Modeling Energy Conservation in Wireless Sensor Network using Cell-DEVS

Victor Soto
Department of Systems and
Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON. K1S-5B6
Canada
vsoto101@uottawa.ca

ABSTRACT

Simulation is a tool for studying complex systems. When simulating a process, we always start from the observation of a real system, and then create a model. The DEVS formalism provides a framework for the construction of hierarchical models in a modular manner. DEVS has been extended with Cell-DEVS to accommodate cellular automata. Timed Cell-DEVS is a formalism based on DEVS for the simulation of cellular models. As the development of computer technique, it is possible to execute the model's instructions to generate its behavior by computer system. Cellular automata can be used to model various physical phenomena. In this term paper, we will create a Cell-DEVS model to simulate a Sensor Network while sending/receiving a message using the topology control method for energy conservation and implement CD++ to analysis its result.

Keywords
DEVS; topology control; cluster head

INTRODUCTION

Modeling and Simulation is a popular tool for studying many different types of complex systems. The need for such a tool has arisen with the advent of systems that may be too big, small, costly, dangerous or time consuming to observe directly. Discrete Event Systems (DEVS) is a popular formalism for describing simulations and can be used to construct simulations of real-world phenomena. Fundamentally, the DEVS formalism is implemented by subdividing a system into atomic models, each with their own state and behavior, and modeling the interaction between these models. This type of modeling allows for hierarchical construction of complex models and is a powerful simulation tool for many different types of simulation. To further extend the simulation capabilities of the system, extensions were specified to accommodate different simulation techniques.

This study will specifically utilize the Cell-DEVS specification which allows for the implementation of cellular automata in DEVS. This study aims to demonstrate how Cell-DEVS can be effectively used to create a working model of a physical system. Specifically, a cellular automata model for the conservation of energy in a wireless sensor network will be define and tested using the CD++ toolkit. Additionally, new techniques for simulation will be explored and the performance advantages will be discussed

BACKGROUND

Discrete Event Simulation is a formalism that allows for the modeling and simulation of many different types of systems. One of the goals of DEVS is to be able to separate the modeling and simulation aspects of a given problem. Using hierarchies of atomic components, DEVS is able to abstract these aspects and increases usability and interoperability of simulations.

Atomic models form the basis of any DEVS model [2]. Atomic models consist of input and output ports along with internal states and behaviors. Using time advance functions, a state transition can be triggered and output is generated.

Additionally, external events can be received on the input ports and cause external transitions to be executed within the model. The overarching idea is that models only interact through ports and never know anything about other models other than what is provided on the input ports.

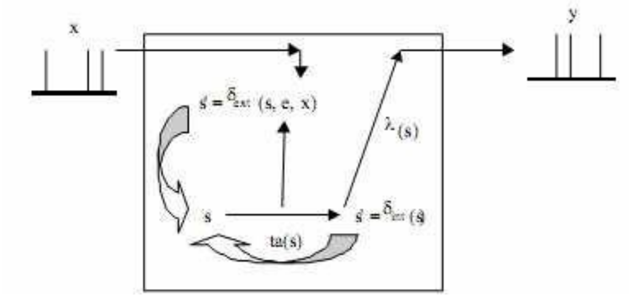


FIGURE 1 – ATOMIC DEVS MODEL

Coupled models are DEVS models that consist of multiple atomic models that are connected through their input and output ports in a specific way. Using the model specification (.ma) file, the specifics of the interconnections between the models and how these models interact with the output of the simulation.

To extend DEVS to include cellular automata phenomena, the Cell-DEVS [3] framework was created. This framework allows for the implementation of cellular automata models and includes functionality for timing delays. It is worth noting that each cell in the cellular automata created in Cell-DEVS is implemented as a DEVS atomic model and the cellular automata can be thought of as a large interconnection of DEVS atomic models.

Each cell in a Cell-DEVS model has inputs coming from each of its neighboring cells. It uses these inputs to compute its next state through the local computing function. The state of the cell is transmitted to other cells using the output port of the model. Output can appear on the output port after a specified delay.

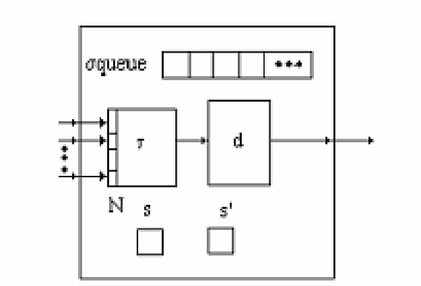


FIGURE 2 – ATOMIC CELL-DEVS MODEL

A coupled Cell-DEVS model is created by interconnecting these cells and specifying the rules with which these cells interact. Generally, the Cell-DEVS models contains many interconnected atomic DEVS models and each cell is connected through input and output ports to cells in its neighborhood. For the purposes of this study, it is useful to note that certain cells can have behaviors that are non-general, such as for border cells, it is possible and useful to specify custom rules.

To create these models, the CD++ modeling environment is used. This framework, developed in C++ implements the DEVS formalism and has been extended to include an implementation of Cell-DEVS. Once models have been written, they can be executed using the CD++ environment. DEVS atomic models are generally written in C++ and included as part of a class hierarchy in a CD++ executable, whereas Cell-DEVS models could function this way or use standard cells with specified rules. There have been many extensions to the DEVS formalism that are worth noting for the purposes of this study. Specifically, the Parallel DEVS framework will be used to implement state variables and specific ports for the later part of the study. Additionally, real-time, embedded and traffic simulation engines have been implemented in DEVS. Since the study presented mainly focuses on the Cell-DEVS models, this framework will be covered in greater detail.

RISE CLIENT

For user's point of view, RISE [7] is intended to be easy to use. RISE supports different CD++ versions of DEVS/Cell-DEVS formalism, including DCDpp for normal and distributed simulation and Lopez with different ports/variables. You do not need to install complicated simulation software, the only need to choose the right RISE API, upload the model file and run it online with standard HTTP requests, then retrieve simulation results from the website.

Most modeling and simulation methods run on single-user workstations, which normally cost too much time of installation and configuration of all the software and dependencies needed by the simulation. It is better to realize a much easy way to remote access of the simulation resources with web service interfaces, improving data accessibility, interoperability and user experience. In this way, users can reuse and share simulation resources on site without worrying about the capability of their local machine CPU or memory. Plus, with the help of advanced distributed simulation technologies, the simulation can be executed on distributed computers via communication networks, which can further improve interoperability and speed up the execution time.

RESTful WS can solve these issues. It is to transfer message representations of Web resources using a set of uniform stateless operations. The resources here are not only files, but also a query, a table, or any concepts. Access to RESTful WS is through Web resources (URIs) and XML messages using HTTP methods (GET, PUT, POST and DELETE). Its strengths of simplicity, efficiency and scalability make RESTful WS an excellent candidate to perform remote simulation. Based on these ideas, in, the authors presented the first existing RESTful Interoperability Simulation Environment (RISE) middleware. The main objective of RISE is to support interoperability and mash-ups of distributed simulations regardless of the model formalism, model language or simulation engine. RISE is accessed through Web resources (URIs) and XML messages using HTTP methods. Implementations can be hidden in resources, which are represented only via URIs. Users can run multiple instances as needed, which are persistent and repeatable by specific URIs. The HTTP methods are typical four types: GET (to read a resource), PUT (to create or update a resource), POST (to append data to a resource), and DELETE (to remove a resource).

In each HTTP response (no matter from which URI), the Response Status is very informative. Normally, it means the request is successful if you see 200 (OK) and 201 (Created); otherwise, some error or unexpected things would have happened, such as 400(bad request), 401(unauthorized), 403(forbidden), 404(not found), 406(not acceptable), 501(not implemented).

DEFINING ENERGY CONSERVATION MODEL

DEVS and Cell-DEVS have been used to model and simulate energy conservation systems.

In this paper [4] we are dealing with topology control problem in Wireless Sensor Node (WSN) using Block cellular automata which in turn help reducing the power consumption. Using block cellular automata, we are able to divide number of cells into particular number of blocks and thus control the states of a group of entities together, instead of one.

The main focus of any sensor network is how duty cycle can be controlled. For energy conservation, it is required to put radio receiver in sleep or active mode, accordingly when communication is required and when not.

The active nodes are selected by topology control method. The topology control mechanism means a collection of nodes which monitors a particular region consuming less energy

Random deployment is preferable in a practical scenario but to keep the things simple we consider a uniform grid where each cell contains only one sensor node.

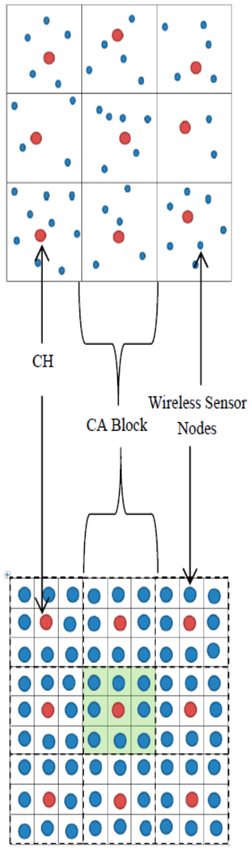


Figure 3. Neighborhood

A Moore type neighborhood is considered here; where each sensor node is surrounded by other 8 neighborhood nodes. The state of that sensor node (either Active or Idle) depends on the states of the neighborhood nodes. Here a simple rule is followed, i.e. a sensor node goes to IDLE state (for that particular round) if at least 2 of its neighbors are ACTIVE. In the next round the node runs a check again and decides on its state depending on the states of neighborhood, at that instant.

We try to address these issues in our proposal. We group/cluster the sensor nodes into blocks. We use grid clustering technique for that. We then apply the same CA neighborhood based rules

The rule that governs the transition of a certain block is:

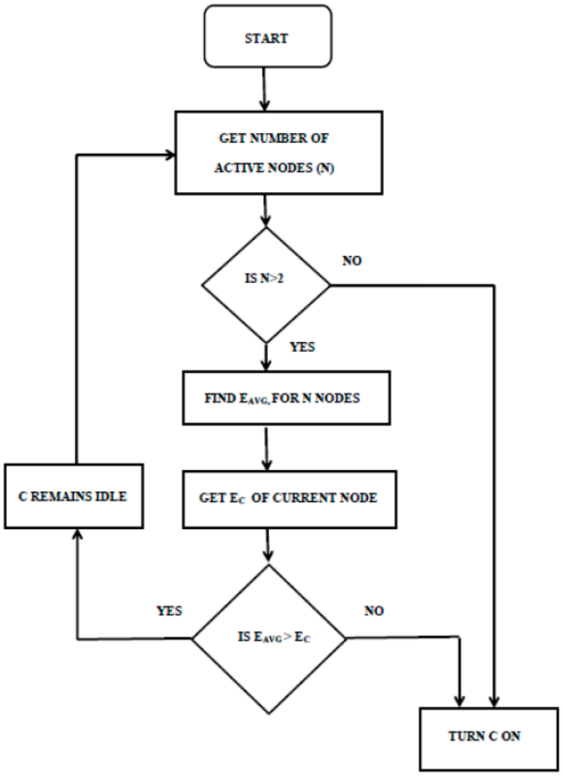

START
GET NUMBER OF ACTIVE NODES (N)
IS N>2
NO
YES
FIND E_{AVG} FOR N NODES
C REMAINS IDLE
GET E_C OF CURRENT NODE
YES
IS E_{AVG} > E_C
NO
TURN C ON

Figure 4. Flowchart

N=Number of active neighbor nodes (CH)

E_{AVG}=Mean residual energy of the active neighbors (CH)

E_C=Residual energy of the current CH

C=CH current state

IF N>2 AND $E_{AVG}>E_C$ THEN C=” IDLE”

ELSE C=” ACTIVE” (See Fig 4).

On Table 1 we can see the variables used in the experiment for the states in CD++ along with the colors for each cell value included in the EnergyCon.pal file. The first three values (0-2) are used to define the energy level of the signal, they go from low energy signal to high energy signal along with a blue color going lighter. The fourth value (3) is used for an active state for the cluster head with a red color, and the last value (4) is used for a cluster head with an idle state along with black cell color.

Cell State	State Name	Color
0	Low Energy Signal	⬜
1	Mid Energy Signal	🟦
2	High Energy Signal	🟦
3	Active CH	🟥
4	Idle CH	⬛

Table 1. Variable definitions

This experiment follows completely the flowchart from Figure 3. The grid cluster is modeled using Cell-DEVS. The model has one layer with dimensions of 20x20, where each cell represents an open area and there are 36 cluster heads (CH) as shown on Figure 5.

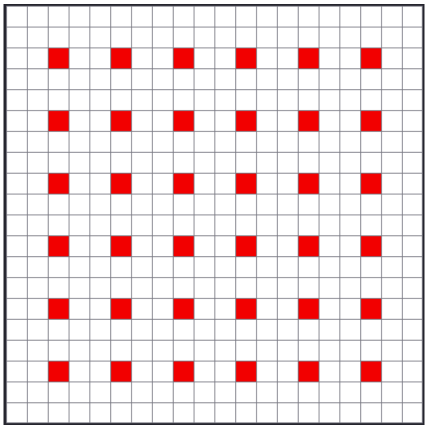


Figure 5. Grid cluster

The system can be summarized as follows:

- A two dimensional area where 36 sensor controllers are located
- Each sensor controller or cluster head is in one of two states: active or idle
- Each cluster head is only affected by the residual energy of the sensors being controlled
- If the neighborhood being controlled by the cluster head has enough residual energy the controller stays in an active state
- If the residual energy of the neighborhood controlled by the cluster head is low, then the sensor goes to an idle state

CD ++ IMPLEMENTATION

The neighborhood in this experiment is defined as shown in Figure 3 using 9 blocks and each block taking 9 cells with a CH in the center. The important cells to consider here are the CH cells, everything else is considered as open space where the wireless signal energy varies at random.

neighbors : EnergyCon(-4,-4) EnergyCon(-3,-4) EnergyCon(-2,-4) EnergyCon(-1,-4) EnergyCon(0,-4) EnergyCon(1,-4) EnergyCon(2,-4) EnergyCon(3,-4) EnergyCon(4,-4)

neighbors : EnergyCon(-4,-3) EnergyCon(-3,-3) EnergyCon(-2,-3) EnergyCon(-1,-3) EnergyCon(0,-3) EnergyCon(1,-3)

EnergyCon(2,-3) EnergyCon(3,-3) EnergyCon(4,-3)

.

.

.

neighbors : EnergyCon(-4,4) EnergyCon(-3,4) EnergyCon(-2,4) EnergyCon(-1,4) EnergyCon(0,4) EnergyCon(1,4)

EnergyCon(2,4) EnergyCon(3,4) EnergyCon(4,4)

Figure 6. Definition of neighborhood

To follow the flowchart on Figure 2, the following rules are used (Figure 7), where we can define the state of the CH if the number of active neighbor CH is higher than 2 and the average residual energy from all the neighborhoods is higher than the average residual energy from the actual CH.

rule : 4 100 { (0,0) = 3 and ((-3,-3)+(-3,0)+(-3,3)+(0,-3)+(0,3)+(3,-3)+(3,0)+(3,3)) >= 9 and ((-2,-2)+(-1,-2)+(0,-2)+(1,-2)+(2,-2)+(2,-1)+(2,0)+(2,1)+(2,2)+(1,2)+(0,2)+(-1,2)+(-2,2)+(-2,1)+(2,0)+(-2,-1))/16 >= ((0,-1)+(1,-1)+(1,0)+(1,1)+(0,1)+(-1,1)+(-1,0)+(-1,-1))/8 }

rule : 4 100 { (0,0) = 4 and ((-3,-3)+(-3,0)+(-3,3)+(0,-3)+(0,3)+(3,-3)+(3,0)+(3,3)) >= 9 and ((-2,-2)+(-1,-2)+(0,-2)+(1,-2)+(2,-2)+(2,-1)+(2,0)+(2,1)+(2,2)+(1,2)+(0,2)+(-1,2)+(-2,2)+(-2,1)+(2,0)+(-2,-1))/16 >= ((0,-1)+(1,-1)+(1,0)+(1,1)+(0,1)+(-1,1)+(-1,0)+(-1,-1))/8 }

Figure 7. Active CH rules

In the next rule (Figure 8) we define that if the CH is Idle and the values of the neighborhood average residual energy is now lower than the residual energy surrounding the actual CH the state becomes Active

rule : 3 100 { (0,0) = 4 and ((-2,-2)+(-1,-2)+(0,-2)+(1,-2)+(2,-2)+(2,-1)+(2,0)+(2,1)+(2,2)+(1,2)+(0,2)+(-1,2)+(-2,2)+(-2,1)+(2,0)+(-2,-1))/16 <= ((0,-1)+(1,-1)+(1,0)+(1,1)+(0,1)+(-1,1)+(-1,0)+(-1,-1))/8 }

Figure 8. Idle CH rules

In figure 9 we can see the initial values used in the simulation, where all the cluster heads are in an active state and the energy level of the neighborhood is not shown yet.

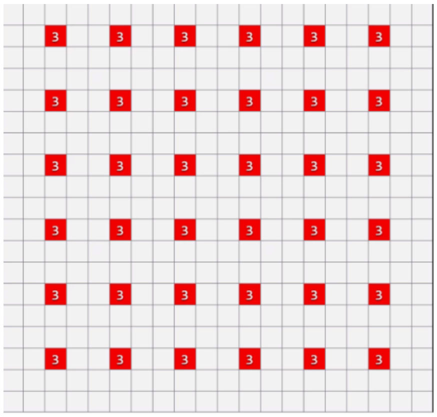


Figure 9. Initial values

For the next step we see (Figure 10) how the energy level of the neighborhood is shown now and then the values of energy start changing at random for simulation purposes.

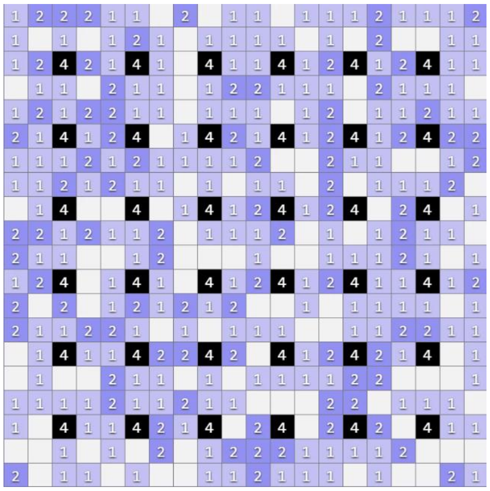


Figure 10. Simulation starting

SIMULATION RESULTS

For a good view of the results a simulation with a time of 00:00:05:00 it’s enough and to understand better the above explained we will use an example of the results obtained in the simulation focusing on a specific CH and its neighborhood.

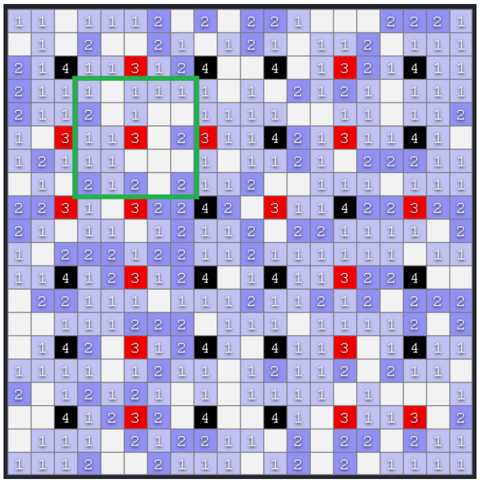


Figure 10. Simulation result example (First step)

In the first step if we focus on the CH inside the green square shown on figure 4, we can see how both of the rules necessary of it to become Idle are true.

- It has more than 2 neighbor CH with an Active state
- The average residual energy from the neighbor CH (1.0625) is higher than the one from the actual CH (0.375)

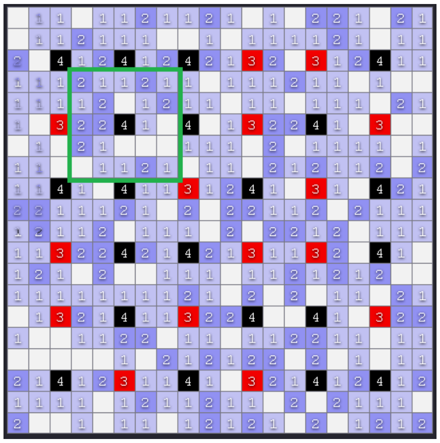


Figure 11. Simulation result example (Second step)

In figure 6 we can see the next step to look how the state changes from Active to Idle.

DEFINING ADVANCED MODEL

Reading this paper [6], arose the idea of an advanced model to show a simulation of how the energy conservation topology control method would work while the sensors where in activity.

On Table 2 we can see the variables used in the advanced model experiment for the states in CD++ along with the colors for each cell value included in the EnergyCon.pal file.

The first three values (0-2) are used to define the energy level of the signal, they go from low energy signal to high energy signal along with a blue color going lighter. The fourth value (3) is used for an active state for the cluster head with a red color, and the last value (4) is used for a cluster head with an idle state along with black cell color.

Almost all the values are the same but there are 3 values added. The value of 5 is used as a state for when the cluster head is sending a message requested, value 6 is used for when a cluster head is repeating the message sent by the CH with a state of 5, value 7 is for when the message is received and an acknowledge message is sent back to the sender, and value 8 is for when a CH is requesting a message (information obtained) from another CH.

Cell State	State Name	Color
0	Low Energy Signal	
1	Mid Energy Signal	🟦
2	High Energy Signal	🟦
3	Active CH	🟥
4	Idle CH	⬛
5	Sending Message	🟩
6	Repeating Message	🟪
7	Acknowledge	🟨
8	Requesting Message	▨

Table 2. Variable definitions

This experiment follows completely the flowchart from Figure 3. The grid cluster is modeled using Cell-DEVS.

The model has one layer with dimensions of 20x20, where each cell represents an open area and there are 36 cluster heads (CH).

Simulations were made for 4 different scenarios, and each one has a different set of initial values which will be shown in the next figures

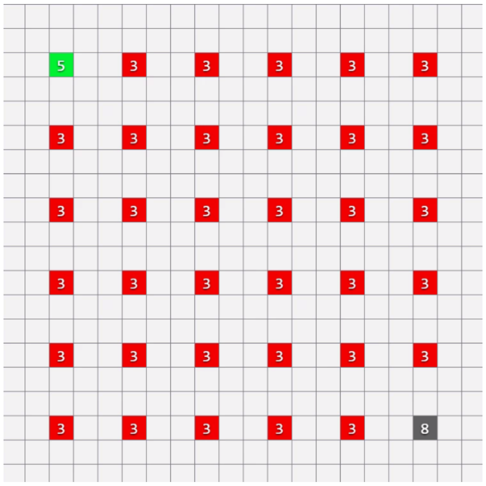


Figure 12. Initial Values (Scenario #1)

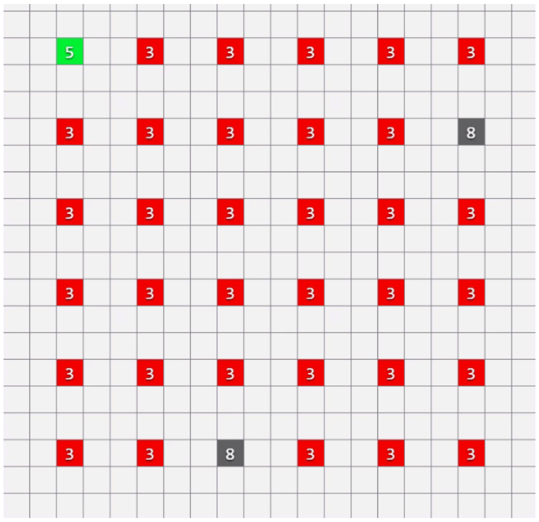


Figure 13. Initial Values (Scenario #2)

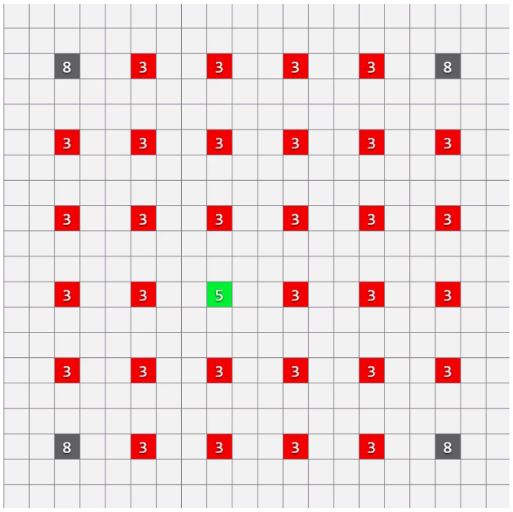


Figure 14. Initial Values (Scenario #3)

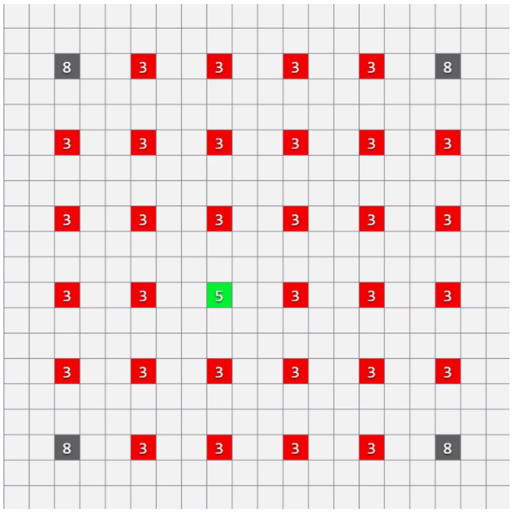


Figure 15. Initial Values (Scenario #4)

In figure 16 we can see the new rules added on scenarios 1, 2 and 3 for the message to be delivered. In the first 3 scenarios the CHs continue to perform the topology control method, so the message waits for a CH to be in an active state to continue repeating it.

```
rule : 6 100 {(0,0) = 3 and statecount(5) >= 1}
rule : 6 100 {(0,0) = 3 and statecount(6) >= 1}
rule : 7 100 { (0,0) =6 and statecount(8) >= 1}
rule : 3 100 { (0,0) =5 and statecount(7) >= 1}
rule : 7 100 { (0,0) =6 and statecount(7) >= 1}
rule : 3 100 { (0,0) =8 and statecount(6) >= 1}
rule : 5 100 {(0,0) = 5}
rule : 6 100 { (0,0) = 6}
rule : 8 100 { (0,0) = 8}
rule : 3 100 {(0,0) = 7}
```

Figure 16. Rules added for scenarios #1, #2 and #3

In figure 17 we can see the rules for scenario #4, there is a small change because of the results obtained in scenario #3. Here the CHs that are in an inactive state change to an active state to deliver the message faster and to ensure al the receivers get the message properly.

```
rule : 6 100 {(0,0) = 3 and statecount(5) >= 1}
rule : 6 100 {(0,0) = 3 and statecount(6) >= 1}
rule : 7 100 { (0,0) =6 and statecount(8) >= 1}
rule : 3 100 { (0,0) =5 and statecount(7) >= 1}
rule : 7 100 { (0,0) =6 and statecount(7) >= 1}
rule : 3 100 { (0,0) =8 and statecount(6) >= 1}
rule : 5 100 {(0,0) = 5}
rule : 6 100 { (0,0) = 6}
rule : 8 100 { (0,0) = 8}
rule : 3 100 {(0,0) = 7}
```

Figure 17. Rules added for scenario #4.

MODEL IMPROVEMENT

To handle the random generation of state variables an additional term was added to each transition rule that contained a probability distribution.

An updated CD++ software was obtained that allows the definition of multiple variables for each cell. The addition of multiple variables will allow the existing model to be expanded to provide additional functionality. The transition functions have been expanded to allow for both the assigning of new values to the internal state variables and for sending the state variables to input ports on the neighboring cells. The transition function is broken into four components:

{port_assignations} [{assignations}] {delay} {condition}

Output ports are assigned using in port_assignations:

~variable_port ≔ value;

The default output port can still be assigned using either ~out or (0,0). Assigning state variables is optional and is done in assignations:

$variable ≔ value;

In both of these cases the value can be the result of any function that returns a value. It should be noted that with the internal value of state variables and the variables sent on the output port can now be different.

With the introduction of multiple variables into CD++ the τ(N) function used in the Cell-DEVS formal specification to define the output value of the transition function must be expanded to reflect both the internal variable transition and the value sent to the output port.

```
rule : { $charge } { $charge := randInt(99);} 100 { $charge <100 }

rule : { $state } { $state := 0;} 100 { $charge < 25 }

rule : { $state } { $state := 1;} 100 { $charge > 25 }

rule : { $state } { $state := 4; } 100 { $state = 3 and truecount >= 8 }

rule : { $state } { $state := 3; } 100 { $state = 4 and truecount <= 8 }

rule : { $state } { $state := 9; } 100 {  %state = 9 }

rule : { $state } { $state := 3; } 100 {  %state != 9 }
```

Figure 18. Rules for the extended version

The initial values where changed to make a different simulation and obtain different results. The values can be found in the energycon.stvalues file. Where the value of 9 represent an obstacle on the network.

SIMULATION RESULTS

For a good view of the results a simulation with a time of 00:00:05:00 it's enough and to understand better the above explained we will follow the simulation focusing on the message sent and acknowledged when received.

First we will go on detail with Scenario #1 and show images of the process, then as scenario #2 and #3 are similar we will only give a short explanation. Finally, Scenario #4 and the new rules will be explained.

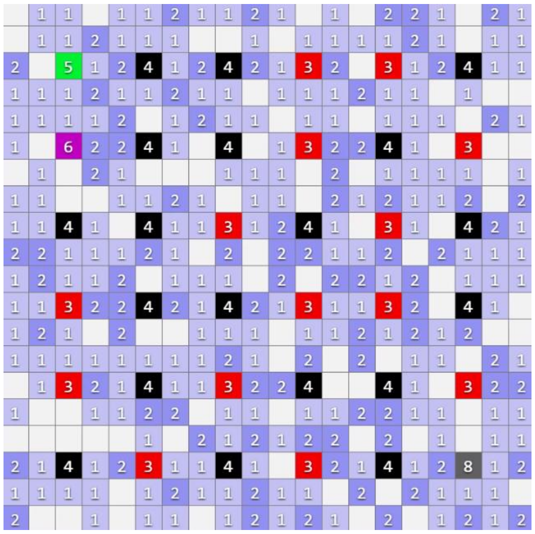


Figure 18. Simulation result follow-up (Scenario #1)

In Figure 18 we can see how the simulation has started and how the sender and receiver are shown, and the message has begun to be repeated, it will be repeated to every CH until it gets to the receiver as shown in Figure 19.

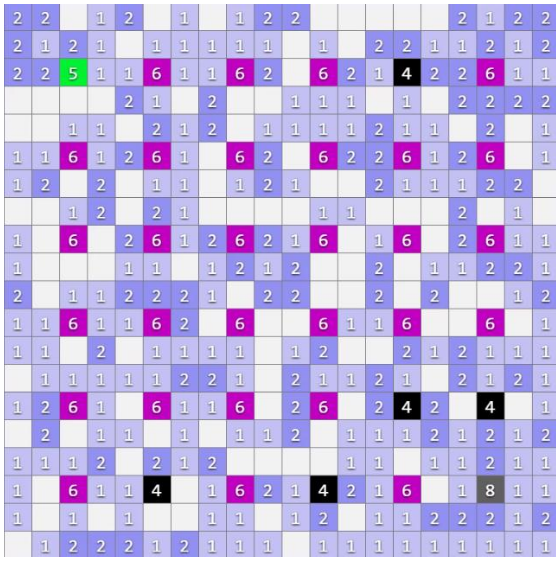


Figure 19. Simulation result follow-up (Scenario #1)

We can see how some of the CH are still in an inactive state, this is because they don't change to repeat the message, the topology control method continues without interruption.

Finally, we can see in Figure 20 how the message has been delivered to the receiver and the acknowledge message has begun to be sent back.

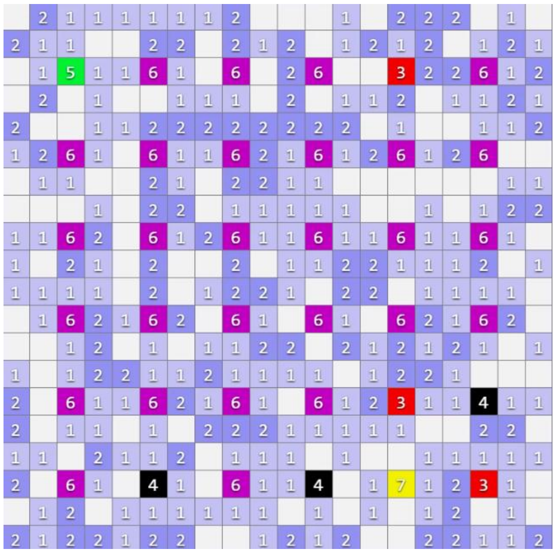


Figure 20. Simulation result follow-up (Scenario #1)

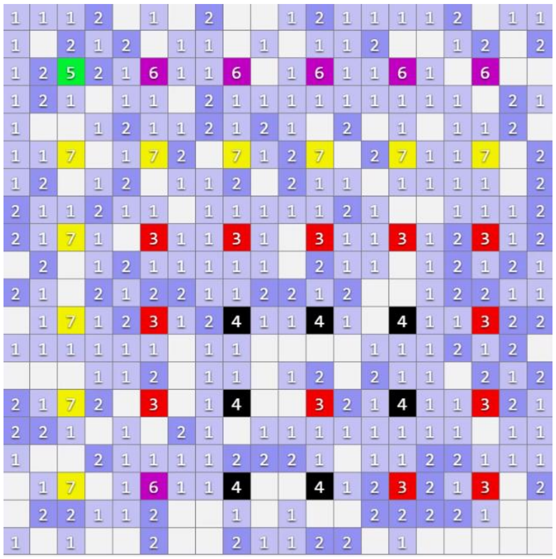


Figure 21. Simulation result follow-up

On Figure 21, we see how the message is being sent back. Once each CH receives the acknowledge message they go back to follow the topology control method normally, even the Sender CH as shown on Figure 22.

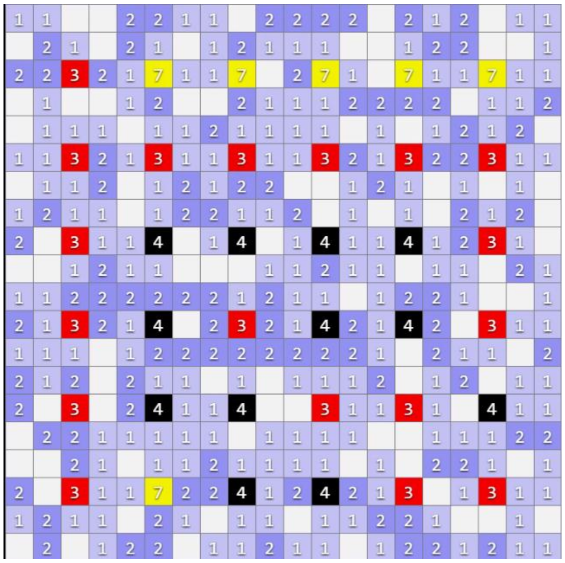


Figure 22. Simulation result follow-up

The results for Scenario #2 are pretty similar, the only thing that changes is the initial values, so this time the path the message follows is different.

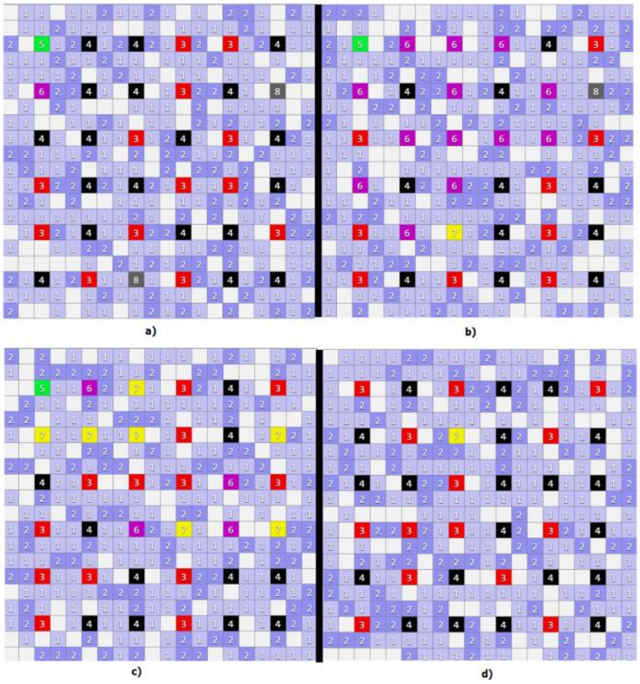


Figure 23. Simulation results Scenario #2; a) first step; b) second step; c) third step; d) fourth step

For Scenario #3 (Figure 14) we added 2 more receivers to the initial values to see if the message was delivered correctly to each one of them and if the sender gets the acknowledge message properly. In figure 24 we can see the message starting to repeat along the CH.

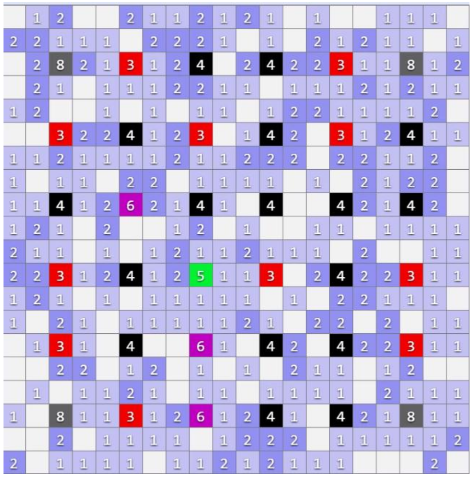


Figure 24. Simulation result follow-up (Scenario #3)

As the message repeating rules have no influence over the topology control method the message starts to repeat in a nonuniform way as seen on figure 25.

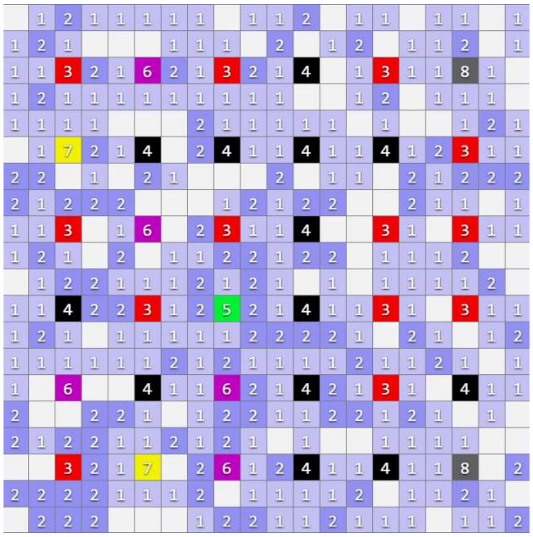


Figure 25. Simulation result follow-up (Scenario #3)

Finally, the simulation for Scenario #3 ends leaving a receiver without getting the message requested as shown on figure 26.

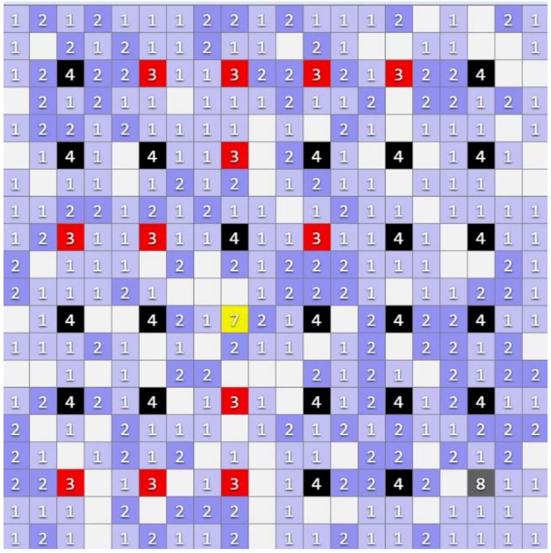


Figure 26. Simulation result follow-up (Scenario #3)

There is a need to change the rules for cases when there are several receivers, for them to receive the message properly and in a faster way the rules for the CHs were changed.

In scenario #4 when there is an immediate neighbor repeating or sending a message, the CHs in an inactive state change to active to repeat the message. To demonstrate this we repeat scenario #3 with the new rules.

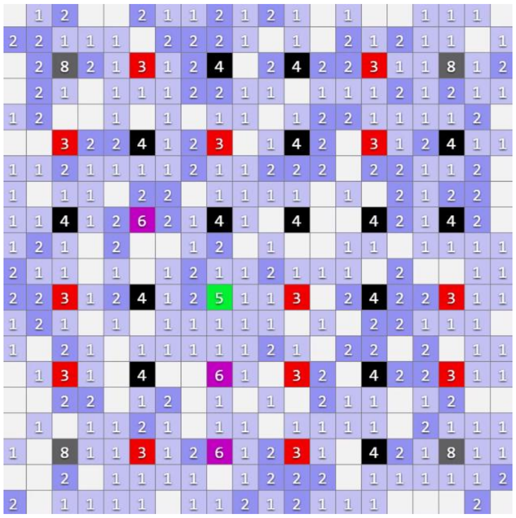


Figure 27. Simulation result follow-up (Scenario #4)

We can see in Figure 27 how the message spreading begins in the same way as scenario #3, but then it gets significantly faster to all the receivers.

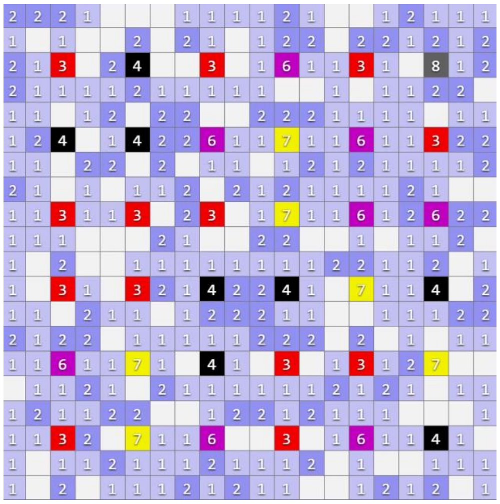


Figure 28. Simulation result follow-up (Scenario #4)

Finally, in figure 28 and 29 we see how every receiver acknowledge receiving the message and the topology control method continues running.

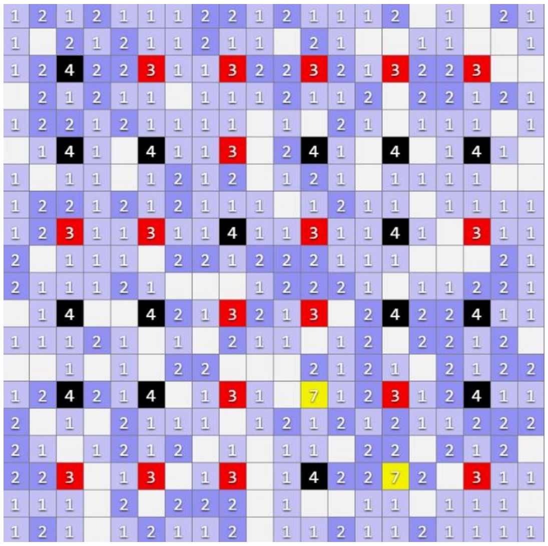


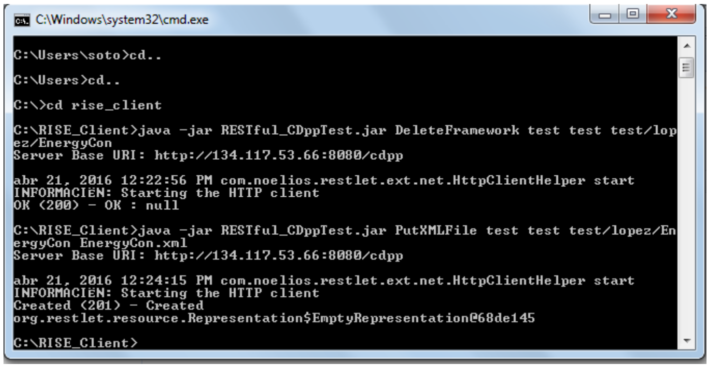
Figure 29. Simulation result follow-up (Scenario #4)

The results on Figure 29 show us that the new rules are more complete than before.

RISE CLIENT COMPARISON

Both models were run using the newest version of CD++ with the help of the RISE client and we have some images of the process below.

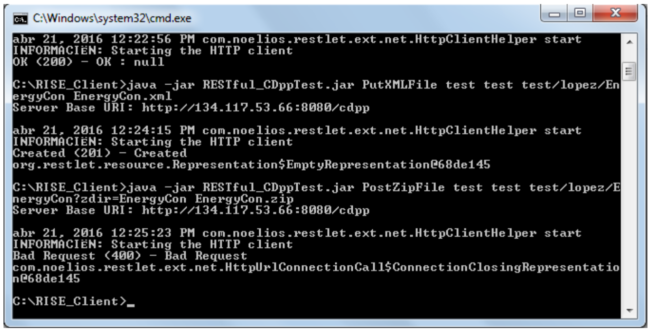
First (Figure 30) we have to get access the RESTful Web server through web resources and xml messages.



```
C:\Users\soto>cd..

C:\Users>cd..

C:\>cd rise_client

C:\RISE_Client>java -jar RESTful_CDppTest.jar DeleteFramework test test test/lop
ez/EnergyCon
Server Base URI: http://134.117.53.66:8080/cdpp

abr 21, 2016 12:22:56 PM com.noelios.restlet.ext.net.HttpClientHelper start
INFORMACIËN: Starting the HTTP client
OK (200) - OK : null

C:\RISE_Client>java -jar RESTful_CDppTest.jar PutXMLFile test test test/lopez/En
ergyCon EnergyCon.xml
Server Base URI: http://134.117.53.66:8080/cdpp

abr 21, 2016 12:24:15 PM com.noelios.restlet.ext.net.HttpClientHelper start
INFORMACIËN: Starting the HTTP client
Created (201) - Created
org.restlet.resource.Representation$EmptyRepresentation@68de145

C:\RISE_Client>
```

Figure 30. Setting up the framework

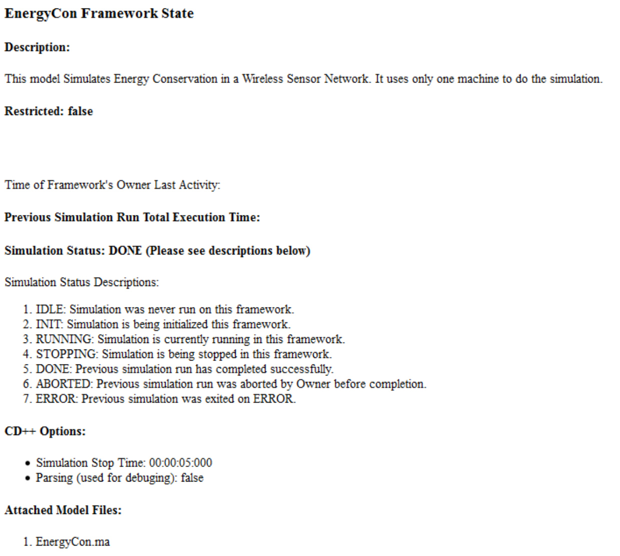
Now we can access the framework where we will see the complete process of the simulation shown and we can get all the information we need. The next step is to upload the Model file (EnergyCon.ma) inside a .zip file to the web server (Figure 31).



```
abr 21, 2016 12:22:56 PM com.noelios.restlet.ext.net.HttpClientHelper start
INFORMACIËN: Starting the HTTP client
OK (200) - OK : null

C:\RISE_Client>java -jar RESTful_CDppTest.jar PutXMLFile test test test/lopez/En
ergyCon EnergyCon.xml
Server Base URI: http://134.117.53.66:8080/cdpp

abr 21, 2016 12:24:15 PM com.noelios.restlet.ext.net.HttpClientHelper start
INFORMACIËN: Starting the HTTP client
Created (201) - Created
org.restlet.resource.Representation$EmptyRepresentation@68de145

C:\RISE_Client>java -jar RESTful_CDppTest.jar PostZipFile test test test/lopez/E
nergyCon?zdir=EnergyCon EnergyCon.zip
Server Base URI: http://134.117.53.66:8080/cdpp

abr 21, 2016 12:25:23 PM com.noelios.restlet.ext.net.HttpClientHelper start
INFORMACIËN: Starting the HTTP client
Bad Request (400) - Bad Request
com.noelios.restlet.ext.net.HttpUrlConnectionCall$ConnectionClosingRepresentatio
n@68de145

C:\RISE_Client>_
```

Figure 31. Upload process

Now we can see the model file has been uploaded and we are ready to start the simulation



EnergyCon Framework State

Description:

This model Simulates Energy Conservation in a Wireless Sensor Network. It uses only one machine to do the simulation.

Restricted: false

Time of Framework's Owner Last Activity:

Previous Simulation Run Total Execution Time:

Simulation Status: DONE (Please see descriptions below)

Simulation Status Descriptions:

1. IDLE: Simulation was never run on this framework.
2. INIT: Simulation is being initialized this framework.
3. RUNNING: Simulation is currently running in this framework.
4. STOPPING: Simulation is being stopped in this framework.
5. DONE: Previous simulation run has completed successfully.
6. ABORTED: Previous simulation run was aborted by Owner before completion.
7. ERROR: Previous simulation was exited on ERROR.

CD++ Options:

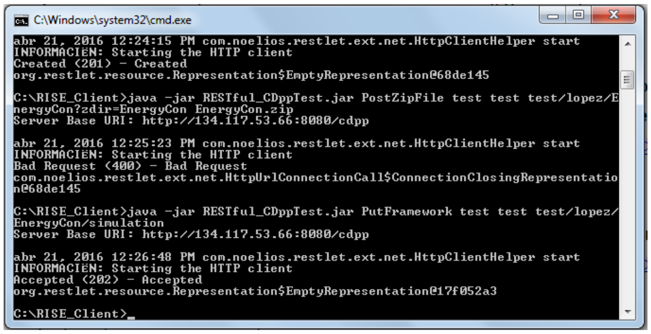
- Simulation Stop Time: 00:00:05:000
- Parsing (used for debuging): false

Attached Model Files:

1. EnergyCon.ma

Figure 31. Framework showing the .ma file uploaded

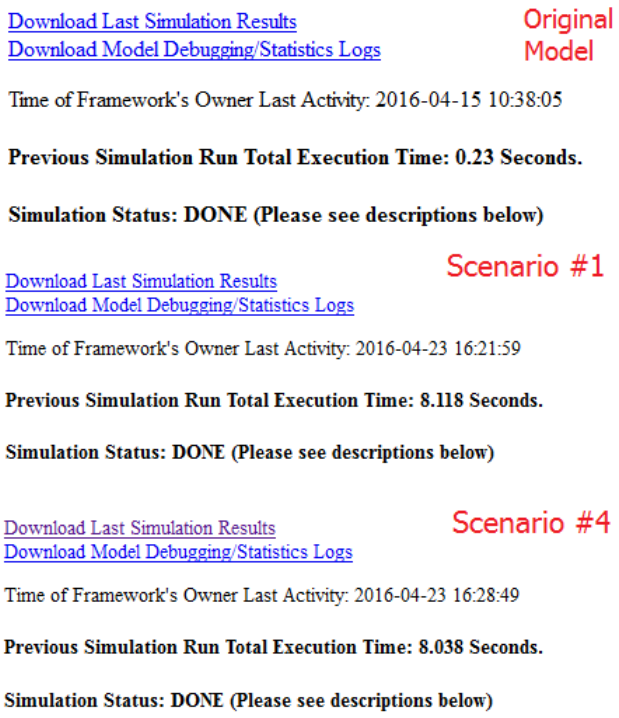
Finally, after we run the simulation (Figure 32) we can download the results obtained and we get the detail of the simulation time.



```
abr 21, 2016 12:24:15 PM com.noelios.restlet.ext.net.HttpClientHelper start
INFORMACIËN: Starting the HTTP client
Created (201) - Created
org.restlet.resource.Representation$EmptyRepresentation@68de145

C:\RISE_Client>java -jar RESTful_CDppTest.jar PostZipFile test test test/lopez/E
nergyCon?zdir=EnergyCon EnergyCon.zip
Server Base URI: http://134.117.53.66:8080/cdpp

abr 21, 2016 12:25:23 PM com.noelios.restlet.ext.net.HttpClientHelper start
INFORMACIËN: Starting the HTTP client
Bad Request (400) - Bad Request
com.noelios.restlet.ext.net.HttpUrlConnectionCall$ConnectionClosingRepresentatio
n@68de145

C:\RISE_Client>java -jar RESTful_CDppTest.jar PutFramework test test test/lopez/
EnergyCon/simulation
Server Base URI: http://134.117.53.66:8080/cdpp

abr 21, 2016 12:26:48 PM com.noelios.restlet.ext.net.HttpClientHelper start
INFORMACIËN: Starting the HTTP client
Accepted (202) - Accepted
org.restlet.resource.Representation$EmptyRepresentation@17f052a3

C:\RISE_Client>_
```

Figure 32. Running simulation

And for comparison purposes we ran the original model and the advanced model and we can see (Figure 33) how the advanced model takes a little more time to complete the simulation, this is because it is more complex and even though it has the same neighborhood it also has some rules added.



Download Last Simulation Results
Download Model Debugging/Statistics Logs

Time of Framework's Owner Last Activity: 2016-04-15 10:38:05

Previous Simulation Run Total Execution Time: 0.23 Seconds.

Simulation Status: DONE (Please see descriptions below)

Download Last Simulation Results
Download Model Debugging/Statistics Logs

Time of Framework's Owner Last Activity: 2016-04-23 16:21:59

Previous Simulation Run Total Execution Time: 8.118 Seconds.

Simulation Status: DONE (Please see descriptions below)

Download Last Simulation Results
Download Model Debugging/Statistics Logs

Time of Framework's Owner Last Activity: 2016-04-23 16:28:49

Previous Simulation Run Total Execution Time: 8.038 Seconds.

Simulation Status: DONE (Please see descriptions below)

Figure 33. Simulation times obtained

CONCLUSION

Cell-DEVS allows to use a cell-based approach to describe systems, and we have seen how it is a very useful technique for modeling and simulating space-shape models. One can use an existing model of any system, start creating and modifying it to get different results, sometimes better ones.

As was presented, a Cell-DEVS model was created to simulate the effect of the topology control in energy conservation for wireless sensor networks using predefined rules as specified in the paper that formed the basis for this study.

Rules for an advanced energy conservation model was created to show how the topology control method works while sending a message from one sensor to another and the model was tested using the latest version of CD++.

Additionally, as results from the advanced model Scenario #3 were not as expected, new rules were added to the model so it has no problems with different initial values.

FUTURE WORK

This project is open to a lot more development and would be greatly aided with 3D simulation such as in Blender. The 3D realization would allow users to see the effects of the energy conservation method more easily and friendly.

Additionaly more advanced rules for how messages are sent and wich paths to follow could be analyzed.

REFERENCES

1. Discrete-Event Modeling and Simulation, Gabriel A.Wainer, 2009
2. B. Zeigler; T. Kim; H. Praehofer: Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Academic Press, 2000.
3. G. Wainer; N. Giambiasi: "Application of the Cell-DEVS Paradigm for Cell Spaces Modeling and Simulation", Simulation, Vol. 71, No. 1, pp. 22-39, January 2001
4. Giuseppe Anastasi, Marco Conti, Mario Di Francesco, Andrea Passarella, “Energy conservation in wireless sensor networks: survey,” Elsevier, 2008.
5. Energy Conservation in Wireless Sensor Network using Block Cellular Automata, International conference on computer communication and Informatics, 2013.
6. Energy saving in wireless sensor networks, Zahra Rezaei, Shima Mobininejad, IJCSES, 2012
7. RISE Manual draft v2.0 – S. Wang, G. Wainer, 2014

ACKNOWLEDGMENTS

This project was completed with the use of the Eclipse plataform, the CD++ toolkit (open source and /lopez versions), RISE client and the support of Professor Gabriel Wainer and his laboratory members