

- A base station *BS* plays the role of a gateway between the WSN and its administrator. It receives the sensing data from the gateways and analyses them, which allows the administrator of the network to take an overview of the network and to call human intervention in some cases.
- A set of zones.
- Each zone contains a gateway that collects the sensing data from the cluster heads and sends them to the base station.
- Each zone contains a set of clusters.
- Each cluster contains a cluster head that collects the sensing data from its cluster sensor nodes and sends them to its zone gateway.
- Each cluster contains a set of sensor nodes that monitor the chemical and physical measurements in the surrounding environment and send them to their cluster head.

(b) *Reconfigurable Wireless Sensor Networks (RWSNs):*

With *RWSNSim*, each reconfigurable wireless sensor network composed of:

- A base station *BS* plays the role of a gateway between the RWSN and its administrator. It receives alert messages from the zone agents, analyses them and communicates with the administrator of the network to call human intervention in some cases.
- A controller agent Ag_{Ctrl} controls the whole network by executing several tasks such as receiving the sensing data from the zone agents, analysing them, sending alert messages to the *BS*, and applying the resizing of zones.
- A set of zones.
- Each zone contains a zone agent which controls the zone by executing several tasks such as organizing the sensor nodes in subzones into clusters, applying the mobility, and detecting the malfunctioning sink nodes.
- Each zone contains a set of subzones.
- Each subzone contains a mobile sink node that executes several tasks like collecting the sensing data from the cluster heads in the subzone, sending them to the zone agent, and detecting the malfunctioning sensor nodes. Each mobile sink node is controlled by a software agent installed in the sink itself named sink agent.
- Each subzone contains a set of clusters.
- Each cluster contains a cluster head that collects the sensing data from the sensor nodes in the cluster and sends them to the sink node.
- Each cluster contains a set of sensor nodes that monitor the chemical and physical measurements in the surrounding environment and send them to their cluster head. Each sensor node is controlled by a software agent installed in the sensor node itself named node agent.

3.2.2 Class Diagrams

Figure 1 presents the class diagram of wireless sensor networks which is used to develop *RWSNSim.* In this class diagram, we have 11 classes of which we define the following:

- *WSN:* It models a WSN and contains the different thresholds, constants, and consumption values. They can be used by the different entities

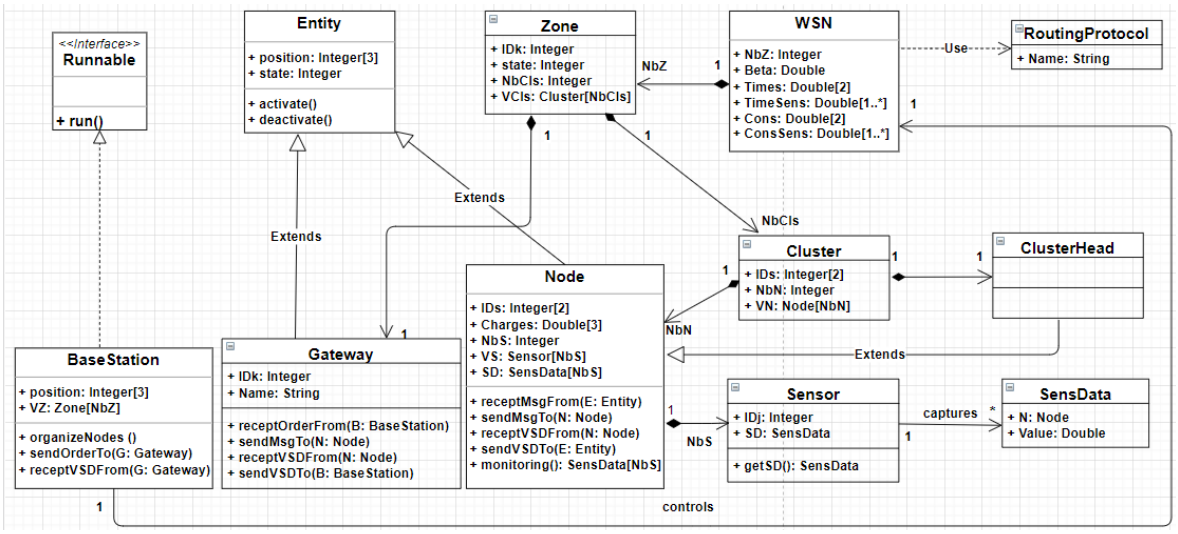


Figure 1: WSN class diagram.

in WSNs such as an energy threshold [*Beta*], and vectors of consumed values of energy and time parameters [*Times*, *TimeSens*, *Cons*, *ConsSens*].

- *BaseStation:* it models the base station *BS*. It contains the *BS* principal properties such as its position [*position*], vector of zones [*VZ*] and operations like organizing the nodes in each zone Z_k [*organizeNodes()*], sending an order to a gateway G_k [*sendOrderTo(G: Gateway)*].
- *Zone:* it models each zone Z_k and contains its principal properties such as an identifiant [*IDk*], its state [*state*], and a vector of clusters [*VCls*].
- *Gateway:* it models each gateway G_k and contains its principal properties like an identifiant [*IDk*] and a name [*Name*] and operations such as receiving an order from *BS* to collect the sensing data [*receptOrderFrom(B: BaseStation)*] and sending a vector of sensing data to *BS* [*sendVSDTo(B: BaseStation)*].
- *Cluster:* it models each cluster in Z_k. It contains the clusters' principal properties like identifiants [*IDs*] and a vector of sensor nodes belonging to this cluster [*VN*].
- *Node:* it models each sensor node $N_{i,k}$ in *WSN*. It extends from the *Entity* class and contains the sensor nodes' principal properties such as identifiants [*IDs*], a vector of batteries charges [*Charges*], and a vector of sensors [*VS*] and operations like monitoring physical or chemical measurements in the environment [*monitoring(): SensData[NbS]*] and sending a vector of sensing data to the successor [*sendVSDTo(VSD: SensData, E: Entity)*].

Figure 2 presents the class diagram of reconfigurable wireless sensor networks which is used to develop *RWSNSim*. In this class diagram, we have 17 classes of which we define the following:

- *RWSN:* it models a reconfigurable wireless sensor network. It contains the different constants, variables, energy and time consumption values. They can be used by the different entities in RWSNs such as (number of zones [*NbZ*] and vectors of thresholds and consumed values of energy and time parameters [*Thresholds*, *SensThr*, *Cons*, *ConsSens*]).
- *Entity:* is the class model of all physical entities in RWSNs. It contains their common properties like the state of the entity [*state*] and its position [*position*] and operations such as activating the entity [*activate()*] and deactivating it [*deactivate()*].
- *AgCtrl:* it models the controller agent Ag_{Ctrl}. It extends from the *Entity* class and contains the Ag_{Ctrl} principal properties like a vector of zones [*VZ*] and a vector of sensing data [*VSD*] and operations such as applying the resizing task [*resizing(Za: Zone, Zb: Zone)*], sending an order to a zone agent [*sendOrderTo(Ag: ZoneAgent)*], sending a test packet to zone agents to detect the malfunctioning zone agent [*sendTstPckTo(Ag: ZoneAgent)*], and isolate a malfunctioning zone agent [*isolate(Ag: ZoneAgent)*].
- *ZoneAgent:* it models each zone agent Ag_k in *RWSN*. It extends from the *Entity* class and contains the zone agents' principal properties like an identifiant [*IDk*], a vector of sensor nodes [*VN*], and a vector of subzones [*VSZ*] and operations such as organizing the sensor nodes in Z_k [*orga-

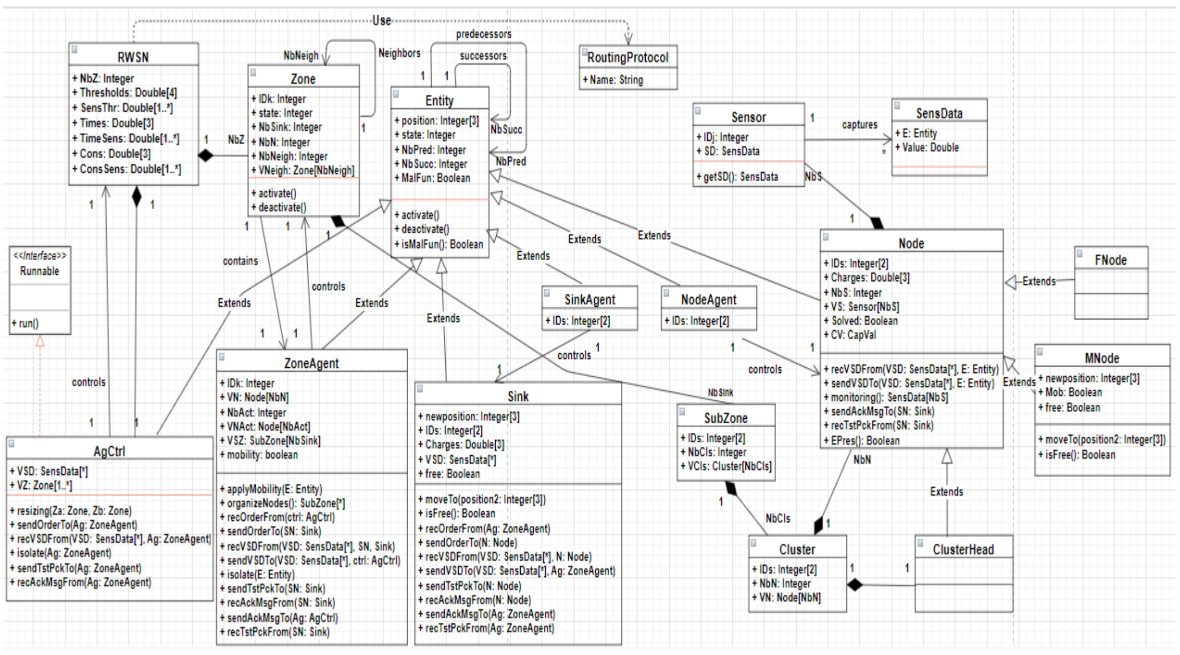


Figure 2: RWSN class diagram.

nizeNodes(): SubZone[*]], applying the mobility task [*applyMobility(E: Entity)*], and isolate the malfunctioning sinks and sensor nodes [*isolate(E: Entity)*].

- *Sink:* it models each mobile sink node $SN_{m,k}$ in *RWSN*. It extends from the *Entity* class and contains the mobile sink nodes' properties like identifiants [*IDs*], new position coordinations [*newposition*], and a boolean property [*free*] and operations such as moving to another position [*moveTo(position2: Integer[3])*], sending an acknowledge message to the zone agent Ag_k [*sendAckMsgTo(Ag: ZoneAgent)*] and sending a test packet to a sensor node $N_{i,k}$ [*sendTstPckTo(N: Node)*].
- *Sensor:* it models each sensor $Sens_{j,N_{i,k}}$ in a sensor node $N_{i,k}$. It extends from the *Entity* class and contains the sensors' principal properties such as a sensing value [*SD*].

3.2.3 Validation Model

In order to validate the class diagrams which are presented in figure 1 and 2, we use natural language and ROCL. The natural language permits to describe the different constraints that must be respected by the different entities in each network. While, ROCL allows formalizing them. We present in the following some constraints that must be respected by some entities in an RWSN.

A. Natural Language - Constraint 01

- **AgCtrl Class:** in this class, we have two important constraints which are explained as follows:

  - Before the application of the resizing task between two zones Z_a and Z_b, the following conditions must be met:

  a- The number of active nodes in Z_a must be less than or equal to $\Lambda = Thresholds[3]$.

  b- Z_b must be a neighbor zone of Z_a which contains the minimum number of active nodes.

  - After the application of the resizing task between Z_a and Z_b the following results must be met:

  a- The number of zones will be decreased.

  b- The number of nodes in Z_b will be increased.

  c- The zone agent Ag_a and Z_a will be deactivated.

  d- The number of sinks in Z_b will be increased.

  e- The number of Z_b neighbors will be increased.

B. Natural Language - Constraint 02

- **ZoneAgent Class:** in this class, we have two important constraints which are defined as follows:

  - If the total charge of a node $N_{i,k} \in S_N$ is less than or equal to $\alpha = Thresholds[1]$, the zone agent Ag_k must apply the mobility task.

  - After the application of the mobility, the deactivation or the isolation of sensor nodes or sink nodes in a zone Z_k, Ag_k must organize the order of nodes in Z_k in subzones into clusters.

ROCL - Constraint 01
context $AgCtrl,\ RWSN,\ ZoneAgent\ Aga$ **INV**
$resizing(Zone\,Za,\ Zone\,Zb) :: Execute()$
IMPLIES$deacivate(Aga)$**AND**$deactivate(Za)$
pre : $Za.NbNAct \leq Thresholds[3]$ **AND**
$Za.VNeigh \rightarrow$ **includes**(Zb) **AND** $Zb.NbNAct \leq$
$Za.VNeigh.allInstances \rightarrow NbNAct$
post : $NbZ = NbZ - 1$ **AND**
$Zb.NbN = Zb.NbN + Za.NbN$
$deacivate(Za) :: Execute()$ **IMPLIES**
$Zb.NbSink = Zb.NbSink + Za.NbSink$**AND**
$Zb.NbNeigh = (Zb.NbNeigh - 1) + Za.NbNeigh$
AND $Zb.VNeigh = Zb.VNeigh -$
$Za \rightarrow union(Za.VNeigh)$

ROCL - Constraint 02
context $ZoneAgent\ Ag,\ Entity\ E,$
$NodeN, SinkSN, MNodeMN$**INV**
IF $(N.charges[2] \leq \alpha)$ **IMPLIES**
$(applyMobility(MN)$ **OR** $applyMobility(SN))$
context $deactivate(Entity\,E) :: Execute()$
$(Node, Sink) \rightarrow$ **includes**$(self)$ **IMPLIES**
$organizeNodes()$
context $Node\,N,\ ZoneAgent\,Ag,\ Entity\,E,$
$Ag.applyMobility(Entity\,E) :: Execute()$
$(MNode, Sink) \rightarrow$ **includes**$(self)$ **IMPLIES**
$organizeNodes()$

C. Natural Language - Constraint 03

- **Sink Class and MNode Class:** in these classes, we have an important constraint which is defined as follows:

  - When a zone agent Ag_k applies the mobility of a mobile entity E where E is a mobile sink node or a mobile node. The following changes will take place:

   a- The position of E must be changed.

   b- The charge of the principal battery of E must be decreased.

ROCL - Constraint 03
context $Sink\ E,\ MNodeE, E.moveTo(position2 :$
$Integer[3]) :: Execute()$
post :$E.Charges[0] = E.charges[0] -$
$Emob(newposition, position2)$
AND $E.newposition = position2$

3.3 Databases Modeling

To create the databases used to store the information related to the simulated WSNs and RWSNs, we model them using entity relationship diagrams (ERD). Indeed, we have two ERDs:

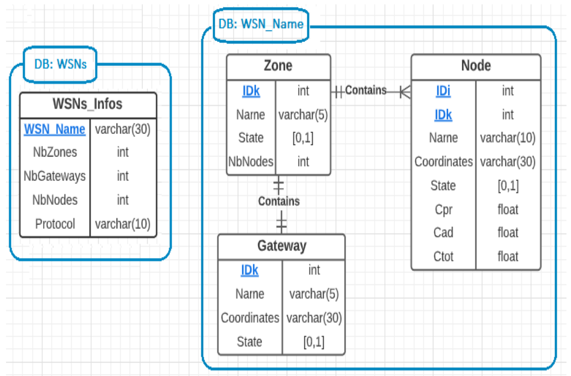


Figure 3: Entity relationship diagram for WSNs used by *RWSNSim*.

1) ERD for WSNs: Figure 3 presents the entity relationship diagram of wireless sensor networks used in *RWSNSim* to generate the databases used to store the wireless sensor networks' information entered by the user or generated automatically. It contains the following databases:

- **[DB: WSNs]:** is a database that includes a single entity named **WSN_Infos**. It is a weak entity which contains the general information of each wireless sensor network such as the WSN name [*WSN_Name*] which is the key attribute, the number of zones [*NbZones*], and the used protocol [*protocol*].
- **[DB: WSN_Name]:** For each WSN created by the user, a database with the same name is created to store its information. It includes three entities defined as follows:
  - **Zone:** is a strong entity which contains the general information of each zone like an identifiant [*IDk*] which is the key attribute, its state [*State*], and the number of sensor nodes it contains [*NbNodes*].
  - **Gateway:** is a strong entity which contains the general information of each gateway in the network such as its identifiant [*IDk*] which is the key attribute, its coordinates which presents its position in the network environment [*Coordinates*], and its state [*State*].
  - **Node:** is a strong entity which contains the general information of each sensor node in the network such as two identifiants [*IDi*, *IDk*] which are the key attributes, the initial charge of the principal battery [*Cpr*], and the initial charge of the additional battery [*Cad*].

2) ERD for RWSNs: Figure 4 presents the entity relationship diagram of reconfigurable wireless sensor networks used in *RWSNSim* to generate the databases used to store the reconfigurable wireless

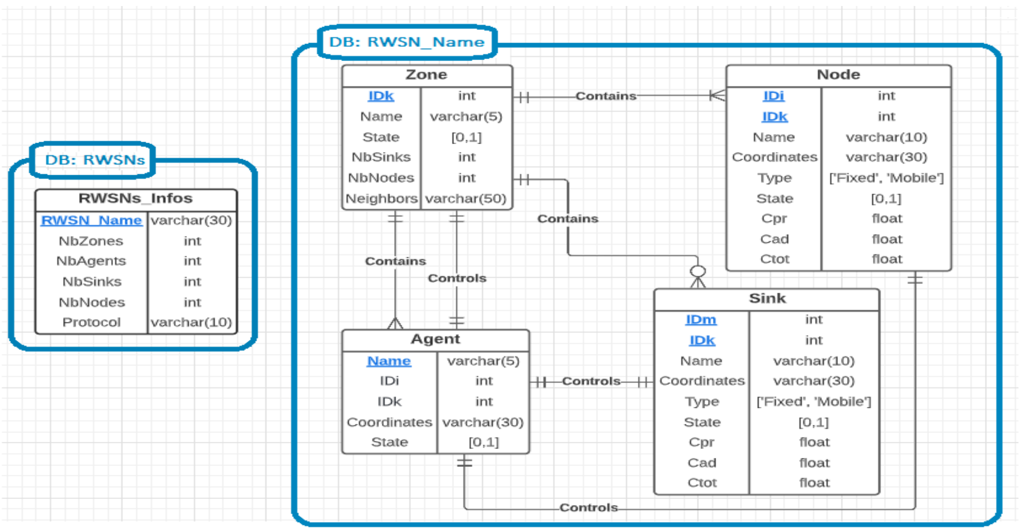


Figure 4: Entity relationship diagram for RWSNs used by *RWSNSim*.

sensor networks information entered by the user or generated automatically. It contains the following databases:

- **[DB: RWSNs]:** is a database that includes a single entity named **RWSN_Infos**. It is a weak entity which contains the general information of each reconfigurable wireless sensor network such as the RWSN name [*RWSN_Name*] which is the key attribute, the number of agents whatever their types [*NbAgents*], and the number of sink nodes [*NbSinks*].
- **[DB: RWSN_Name]:** For each RWSN created by the user, a database with the same name is created to store its information. It includes four entities defined as follows:
  - **Zone:** is a strong entity which contains the general information about each zone like an identifiant [*IDk*] which is the key attribute, its name [*Name*], and the list of neighbor zones [*Neighbors*].
  - **Agent:** is a strong entity which includes the general information about each zone agent in the network such as its name [*Name*] which is the key attribute, its coordinates which present its position in the network environment [*Coordinates*], and its state [*State*].
  - **Sink:** is a strong entity which contains the general information of each sink node in the network like two identifiants [*IDm*, *IDk*] which are the key attributes, its type which can be 'fixed' or 'mobile' [*Type*], and the total charge of its batteries [*Ctot*].
  - **Node:** is a strong entity which includes the general information of each sensor node in the network such as two identifiants [*IDi*, *IDk*] which are the key attributes, its coordinates which present its position in the network environment [*Coordinates*], and its type which can be 'fixed' or 'mobile' [*Type*].

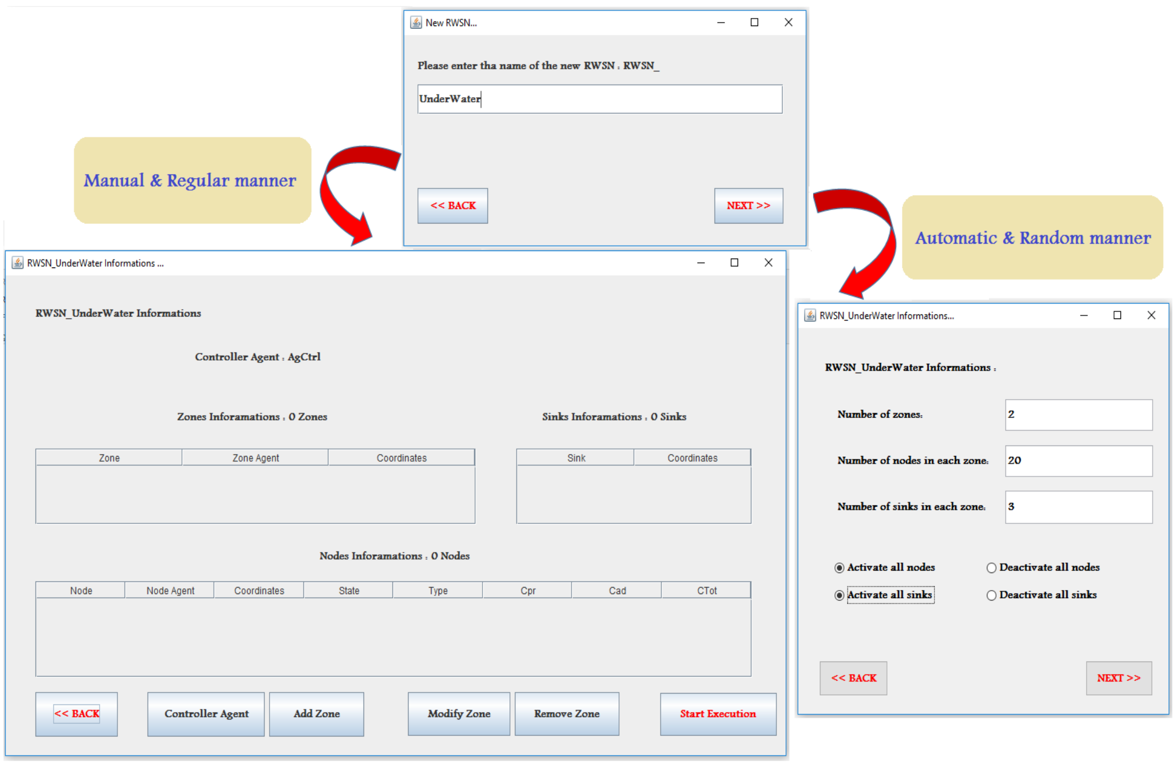
4 CASE STUDY

In this case study, we simulate a reconfigurable wireless sensor network to show all provided services by *RWSNSim* which cannot be shown if we simulate a wireless sensor network such as mobility and resizing notification windows.

We propose a reconfigurable wireless sensor network denoted by *RWSN_UnderWater* as a simulation example with *RWSNSim*. *RWSN_UnderWater* is composed of a base station termed by *BS*, a controller agent denoted by Ag_{Ctrl}, and two zones $\{Z_1$ and $Z_2\}$. Each zone $Z_k | k \in \{1,2\}$ contains one zone agent Ag_k, 20 sensor nodes (12 fixed nodes and 8 mobile nodes) denoted by $S_N = \{N_{i,k} | i \in [1..20] \quad k \in 1,2\}$, and 3 mobile sink nodes termed by $S_{SN} = \{SN_{m,k} | m \in [1..3]\}$. Each sensor node has two sensors: CO_2 sensor and CH_4 sensor.

A. Choosing the Appropriate Routing Protocol

In *RWSNSim*, we have two routing protocols: LEACH and WBM-TEEN. Before the construction of a WSN or an RWSN network, the user has to choose one of them. In this case study, we chose LEACH protocol to simulate *RWSN_UnderWater* network.



Please enter tha name of the new RWSN : RWSN_

UnderWater

RWSN_UnderWater Informations

Controller Agent : AgCtrl

Zones Inforamations : 0 Zones

Zone	Zone Agent	Coordinates

Sinks Inforamations : 0 Sinks

Sink	Coordinates

Nodes Inforamations : 0 Nodes

Node	Node Agent	Coordinates	State	Type	Cpr	Cad	CTot

RWSN_UnderWater Informations :

Number of zones:	2
Number of nodes in each zone:	20
Number of sinks in each zone:	3

[✓ Activate all nodes] Deactivate all nodes

[✓ Activate all sinks] Deactivate all sinks

Figure 5: The two manners to construct *RWSN_UnderWater* using *RWSNSim*.

B. Construction of RWSN_UnderWater using RWSNSim

With *RWSNSim*, the user can construct a WSN or an RWSN and save it in a database. We have two manners of network construction:

1) Manual and regular: depending on this manner, all information related to each entity in the network must be entered manually such as the coordinates and the state of each entity.

2) Automatic and random: depending on this manner, all information related to each entity in the network is randomly selected by the system according to some inputs such as the total number of zones and the number of sensor nodes and sink nodes in each zone. All information can be modified by the user in the created database.

Figure 5 shows these two manners in the *RWSNSim* graphic user interface.

In the proposed case study, we chose the manual and regular manner. Therefore, we have added two zones and the entities they contain (zone agents, mobile sink nodes, and sensor nodes) with their information.

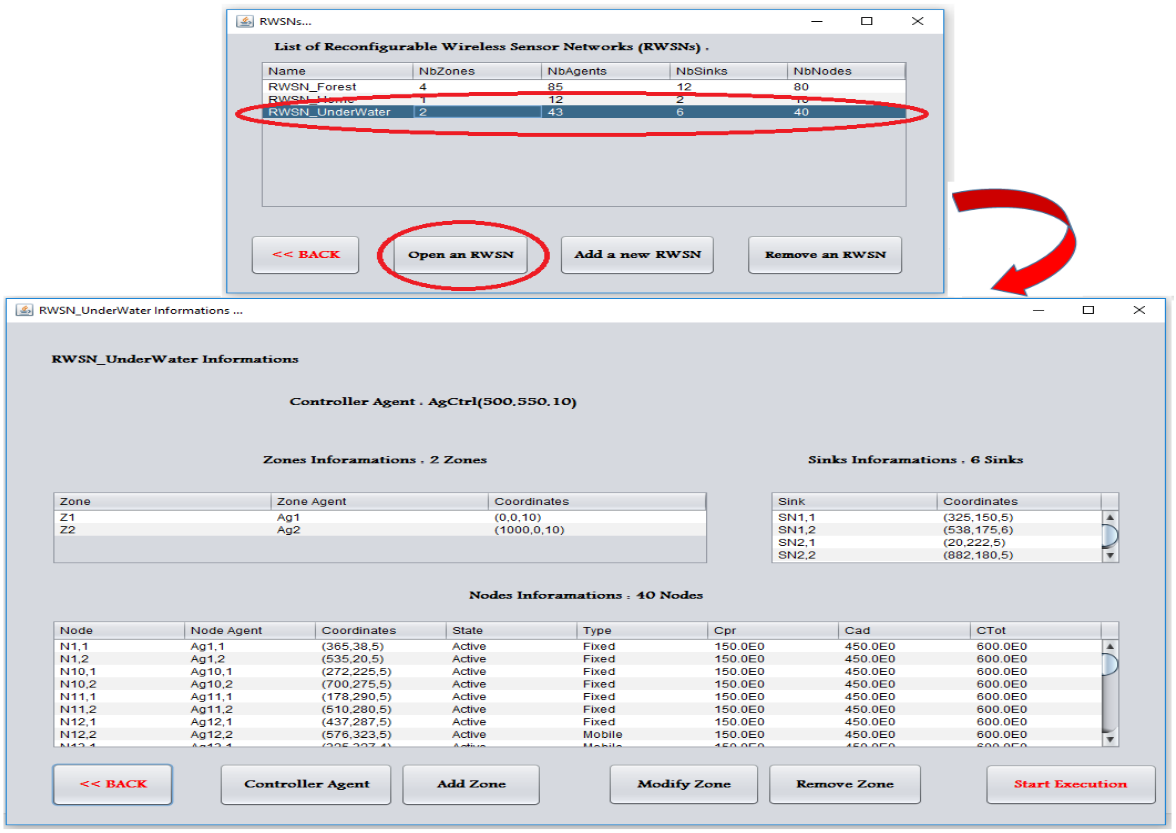
After the construction of *RWSN_UnderWater* network, *RWSNSim* saves all information entered by the user or generated automatically in a database. Figure 6 shows the *RWSN_UnderWater* information stored in a database using *RWSNSim*.

C. Execution of RWSN_UnderWater using RWSNSim

After the construction of *RWSN_UnderWater* network, the user can start its execution. Before starting the network execution, *RWSNSim* demands a set of parameters and thresholds which showed in figure 2 and detailed in the methodology proposed in (Rouainia et al., 2022) like α (which is an energy charge threshold in sensor nodes used by the zone agent Ag_k to apply the mobility), γ (which is a number of active sensor nodes threshold in a zone Z_k used by the zone agent Ag_k to apply the mobility), and λ (which is a number of active sensor nodes threshold in a zone Z_k used by the controller agent Ag_{Ctrl} to apply the resizing of zones). After that, two windows appear: the simulation graph window and the execution report window.

1) Simulation graph window: it displays the different *RWSN_UnderWater* network entities and their distribution in the network and shows the communication between them in real time. Figure 7 presents the simulation graph captured during the simulation of *RWSN_UnderWater* network.

2) Execution report window: It is a window containing the different sensing data values received by the controller agent from zone agents. Figure 8


RWSNs...
List of Reconfigurable Wireless Sensor Networks (RWSNs) :
<< BACK
Open an RWSN
Add a new RWSN
Remove an RWSN
RWSN_UnderWater Informations ...
RWSN_UnderWater Informations
Controller Agent : AgCtrl(500,550,10)
Zones Inforamations : 2 Zones
Sinks Inforamations : 6 Sinks
Nodes Inforamations : 40 Nodes
<< BACK
Controller Agent
Add Zone
Modify Zone
Remove Zone
Start Execution

Name	NbZones	NbAgents	NbSinks	NbNodes
RWSN_Forest	4	85	12	80
RWSN_Home	1	12	2	10
RWSN_UnderWater	2	43	6	40

Zone	Zone Agent	Coordinates
Z1	Ag1	(0,0,10)
Z2	Ag2	(1000,0,10)

Sink	Coordinates
SN1,1	(325,150,5)
SN1,2	(538,175,6)
SN2,1	(20,222,5)
SN2,2	(882,180,5)

Node	Node Agent	Coordinates	State	Type	Cpr	Cad	CTot
N1,1	Ag1,1	(365,38,5)	Active	Fixed	150.0E0	450.0E0	600.0E0
N1,2	Ag1,2	(535,20,5)	Active	Fixed	150.0E0	450.0E0	600.0E0
N10,1	Ag10,1	(272,225,5)	Active	Fixed	150.0E0	450.0E0	600.0E0
N10,2	Ag10,2	(700,275,5)	Active	Fixed	150.0E0	450.0E0	600.0E0
N11,1	Ag11,1	(178,290,5)	Active	Fixed	150.0E0	450.0E0	600.0E0
N11,2	Ag11,2	(510,280,5)	Active	Fixed	150.0E0	450.0E0	600.0E0
N12,1	Ag12,1	(437,287,5)	Active	Fixed	150.0E0	450.0E0	600.0E0
N12,2	Ag12,2	(576,323,5)	Active	Mobile	150.0E0	450.0E0	600.0E0

Figure 6: *RWSN_UnderWater* information stored in a database using *RWSNSim*.

displays the execution report of *RWSN_UnderWater* network using *RWSNSim*.

D. Notifications & Alerts

During the simulation, many notification and alert windows appear synchronized with various events that occur during the execution of *RWSN_UnderWater* network. There are four types of notification and alert windows defined as follows:

1) Deactivation notification window: It appears when an entity or a zone in the network is deactivated. Figure 9 shows the deactivation notification window which appears when the node $N_{6,1}$ is deactivated.

2) Mobility notification window: It appears when a mobile entity in the network has moved to another position. Figure 9 illustrates the mobility notification window which appears when the mobile sink nodes $SN_{1,2}$ and $SN_{2,2}$ have moved to other positions in *RWSN_UnderWater* network.

3) Resizing notification window: It appears when the controller agent Ag_{Ctrl} applies the resizing of zones between two zones Z_a and Z_b where $a \in [1..NbZ]$ and $b \in [1..NbZ]$. Figure 9 illustrates the resizing notification window which appears when the controller agent Ag_{Ctrl} applies the resizing between Z_1 and Z_2.

4) Alert window: It appears with an alert sound when the controller agent Ag_{Ctrl} received sensing data with values exceeding the sensing data thresholds which are illustrated in figure 2 and detailed in the methodology proposed in (Rouainia et al., 2022). Figure 9 shows the alert window which appears when the controller agent Ag_{Ctrl} received several sensing data with values exceeding the sensing data thresholds ($CO2Threshold = 5\%$ and $CH4Threshold = 10\%$).

E. Draw the Obtained Line Chart of RWSN_UnderWater using RWSNSim

After the simulation, *RWSNSim* provides the obtained results in the form of line charts. Figure 10 displays the obtained line chart of *RWSN_UnderWater*.

Finally, we note that *RWSNSim* is not a specific simulator developed to implement a specific methodology, but rather tends to be more generalized. Indeed, it can simulate WSNs and RWSNs without applying the proposed methodology in (Rouainia et al., 2022). It has also the scability to include new solutions, techniques and routing protocols.

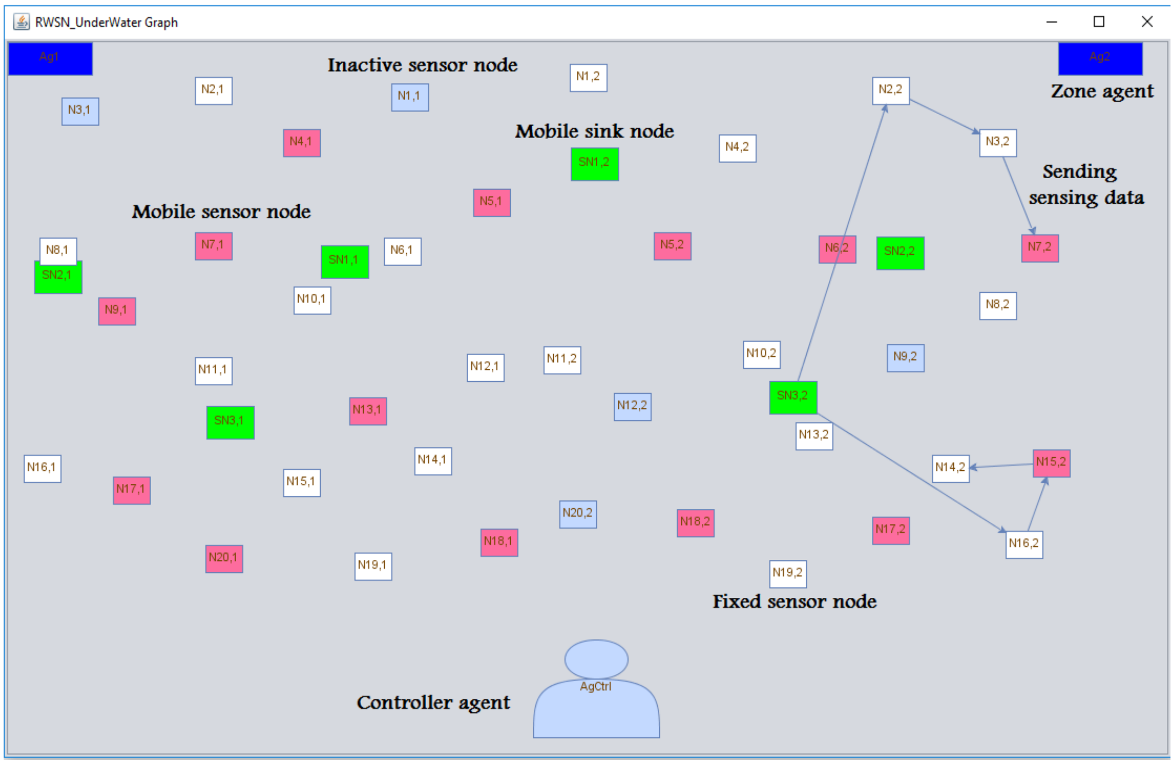
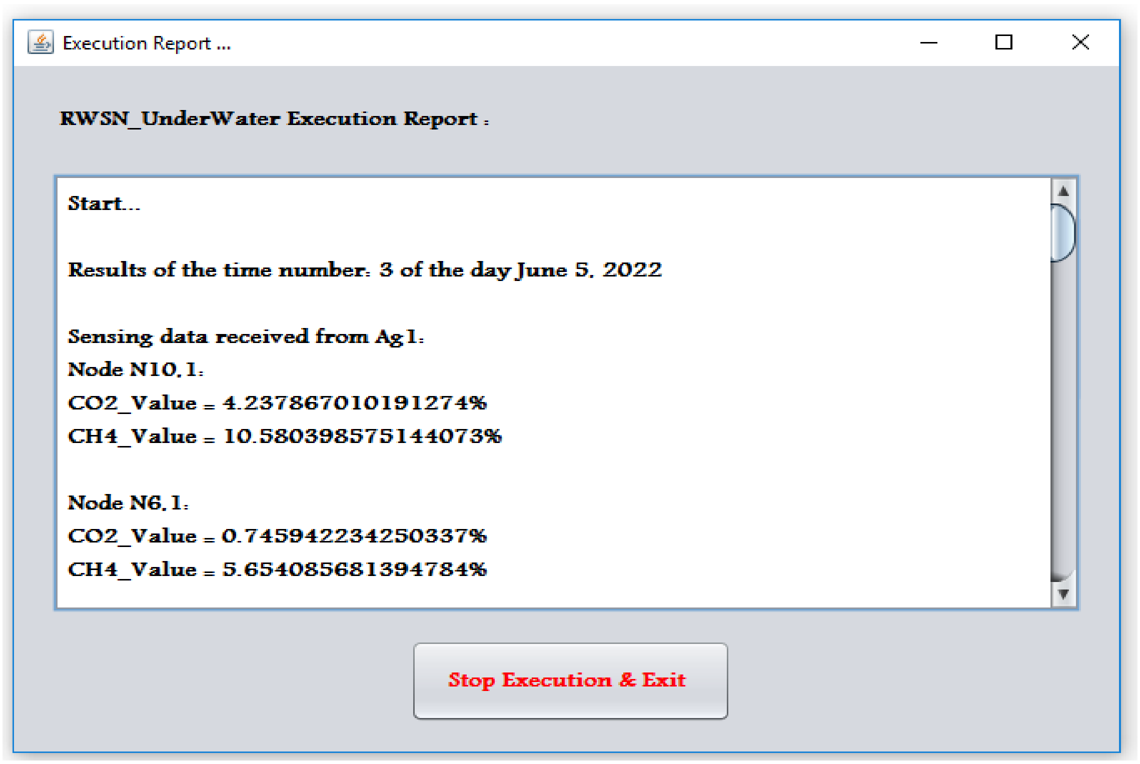


Figure 7: *RWSN_UnderWater* simulation graph using *RWSNSim*.



RWSN_UnderWater Execution Report :

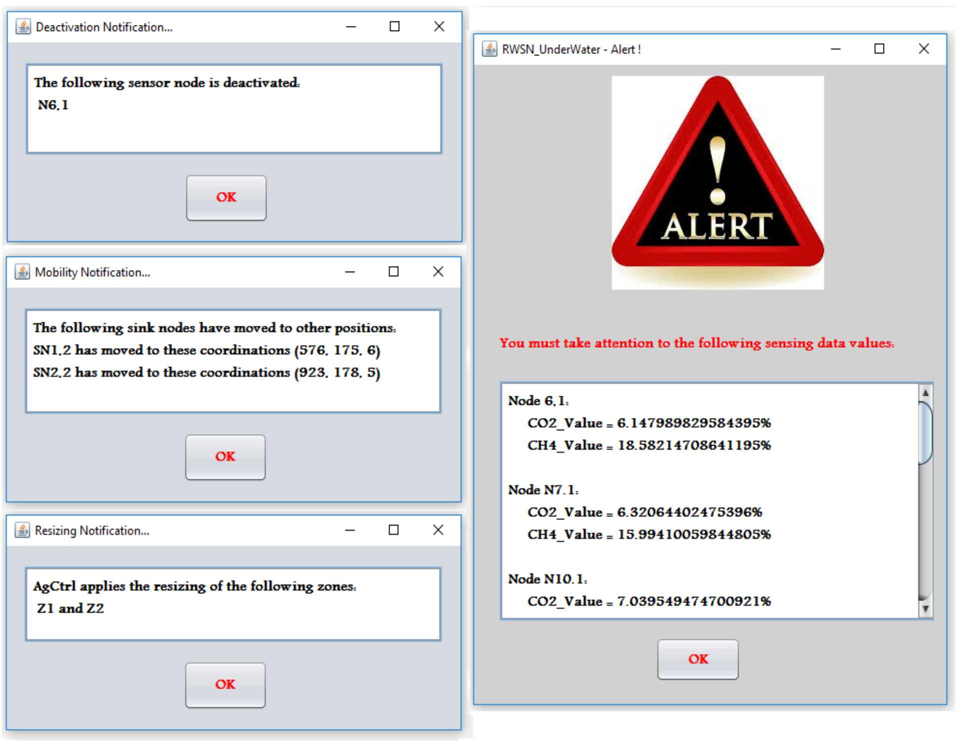
```
Start...

Results of the time number: 3 of the day June 5, 2022

Sensing data received from Ag1:
Node N10,1:
CO2_Value = 4.237867010191274%
CH4_Value = 10.580398575144073%

Node N6,1:
CO2_Value = 0.745942234250337%
CH4_Value = 5.654085681394784%
```

Figure 8: *RWSN_UnderWater* execution report using *RWSNSim*.



The following sensor node is deactivated:
N6.1

The following sink nodes have moved to other positions:
SN1.2 has moved to these coordinations (576, 175, 6)
SN2.2 has moved to these coordinations (923, 178, 5)

AgCtrl applies the resizing of the following zones:
Z1 and Z2

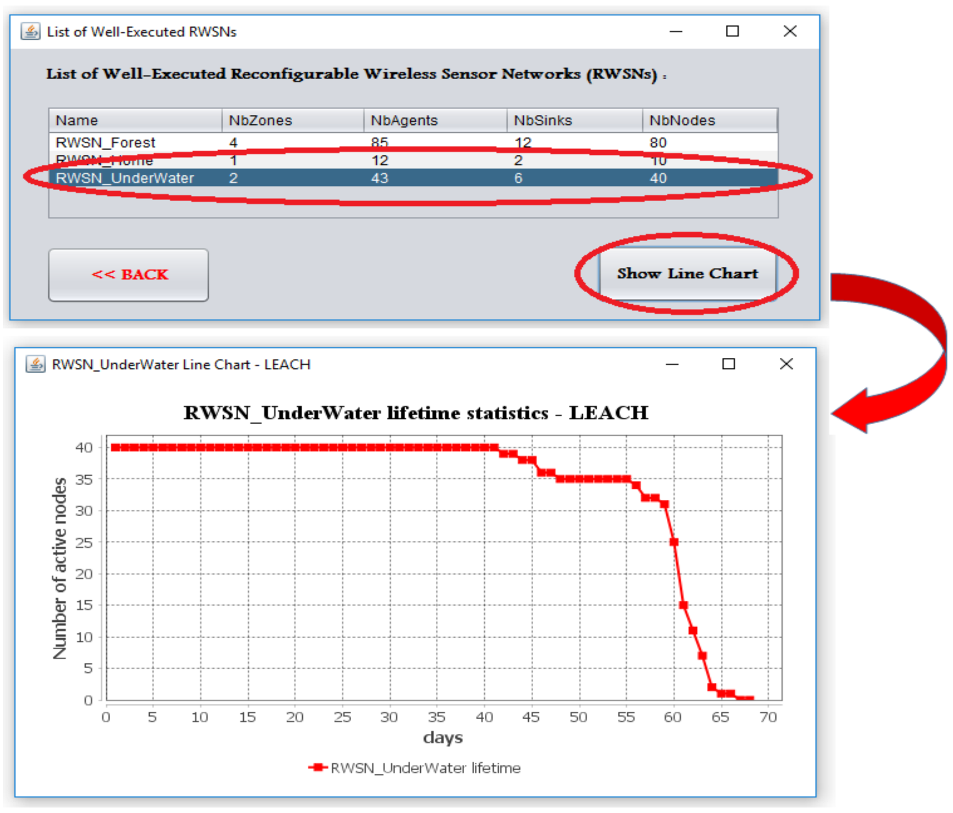
You must take attention to the following sensing data values:

```
Node 6.1:
    CO2_Value = 6.147989829584395%
    CH4_Value = 18.58214708641195%

Node N7.1:
    CO2_Value = 6.32064402475396%
    CH4_Value = 15.99410059844805%

Node N10.1:
    CO2_Value = 7.039549474700921%
```

Figure 9: Notifications and alert windows.



List of Well-Executed Reconfigurable Wireless Sensor Networks (RWSNs) :

Name	NbZones	NbAgents	NbSinks	NbNodes
RWSN_Forest	4	85	12	80
RWSN_Home	1	12	2	10
RWSN_UnderWater	2	43	6	40

Figure 10: The obtained line chart of RWSN_UnderWater.

5 CONCLUSION

In this paper, we proposed a new discrete-event simulator designed for WSNs and RWSNs named *RWSNSim*. We described the provided services by the proposed simulator. We also modeled the architecture of WSNs and RWSNs used by *RWSNSim*. Furthermore, we modeled the databases used by *RWSNSim*. The proposed simulator permits simulating WSNs and RWSNs with hundreds of entities. In the future, we will strive to improve the proposed simulator by adding new routing protocols, new solutions and techniques, new features, and new types of charts.

REFERENCES

Agrawal, D. P. (2017). Sensor nodes (sns), camera sensor nodes (c-sns), and remote sensor nodes (rsns). In *Embedded Sensor Systems*, pages 181–194. Springer.

GloMoSim (2020). GloMoSim Simulator Projects : Online Network Simulator : Network Simulation Tools. https://networksimulationtools.com/glomosim-simulator-projects/. [Online; accessed 27-March-2022].

Grichi, H., Mosbahi, O., and Khalgui, M. (2015). Rocl: New extensions to ocl for useful verification of flexible software systems. pages 45–52.

Grichi, H., Mosbahi, O., and Khalgui, M. (2016a). A development tool chain for reconfigurable wsns. In Fujita, H. and Papadopoulos, G. A., editors, *New Trends in Software Methodologies, Tools and Techniques - Proceedings of the Fifteenth SoMeT_16, Larnaca, Cyprus, 12-14 September 2016*, volume 286 of *Frontiers in Artificial Intelligence and Applications*, pages 101–114. IOS Press.

Grichi, H., Mosbahi, O., Khalgui, M., and Li, Z. (2016b). Rwin: New methodology for the development of reconfigurable wsn. *IEEE Transactions on Automation Science and Engineering*, PP:1–17.

Hafidi, Y., Kahloul, L., Khalgui, M., Li, Z., Alnowibet, K., and Qu, T. (2020). On methodology for the verification of reconfigurable timed net condition/event systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 50(10):3577–3591.

JSim (2022). GitHub - nmcl/JavaSim: JavaSim simulation classes and examples. https://github.com/nmcl/JavaSim. [Online; accessed 27-March-2022].

Khriji, S., Houssaini, D., Kammoun, I., and Kanoun, O. (2018). *Energy-efficient techniques in wireless sensor networks: Technology, Components and System Design*, pages 287–304.

Nayyar, A. and Singh, R. (2015). A comprehensive review of simulation tools for wireless sensor networks (wsns). pages 19–47.

NS3 (2022). ns-3 : a discrete-event network simulator for internet systems. https://www.nsnam.org/. [Online; accessed 26-March-2022].

OMNeTPP (2019). OMNeT++ Discrete Event Simulator. https://omnetpp.org/. [Online; accessed 27-March-2022].

Rajan, C., Geetha, K., Priya, C. R., Geetha, S., and Manikandan, A. (2015). A simple analysis on novel based open source network simulation tools for mobile ad hoc networks. volume 5, pages 716–721.

Rouainia, H., Grichi, H., Kahloul, L., and Khalgui, M. (2020). 3d mobility, resizing and mobile sink nodes in reconfigurable wireless sensor networks based on multi-agent architecture under energy harvesting constraints. In *Proceedings of ICSOFT 2020*, volume 97, pages 394–406. ScitePress.

Rouainia, H., Grichi, H., Kahloul, L., and Khalgui, M. (2022). New energy efficient and fault tolerant methodology based on a multi-agent architecture in reconfigurable wireless sensor networks. pages 405–417.

RWiN (2016). RWiN-Project: New Solutions for Reconfigurable Wireless Sensor Networks. https://lisi-lab.wix.com/rwinproject. [Online; accessed 26-March-2022].

RWSNSim (2022). RWSNSim: Reconfigurable Wireless Sensor Networks Simulator. https://hanenerouainia.wixsite.com/rwsnsim. [Online; accessed 20-April-2022].

Vijayalakshmi, S. and Muruganand, S. (2018). *Wireless Sensor Network: Architecture - Applications - Advancements*. Mercury Learning & Information, 1 edition.