Journal of Simulation

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tjsm20

Discrete Event Systems Specifications Modelling and Simulation of Wireless Networking Applications

Misagh Tavanpour, Baha Uddin Kazi & Gabriel Wainer

To cite this article: Misagh Tavanpour, Baha Uddin Kazi & Gabriel Wainer (2022) Discrete Event Systems Specifications Modelling and Simulation of Wireless Networking Applications, Journal of Simulation, 16:1, 1-25, DOI: 10.1080/17477778.2020.1750313

To link to this article: https://doi.org/10.1080/17477778.2020.1750313

Published online: 28 Apr 2020.

JOURNAL OF SIMULATION
2022, VOL. 16, NO. 1, 1–25
https://doi.org/10.1080/17477778.2020.1750313

THE OPERATIONAL RESEARCH SOCIETY
Taylor & Francis
Taylor & Francis Group

ORIGINAL ARTICLE

Discrete Event Systems Specifications Modelling and Simulation of Wireless Networking Applications

Misagh Tavanpour, Baha Uddin Kazi and Gabriel Wainer

Department of Systems and Computer Engineering, Carleton University Centre for Visualization and Simulation (V-sim), Carleton University, Ottawa, ON, Canada

ABSTRACT

We discuss the use of formal discrete-event modelling and simulation for networking applications, in particular, for mobile networks and Wireless Sensor Networks (WSN). We show how one can develop discrete-event model libraries based on the DEVS formalism for mobile networks, allowing tracking the users' upload status in a given area of coverage, while using a non-cooperative algorithm or Coordinated Multipoint (CoMP) synchronisation. We show a DEVS model library for CoMP-based networks, showing how control messaging can be managed among Evolved Node Base stations (eNBs). We build a case study comparing two CoMP approaches: Joint Processing (JP) and Coordinated Scheduling (CS). Finally, we show how DEVS models can be used to model malware propagation in wireless sensor networks based on epidemic theory.

ARTICLE HISTORY
Received 11 March 2019
Accepted 27 March 2020

KEYWORDS
DEVS; Cell-DEVS; mobile networks; CoMP; WSN

1. Introduction

In recent years, communication networks have become ubiquitous, and their use has increased steadily. In particular, wireless networks such as mobile networks and wireless sensor networks have become extremely popular, as they have reduced cost and provide mobility. Wireless networks have attracted billions of users and it is one of the most valuable markets: a study shows that almost 98% of householders in the USA use wireless services, and nearly 49% of these use wireless services only (Cellular Telecommunications Industry Association [CTIA], Cellular Telecommunications Industry Association, 2016a, 2016b). Ericsson (2016) shows that the global mobile subscriptions reached 7.3 billion in 2015. Different organisations, companies, research groups, inventors are working together to introduce new services and techniques with better performance to attract a greater number of users into their territories.

All these efforts have one issue in common, and that is the need for being evaluated. In some cases, analytical evaluation has been used, but analytical methods are usually limited as they can solve simplified versions of complex networking problems and their underlying protocols. For instance, it is difficult to model phenomena such as channel propagation properties, node mobility, and radio characteristics using analytical methods. Nevertheless, having a set of models that could be used to study unfamiliar problems is useful for reasoning and expert validation. Therefore, many authors build models that can be later simulated for analysing the properties of the networks. Building and using model libraries would allow one to reuse the models but reusing models to extend and adopt them for new proposed solutions is also complex. In order to develop these models' libraries, one needs a flexible approach to model the target systems.

The use of formal simulation techniques in the field of discrete-event modelling has shown to be a suitable candidate for presenting complex systems such as mobile networks or wireless sensor networks (WSN) (Tavanpour et al., 2015, 2014). This problem-solving technique has a number of advantages for modelling and Simulation (M&S) of such complex systems.

Here, we will discuss the definition of an architecture, models, and libraries for modelling wireless networks based on the Discrete-EVent system Specification (DEVS) methodology (A. G. Wainer, 2009; Zeigler et al., 2000). The use of DEVS supplies various advantages. As DEVS specifies formally the network and protocols, we can reason on the models without worrying about the simulation artefacts (that could run in single processors or parallel computers without modifications). These discrete-events system specifications use a modular description for the models, which makes information hiding simple, and the quantitative complexity of the problems is attacked using a hierarchical approach, allowing the reuse of tested models, improving the safety of the simulations and allowing to reduce the development times. Consequently, we can build model libraries with components that can be easily reused, as shown in this article. The modularity and formal I/O port

CONTACT Misagh Tavanpour misagh_tavanpour@yahoo.com Department of Systems and Computer Engineering, Carleton University Centre for Visualization and Simulation (V-sim), Carleton University, Ottawa, ON, Canada

definitions of DEVS allow easy interaction and composition with a variety of environment models, tools, GIS software, data sets, and visualisation mechanisms, both locally and remotely. Another advantage is that the formal model can be checked formally, improving the error detection process and reducing testing time. Similarly, DEVS models are timed models, making the definition of timing properties in the protocols, network, and devices, easy to define, and the simulation software provides verification tools to improve the timing analysis mechanisms. Finally, integration with models defined with other modelling techniques is simpler, allowing one to mix, for instance, complex traffic or pedestrian models and combine them with the communication protocols discussed in this article (A. G. Wainer, 2009; Zeigler et al., 2000).

In (Wainer et al., 2013), we presented some generic ideas related to this research; here we focus on advanced models based on more complex protocols, and we focus on different methods and a new a library for modelling these advanced models. We discuss our proposed architecture and tools and introduce case studies for studying Coordinated Multi-Point (CoMP) and malware propagation in mobile networks and wireless sensor networks.

These case studies allow us to focus on some critical issues for future mobile networks. For instance, in mobile networks, supplying consistent high data rate service for users should be achieved regardless of their location in the coverage area. Providing such a service for the cell-edge users is complicated due to higher signal attenuation and interference at the edge. CoMP, a method introduced in the fourth Generation of mobile networks (4 G), was introduced to deal with this problem. CoMP considers a set of Base Stations that work together to reduce the interference and increase the signal-to-noise ratio. In the case of wireless sensor networks, the problems are also complex; for instance, energy efficiency and security are major concerns. We will focus on a case study of malware and its propagation in WSN, which could lead to the draining out of the energy resources of the sensor nodes, which eventually leads to network breakdown.

The rest of the paper organised as follows. In Section 2, we provide a brief overview on the wireless network concepts that we are going to model in the next sections. We also discuss some of the simulation tools and the related works. Moreover, we discuss the DEVS methodologies. In the following sections, we present different case studies in wireless networks using CD++, an open-source M&S platform for DEVS (A. G. Wainer, 2009). In Section 3, we present a model to track mobile users' upload process, while they are using either a non-cooperative algorithm or CoMP to upload one data file. In Section 4, we present a DEVS model for different CoMP architectures. We show how eNBs establish coordination sets and serve User Equipment (UE) jointly to improve cell-edge performance. In Section 5, we present another model on malware propagation in WSNs based on epidemic theory. This model shows the dynamics of malware propagation in wireless sensor networks as well as how the malware propagation effects the energy consumption of a sensor node.

2. Background

The use of mobile networks (also called cellular networks) is ever increasing, and the study in Ericsson, 2016 shows that the global mobile subscriptions will reach 9 billion by 2021. This massive number of users will generate more data traffic, and the total monthly mobile data traffic is expected to reach 52 EB in 2021. Service providers need to address these demands, which are derived from two sources: a large number of devices to provide service and their high data rate demands.

In recent years, various efforts have focused on improving the performance of mobile networks, focusing on these two problems. Providing high data rates to devices in all coverage areas is challenging, especially when a device is located near to a Cell Tower's border. This group of users has two problems. The first one is the long distance from the Cell Tower's centre, where their serving Base station (also called evolved Node B – eNB) is located. The second one is the higher interference from the neighbouring Cell Towers. Mobile network standards, such as LTE-A (LTE-Advanced), use different techniques to meet the expectations of the Cell-edge users. One of such methods, called Coordinated Multi Point (CoMP), considers a set of eNBs (called the *coordination set*) that work together to reduce interference and enhance the signal strength received (Tavanpour et al., 2015).

In recent years, not only cellular mobile networks have become popular; in recent years Wireless Sensor Networks have become more and more popular (and now there are important efforts deploying such networks combined with sensors in wired and wireless environments focusing on the Internet of Things). These networks have also various problems that need to be addressed. A WSN is a self-configuring network that consists of several sensor nodes distributed in the environment to sense the physical world. These small sensor nodes use radio signals to communicate with each other. The key advantage of WSN is their ability to bridge the gap between the physical and logical worlds, by gathering information from physical world and communicating that to powerful logical devices that can process it (He et al., 2004). Also, from the design prospective, we need to keep in mind that the sensor nodes have some constraints such as computing power, limited energy, bandwidth, communication range, etc. Overall, this kind of

wireless network also became popular, and nowadays, they are being used in a wide range of applications such as environmental monitoring, healthcare systems, smart homes, smart cities, military surveillance, industrial monitoring, traffic control, etc. As we can see, many of these networks are used in critical applications where threats to the security of the network cause serious incidents for lives or assets. In particular, malware software can disrupt the operation of such systems.

To address these issues innovative technologies, protocols, and algorithms are needed, and we need to evaluate these novel approaches properly before deploying them. Modelling and Simulation (M&S) is a good method to evaluate these methods as well as their interaction with the existing subsystems under a wide range of scenarios, as in most cases we cannot conduct extensive experimentation.

Robust M&S tools are thus important to deal with these issues, and in recent years, there have been various network simulation tools introduced. For instance, OPNET (Hammoodi et al., 2009; Koksal, 2008; Xian et al., 2008) is a high-level discrete-event simulator (built using object-oriented programming in C++) which provides a development environment for the users. The users can design and model different approaches in terms of protocols, devices, etc. OPNET supports a wide range of network types and technologies (i.e., optical, wireless, satellite, LAN, WAN), as well as various protocols. It provides a graphical user interface for easy development of the network model. Nevertheless, it is closed source, costly, and it is difficult to develop specific components.

Another popular discrete-event simulator for networks is called Network Simulator Version 2 (NS2) (Kamoltham et al., 2012; Rajankumar et al., 2014). NS2 libraries support the simulation of various network protocols over wired and wireless networks. NS2 is based on two languages: C++ and OTcl. The latter is an object-oriented extension of Tcl, and it is being used for the topology definition and controlling the simulation. The former is used to implement the core system including detailed protocols, algorithm implementation, and packet processing and byte manipulation. NS2 is open source simulator, but its structure is complex; adding a new component into the library or patching the extensions is difficult. In (Xian et al., 2008), the authors reported that NS2 requires a large amount of memory and time for debugging and tracing. Finally, NS2 source code implementation cannot be reused on a real system (Kamoltham et al., 2012). Therefore, the simulation code needs to be adapted to work on a real-world network. In other words, researchers need to implement their design twice: one in NS2 and the other one for the real system.

Network Simulator Version 3 (NS3) is an open-source simulator that provides an extensible network simulation platform. NS3 mostly concentrate on the Internet protocols, but it is not limited to that. Rather, many users employ NS3 for M&S of non-Internet-based systems (NS-3 Project, 2014). In contrast to NS2, by using NS3, researchers do not need to readapt their code for using it in a real system. In other words, the same code can be used for both simulation and emulation (Kamoltham et al., 2012). NS3 uses C++ for both the core system and controlling simulation, and it is not backward compatible with NS2 (Rajankumar et al., 2014).

These simulators have been used for many research projects related to our research. For instance, in (Meenakshi et al., 2014), the authors used OPNET to compare the performance of two routing protocols: RIP (Routing Information Protocol) and EIGRP (Extended Interior Gateway Routing Protocol) in Universal Mobile Telecommunication system (UMTS), a 3 G mobile technology. In (Rahman et al., 2014), the authors formulated a resource allocation problem and proposed an optimal algorithm. They run packet-level simulation by using OPNET to show the performance of their proposed method in the Long-Term Evolution (LTE) networks. In (Zhu et al., 2013), NS2 was used as a simulation tool to evaluate the performance of an energy-efficient routing protocol in WSN and compared it with other existing methods in this area. The authors used NS2 to measure the energy consumption of three protocols and the number of live nodes during the network lifetime. The authors in Qiu et al., 2009 described how one could build an LTE/SAE model in NS2. They measured throughput, average delay, and average jitter. In (Ghanem et al., 2012), the authors studied the ping-pong handover, one major problem in LTE mobile networks, and introduced a novel algorithm for decreasing its probability. They used NS2 to show the efficiency of their method. In (Guidolin et al., 2012), the authors used NS3 to provide a comprehensive evaluation of Multiple Input Multiple Output (MIMO) in cellular networks. To do so, they simulated a 2×2 MIMO system in LTE networks using NS3. In (Guidolin et al., 2014), the authors proposed a dynamic-distributed clustering algorithm for the LTE downlink. They employed NS3 to compare the performance of the proposed method with other clustering solutions in an LTE scenario.

The work presented here has been used for building a library for similar applications, which is based on the DEVS formalism (A. G. Wainer, 2009). DEVS theory is a formal modelling approach with a solid mathematical background, which supports a formal modelling both discrete and continuous systems. DEVS provides a formal framework for M&S of the system of the interest. Therefore, one can use DEVS as a precise methodology to define the models. DEVS provides a framework to build hierarchical models in a modular

fashion. Based on DEVS, the system of interest can be represented as a composition of atomic and coupled components. Atomic models are the basic blocks, which represent the *behaviour* of the system. Coupled models are used to define the *structure* of the system, such as interconnections among the models. These models can be reused, which can lead to reductions in development and testing time. In addition, different DEVS models can be integrated easily to produce more complex models (A. G. Wainer, 2009), and, in particular, they can be easily combined with other external models that could be useful for network simulations (for instance, weather models, pedestrian behaviour, or traffic in smart cities).

We used an extension to DEVS, called Cell-DEVS to model mobile networks. Cell-DEVS allows defining cellular models as discrete-event spatial components that are built as DEVS models. Cell-DEVS combines DEVS and Cellular models with explicit timing delay. It is used to capture the behaviour of the system of interest that can be represented as cell spaces. In Cell-DEVS, each cell is represented as an atomic model. In addition, a procedure is defined to connect cells to their neighbouring cells. Each cell uses a time delay mechanism that implements the required delay for each state change event (A. G. Wainer, 2009; Kazi & Wainer, 2017).

The formal definitions of DEVS and its advantages, which have been discussed in the literature, can be found in the Appendix (they have been included for completion; this research is well known and we suggest that appendix should be deleted in the final version of the paper, unless required by the Editor).

In this paper, we used CD++ (A. G. Wainer, 2009; Tavanpour et al., 2015), a framework for programming DEVS models. In CD++, a Model file is used for defining the DEVS coupled model hierarchical structure and coupling, and atomic models are defined in C++. An explanation of CD±± can be found in the Appendix (this also has been included for completion; this appendix could be cut in the definitive version of the paper if required by the Editor).

3. Cell-DEVS modelling of mobile networks

As mentioned, mobile networks provide radio support for their subscribers within the covered areas. Each area may be divided into a number of Cell Towers, and each Cell Tower has one fixed transceiver (from now on, we will use the term evolved Node B – eNB – to refer to this transceiver). Portable transceivers such as smartphones and tablets will be called User Equipment (UE). When a UE moves within the covered area, it connects to the network via the eNB of the host Cell Tower. This eNB is called the serving eNB of the UE, and the UEs use their serving eNB to have access to the mobile network services.

We mentioned that the LTE-A standard uses the CoMP method to enhance the Cell-edge users' performance. CoMP refers to a set of eNBs that coordinated their work to reduce the interference level at the Cell-edge users' side. This increases the quality of the received signal, and it leads to higher data rates for the Cell-edge users. Section 4 discusses CoMP concept and its advantages in more detail.

In this section, we show how to use Cell-DEVS to model a mobile network, which allows studying the spatial modelling of the phenomena, and the analysis of different options using simulation. In the following subsections, we first focus on model specification, where we describe how we use Cell-DEVS's features, including how to define rules applied to each cell. The model is used to track the UEs upload process, while they are using either a non-cooperative algorithm or CoMP to upload one data file. In CoMP networks, each UE interacts with all the eNBs in its coordination set. Our model will follow the standard specifications. We will also use a non-cooperative algorithm, in which each UE only communicates with its serving eNB, to compare both methods.

3.1. Cell-DEVS model specification

In Figure 1(b), the first two planes keep track of the position of the UEs within the network area for the non-cooperative *(ExistenceNo)* and CoMP methods *(ExistenceCo)*. A value of zero (white cells in the diagram) indicates a vacant cell, and a value of one (black cells) stands for that there is a UE in the cell. The number of UEs and their locations are selected randomly at the start of the simulation.

In Figure 1(b), the third plane (*Movement*) holds the direction of the next move for the UEs. At each step, a new random value is assigned to each of the cells in the third plane. This value is represented as an integer, where 2, 3, 4, 5, or 6 correspond to moving to the E, S, W, N, or not moving, respectively. Moreover, we use a different colour for each of these values to make UEs movements more visible in the figures. To do so, white, light grey, medium grey, dark grey, or black colours are corresponding to 2, 3, 4, 5, or 6 in order. If more than one UE want to go to move to the same target cell, the ones with higher priority complete their move, and the rest of the UEs with the same destination stay in their current location. The priority is chosen according to the direction of the UEs movement.

Figure 1: (A) a mobile network model with two Cell Towers, (B) Cell-DEVS model's planes

Figure 2 presents a fragment of the model definition that describes this model in CD++. The top model, named *uploadFile*, defines the dimensions of the cell grid, the type of delay functions used, the border definition (a wrapped model), as well as the neighbourhood scheme and the model's planes

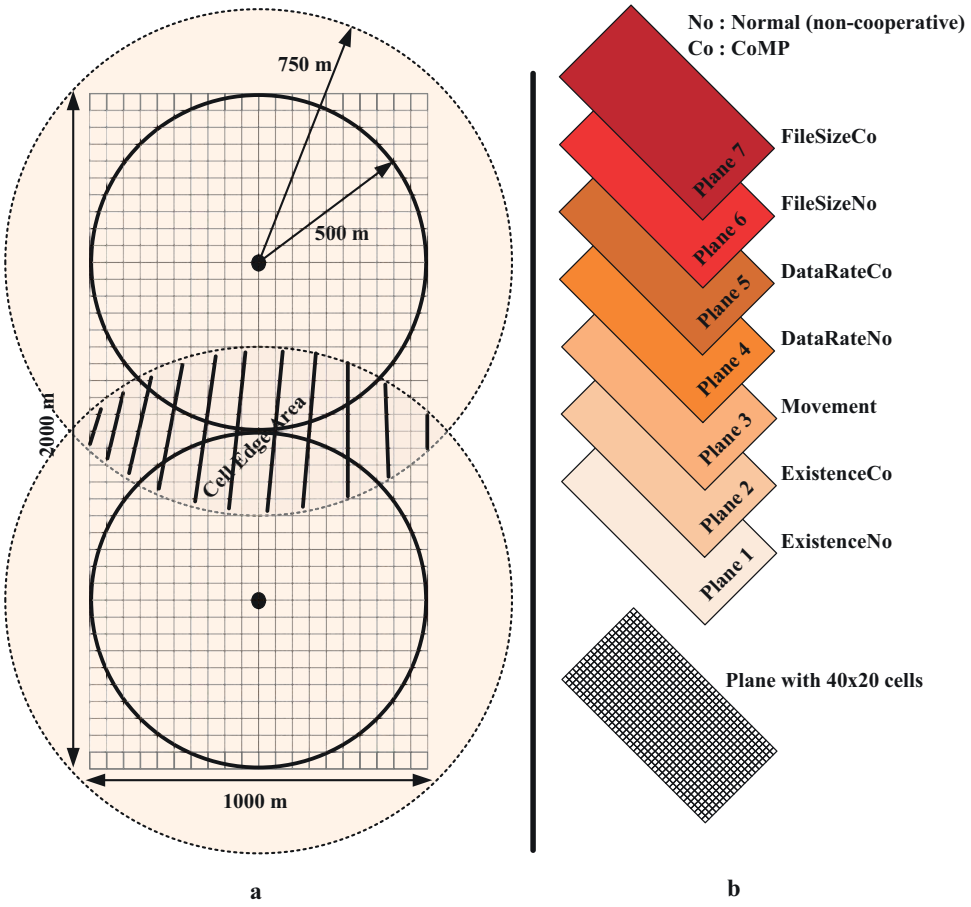

Figure 1. (A) shows a 2D representation of a mobile network area with two Cell Towers (*Notation: from now on, "Cell Tower" refers to a geographic area and "cell" refers to a single cell in a cellular model using Cell-DEVS*). Each Cell Tower has one serving eNB, which is located in its centre. The Cell Tower radius is 500 m; however, the eNBs signal can be received within a 750 m radius from the Cell Tower centre. Each Cell-DEVS cell represents an area of 50 m^2. Any cell in the model can be either vacant or occupied by a UE. The UEs are free to move randomly within an area of 1×2 km^2. In this model, we show a mechanism for defining a complex model and its spatial properties: we use a 3D Cell-DEVS model, in which we represent a different aspect of the model on different 2D plane interconnected. The Cell-DEVS representation allows defining different aspects in the spatial model in different planes, which are interconnected through the Cell-DEVS coupled model specification, and the neighbourhood relationship. This example will show how to address this kind of phenomena using the Cell-DEVS methodology.

```
[uploadFile]
type : cell                dim : (40,20,7)        delay : transport
defaultDelayTime : 100     border : wrapped   ...
initialCellsValue : Coverage_v1.val        %input file
localtransition :   ExistenceNo            %{ (0,0,0)..(39,19,0) }
zone : ExistenceCo { (0,0,1)..(39,19,1) }
zone : Movement    { (0,0,2)..(39,19,2) }
zone : DataRateNo  { (0,0,3)..(39,19,3) }
zone : DataRateCo  { (0,0,4)..(39,19,4) }
zone : FileSizeNo  { (0,0,5)..(39,19,5) }
zone : FileSizeCo  { (0,0,6)..(39,19,6) }
neighbors:...
...
```

Figure 2. Coupled model definition in CD++.

discussed earlier. The model is "wrapped", which means that the model is converted into a hypercube.

Figure 3 represents the initial state of the cells according to the input file that we have defined in Figure 2. Here, the different colours of the cells at each plane represent different values. For example, in the last four planes, a cell in black has a positive value.

The UEs in the cells of the first two planes check their corresponding cell in the third plane to decide the direction of their next move (from now on, we call *corresponding cells* to two cells in different planes with same row and column; for instance, (x, y, z_1) and(x, y, z_2)). For example, in Figure 4, the UE in (0,11,0) checks the cell (0,11,2) to decide the direction of its next move. Although it is

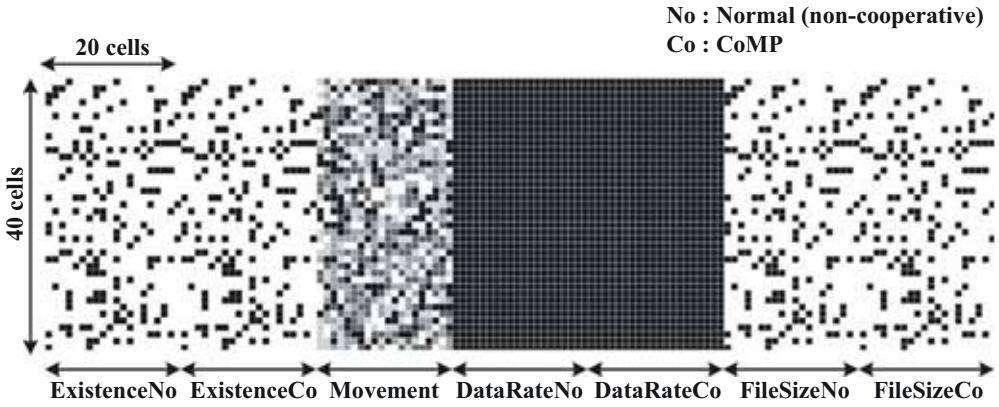

Figure 3. Initial status of the cell of the seven planes using a specific input file.

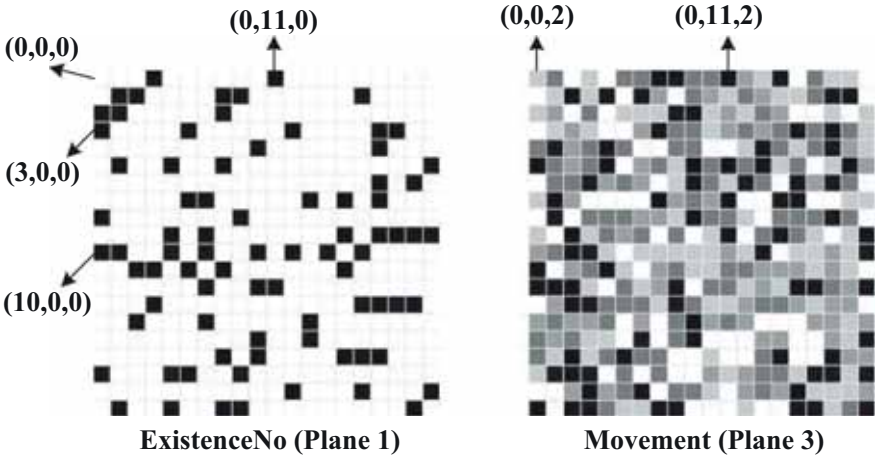

Figure 4. Planes 1 and 3 during the simulation.

not depicted in this figure, the UE in (0,11,1) checks the same cell in the movement plane for its next move. In this example, the UEs in (0,11,0) and (0,11,1) would stay in the same cells during the next interval according to the corresponding cell value in the movement plane. Different mobility models can be easily implemented and modified (Farooq et al., 2007; A. G. Wainer, 2007, 2006).

In Figure 1(b), the cells in the fourth plane (*DataRateNo*) represent the upload data rate of the UEs in their corresponding cell in the first plane (*ExistenceNo*). The value of these cells is fixed during the simulation time. The available data rate for the UEs at each cell is calculated as Equation 1 (The 3rd Generation Partnership Project [3GPP], The 3rd Generation Partnership Project, 2017):

$$\text{data rate} = B \log_2\left(1 + \frac{R_{pwr}}{N_0 \times B}\right) \quad (1)$$

Where B and N_0 are the transmission bandwidth and the noise variance; R_{pwr} is the signal power received, measured as in Equation 2 (where T_{pwr} is the transmitted signal power, T_{Gain} is the transmitter antenna gain, R_{Gain} is the receiver antenna gain and MCL is the minimum coupling loss) (The 3rd Generation Partnership Project, 2017).

$$R_{pwr} = T_{pwr} - \text{Max}(\text{pathloss} - T_{Gain} - R_{Gain}, \text{MCL}) \quad (2)$$

To calculate R_{pwr} we need to know the value of the path loss. Equation 3 shows that this parameter is measured based on the log-normally distributed shadowing with a standard deviation of 10 dB (LogF) and macrocell propagation model for the urban area (L_{urban}) (The 3rd Generation Partnership Project, 2017).

$$\text{pathloss} = L_{urban} + \text{LogF} \quad (3)$$

The macroCell propagation model for an urban area, which is denoted as L_{urban}, is calculated using Equation 4 (The 3rd Generation Partnership Project, 2017):

$$L_{urban} = (40*(1 - (4*10^{-3}*\text{Dhb}))*\log_{10}R) - (18*\log_{10}\text{Dhb}) + (21*\log_{10}f) + 80\text{dB} \quad (4)$$

Where *R* is the eNB-UE separation in kilometres, *f* is the carrier frequency in *MHz* and *Dhb* is the base station antenna height in m. In this article, we use an urban area setting for mobile networks. Table 1

Table 1. Urban area setting.

Parameter	Value
Frequency	2000 MHz
eNB Antenna Gain	15 dB
Transmission bandwidth	5 MHz
Noise Density	−174 dBm/Hz
MCL	70 dB
eNB Antenna Height above rooftop (Dhb)	15 m
Standard deviation	10 dB
File size	[10MB, 640MB]
Maximum eNB power	43
Maximum power per DL traffic channel	30 dBm
Minimum eNB power per user	15 dBm
eNB Noise figure	5 dB
Maximum UE power	21 dBm
Minimum UE power	−50 dBm
UE Noise figure	9 dB

displays the required parameters and their value (The 3rd Generation Partnership Project, 2017).

In Figure 1(b), the cells in the fifth plane (*DataRateCo*) are initialised similarly to the previous plane cells. These cells represent the upload data rate of the UEs in their corresponding cell in the *ExistenceCo* plane. Those UEs were supposed to use CoMP. Therefore, during the upload process the cell-edge UEs (in the second plane) experience reduced interference and higher data rate compared to the UEs in the first plane (Tavanpour et al., 2015, 2014). The CoMP gain for the cell-edge users is up to 10–20%.

In Figure 1(b), the size of data files of the UEs is saved in the last two planes. At the beginning of the simulation, the corresponding cells in these two planes have the same exact values. During the simulation, at each step, we reduce this number according to the data rate at the cell that the UE has occupied. In addition, the updated data file size is moved to the corresponding cell of the new location of the UE. For example, if the UE in the first or second plane moves to the N cell, the updated value of the corresponding cell in the sixth or seventh plane is shifted to the N cell as well.

3.2. Cell-DEVS model plane's rules

We need to define the required rules for the cells of each plane. For the cells of the first two planes, we have three main situations: when a UE leaves a cell or it finishes its upload, when a UE stays in the same cell and when a UE enters a cell. We only discuss the required rules for the first plane. The rules for the second plane are similar (with minor modifications).

The rules above show the first case. The first line checks if there is a UE in the cell, and it has uploaded its data file. To check the upload status of the data file, a UE in the first plane should check its corresponding cell in the sixth plane in order. If the value in that corresponding cell is equal or less than zero, it means that the UE has uploaded the data file completely. Therefore, we change the value of the host cell to zero. The next rules are used for the cells in the first plane that have a UE, and that UE has not finished the data file upload yet. The first rule in this set tells us that the UE wants to go the cell to the east in the next step and that the destination cell is empty. Therefore, the UE can move to that cell, and the value of the current host cell is set to zero. In the next rule, we check if a UE wants to move to the south. We also need to check the priority of the movement. To do so, besides the S direction and the empty destination cell, the host cell also needs to check that nobody from the west side of the destination cell wants to go to the destination cell (south vs. east). We need to define more rules to cover possible UEs movement to other directions as well.

With the previous set of rules, we covered the UEs movement to a new destination. If none of those rules applies to a cell, it may mean that the UE must stay in the same cell for the next iteration (second case). Therefore, regardless of the value of the Movement plane, we need to check if the UE remains active for the next iteration, and if that is the case, we need to set the value of the cell for the next iteration. Otherwise, we assign 0 to the cell. The following rules show the second case.

```
rule: 1 100 {(0,0,0) = 1 and (0,0,5) ! = 0 and ((0,0,5) - (0,0,3)) > 0}
rule: 0 100 {(0,0,0) = 1 and (0,0,5) ! = 0 and ((0,0,5) - (0,0,3)) ≤ 0}
```

The next rules are used for cells without an active UE. These cells check the neighbour cells to see if there is any active UE that wants to move to them. If they find such a UE, then they also need to check the upload status of this UE to make sure that it is not going to be finished at the end of the current cycle. This process must be done according to the moving priority of the UEs. Therefore, these cells check the west neighbour first, then the north neighbour (if it is required), etc. The following rules show this situation.

```
[ExistenceNo]
rule: 0 100 {(0,0,0) = 1 and (0,0,5) ≤ 0}
rule: 0 100 {(0,0,0) = 1 and (0,0,5) > 0 and (0,0,2) = 2 and (0,1,0) = 0}
rule: 0 100 {(0,0,0) = 1 and (0,0,5) > 0 and (0,0,2) = 3 and (1,0,0) = 0 and
      ((1,-1,0) = 0 or (1,-1,2) ! = 2)} ...
```

```
rule: 1 100 {(0,0,0) = 0 and (0,-1,0) = 1 and (0,-1,2) = 2 and ((0,-1,5)-(0,-1,3))>0}
rule: 1 100 {(0,0,0) = 0 and (-1,0,0) = 1 and (-1,0,2) = 3 and ((-1,0,5)-(-1,0,3))>0}
```

As mentioned earlier, the third plane is used to represent the direction of the UEs' movement in the first two planes. The cells' value can be a random number between 2 and 6. At each step, we assign a new random value to each cell of the third plane. However, we can assign the new values only to the cells that the value of their corresponding cell in one of the first two planes is one and assign zero to the rest of the cells. The fourth and fifth planes (*DataRateNo, DataRateCo*) contain fixed parameters that are not changing during the simulation.

The sixth and seventh planes keep track of the data file size of the UEs that use non-cooperative algorithm and CoMP These two planes are coordinated with their corresponding cells in the first two planes. If a UE in one of the cells of the first plane wants to move to another cell (destination cell), the data file size in the sixth plane should be updated according to this movement. This means that the corresponding cell of the destination cell in the sixth plane should have the updated data file size, and the corresponding cell of the current host cell in the sixth plane should be set to zero. Figure 5 shows an example of this situation. In this figure, in the iteration N, there is one active UE in the Existence plane. This UE wants to go the north in the next cycle (according to the Movement plane). Therefore, in the DataFileSize plane, the updated data file size (150–15) should move to the corresponding cell as well.

The following rules are applied for the cells of the sixth plane. The first set is applied to the cells that the UE in their corresponding cell in the first plane moves to a neighbour cell. The second set is applied to the cells that their corresponding cell in the first plane is going to be the host of a UE. As always, the order of the rules is important, and we define the rules according to the UEs movement priority.

```
rule: 0 100 {(0,0,0) > 0 and (0,0,-5) = 1 and (0,0,-3) = 2 and (0,1,-5) = 0}
rule: 0 100 {(0,0,0) > 0 and (0,0,-5) = 1 and (0,0,-3) = 3 and (1,0,-5) = 0
      and ((1,-1,-5) = 0 or (1,-1,-3) ! = 2)}
...
rule: {(0,-1,0) - (0,-1,-2)} 100 {(0,0,0) = 0 and (0,0,-5) = 0 and ((0,-1,-5) = 1
      and (0,-1,-3) = 2) and ((0,-1,0) - (0,-1,-2)) > 0}% input from west
rule: {(-1,0,0) - (-1,0,-2)} 100 {(0,0,0) = 0 and (0,0,-5) = 0 and ((-1,0,-5) = 1
      and (-1,0,-3) = 3) and ((-1,0,0) - (-1,0,-2)) > 0}% input from north
```

3.3. Cell-DEVS model behaviour and results

We will now show to compare the upload time of the UEs while they are using a non-cooperative algorithm and CoMP. The model can be used to study the performance of a network, the location for the eNBs to provide reduced cost and better coverage. We will discuss two of the simulation scenarios we studied. In both, the geographical area is covered by two Cell Towers (as in Figure 1). We have one eNB per each Cell Tower, located at the Cell Tower centre. We used a 40 × 20 model to represent both Cell Towers (a 20 × 20 sub model to represent each Cell Tower). Each cell represents 50 m^2 area. The remaining parameters used are presented in Table 1.

The number of the UEs, initial location, movement during the simulation time and the size of the data file that they want to upload are similar for the both non-cooperative and CoMP We assumed that at the beginning of the simulation, 20% of the cells have an active UE (a UE that wants to upload a data file). These cells are shown with black colour. The UEs are randomly distributed among the cells (by using a uniform distribution), and each cell has at most one active UE at a time. Figure 6 shows the initial setup of this simulation scenario for the first two planes. During the simulation, the UEs travel among the cells, and at each step, we reduce from their data file size according to the available data rate at their host cell. As mentioned in Figure 1, the cell-edge area is located between rows 15 and 25. The UEs that travel between rows 15 and 25 can use either the non-cooperative algorithm or CoMP to upload their data files. The UEs can only use the non-cooperative algorithm as their upload method when they travel in the non-cell edge areas (between rows 0 and 14, and between rows 26 and 39).

The simulation results show that for both the methods, the UEs in the non-cell-edge areas have the same results. However, in the cell-edge areas, the UEs that use the CoMP method show better performance. For example, in Figure 7, which shows the first two planes status in the 25th iteration, if we focus on the common cell-edge area between the two Cell Towers (row 15 to 25), the first plane has more number of the UEs that has not finished their upload yet. In the left side of this figure (plane one), the UEs were able to use only the non-cooperative algorithm as their upload method in the covered area. On contrary, in the right side of the figure (plane two), the UEs were also able to use CoMP if they were in the cell-edge areas. As seen in Figure 7, in the 25th iteration of the simulation, the upload process of a less number of UEs is unfinished when they were able to use CoMP as their upload technique compared to the situation in which the UEs were only able to use the non-cooperative algorithm. This

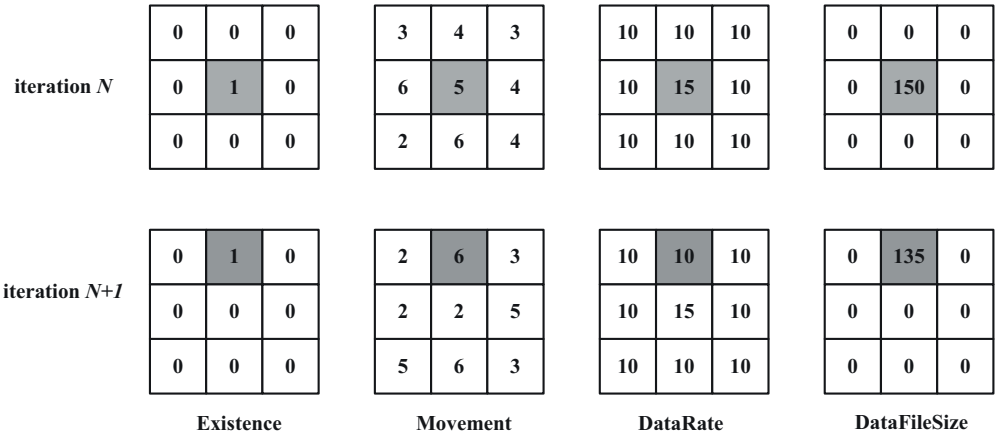

Figure 5. Update the data file size.

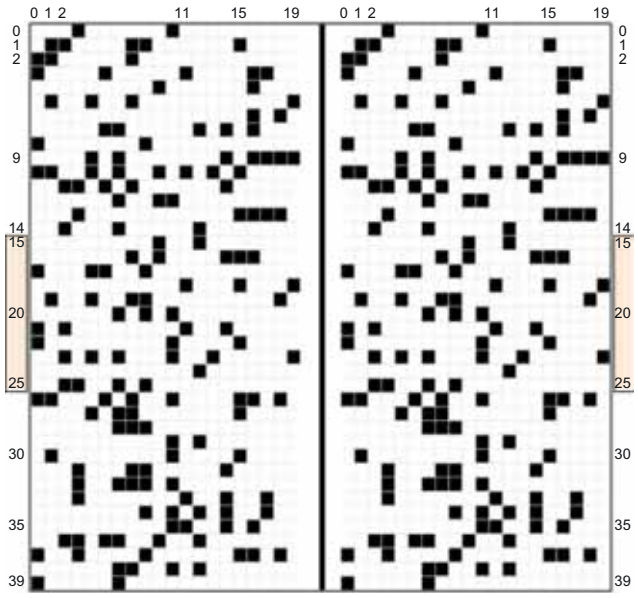

Figure 6. First two planes initial status; simulation scenario one.

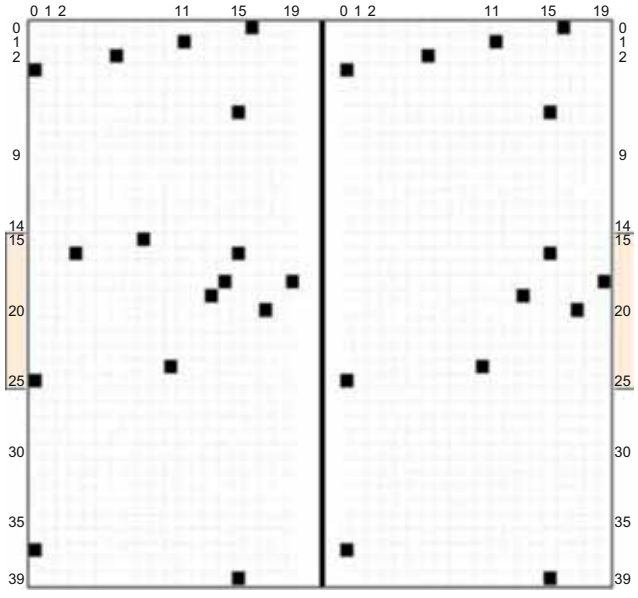

Figure 7. First two planes status in the 25th iteration of the simulation scenario 1.

explains why we see a smaller number of black cells in the cell-edge area of plane two.

The following results present another scenario in which we focus on the behaviour of the UEs when they are initially distributed on the cell-edge area between the two Cell Towers. In this scenario, we randomly distributed 45 active UEs among the cells between row 15 and row 25. Again, each cell has at most one active UE at a time.

Figure 8 shows the initial status of the first two planes. As in the previous scenario, at each step, a new random number is generated for the UEs movement. Therefore, the UEs may travel in any direction. Figure 9 shows the first two planes in the19th iteration. As there are less UEs that remained in the right side of Figure 9, we can see that the UEs in plane 2 that used CoMP had better performance, and more of them finish their upload faster than the UEs in plane 1. This scenario shows the effect of CoMP on the upload performance of the UEs in the cell-edge area.

One important point about the Cell-DEVS model is that we can easily scale it up to cover a bigger area or a greater number of UEs. We also can add more planes to design a more complicated model. For example, to scale it up to cover a bigger area, we only need to increase the number of the rows and columns in the variable "dim" in Figure 2 as well as adjusting the "zone" variables. Since, same rules are applied to the cells; we do not need to add more rules. However, if we want to add more planes to have a more complicated model, we need to define the rules for the new plane accordingly to demonstrate its role in our model.

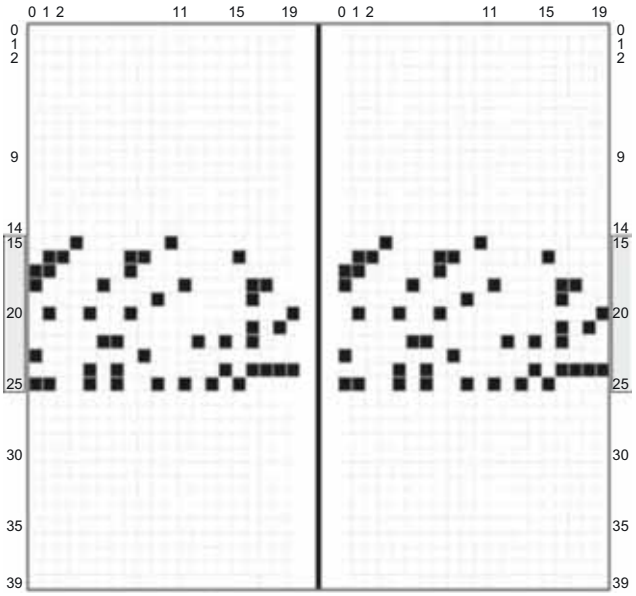

Figure 8. First two planes initial status in the simulation scenario 2.

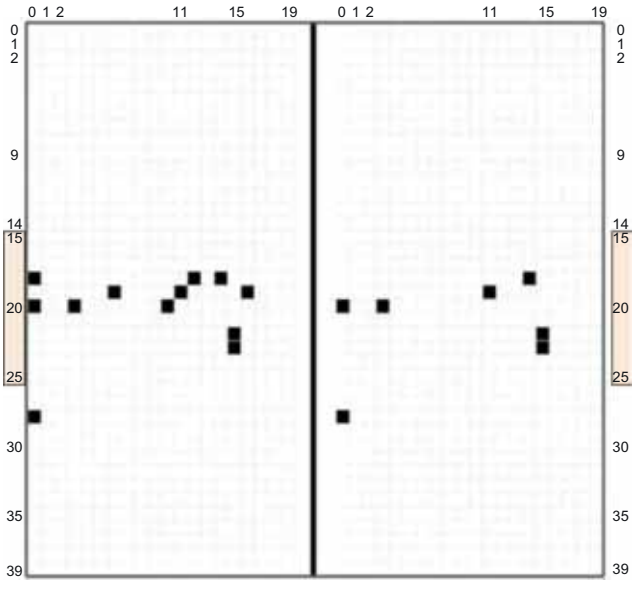

Figure 9. First two planes status in the 19th iteration of the simulation scenario 2.

In this section, we focused on the Cell-DEVS model for the mobile network, where we studied the effect of CoMP technique on the users' upload performance. In the next section, we focus on a DEVS model for the two CoMP approaches, while our focus is on how timing challenges can be addressed by a DEVS model.

4. DEVS modelling of mobile networks

As discussed earlier, UEs located near the Cell Tower's borders suffer from two transmission barriers: the distance between the UE and its serving eNB, and the higher Inter-Cell Interference (ICI) that the UE receives from the neighbouring Cell Towers. Long-Term Evolution-Advanced (LTE-A) was introduced as a mobile communication standard in the 4 G systems and beyond to solve these problems.

As discussed in the previous sections, LTE-A can use CoMP (Tavanpour et al., 2015, 2014) to enhance the performance of the users regardless of their location within the coverage area. In CoMP, a coordination set refers to a set of eNBs that are coordinated jointly and dynamically. The objective of such coordination set is to manage and mitigate interference, enhancing the performance of UEs, especially for those close to the Cell Tower edge. CoMP can be used in both the uplink and downlink processes. To support this feature in LTE-A networks, the eNBs and UEs require the exchange of scheduling decisions, hybrid ARQ feedback, channel state information (CSI), and other control information. The eNBs share the received messages from their UEs with other eNBs in the coordination set using a standard interface denoted as X2.

As seen in Figure 10, CoMP methods can use two different techniques: Joint Processing (JP) and Coordinated Scheduling or Beamforming (CS/CB). In JP two or more coordinating eNBs simultaneously transmit or receive data from the UEs, and this simultaneous data transmission increases the power of the received signal, increasing data rates. In CS/CB, the eNBs coordinate and schedule resources to UEs dynamically, in such a way that minimises the effects of ICI (Tavanpour et al., 2015, 2014). In this method, only one of the eNBs transmits a signal to the UE and the other eNBs of the coordination set try to reduce the interference level.

Such coordination between eNBs is complicated, especially when the network provides coverage for a large area with a large number of users, and therefore, verifying and validating original approaches on live networks is complicated. In the rest of this section, we first focus on the DEVS model hierarchy and model definition. Then, we elaborate the behavioural of the components of the model. Finally, the complete model behaviour and results are presented.

4.1. DEVS model hierarchy definition

The right side of Figure 11 shows a simplified DEVS model hierarchy for a mobile network. The top model is called Area, which includes several Cell Tower coupled models as its sub models. Each Cell Tower has one eNB coupled model, and a few UE coupled models. Both the UE and eNB coupled models have two atomic models as their sub models (processor and queue).

In the mobile network, the eNBs are connected through X2 interfaces. In our DEVS model, we need to connect each pair of eNBs together. To do so, we connect one of the output ports of the eNBProcessor of the first eNB to one of the input ports of the eNBQueue of the second eNB and vice versa. In addition, a UE may travel among the Cell Towers of a mobile network, and it communicates with the eNB of those Cell Towers. Therefore, there should be separate connections between each pair of the UEs and the eNBs-coupled models. To support such a communication, we need two connections between each pair of UEs and eNBs coupled models. One of them will be used to connect one of the output ports of a UEProcessor to an input port of an eNBQueue, and the other will be used to connect one of the output ports of the eNBProcessor to an input port of the

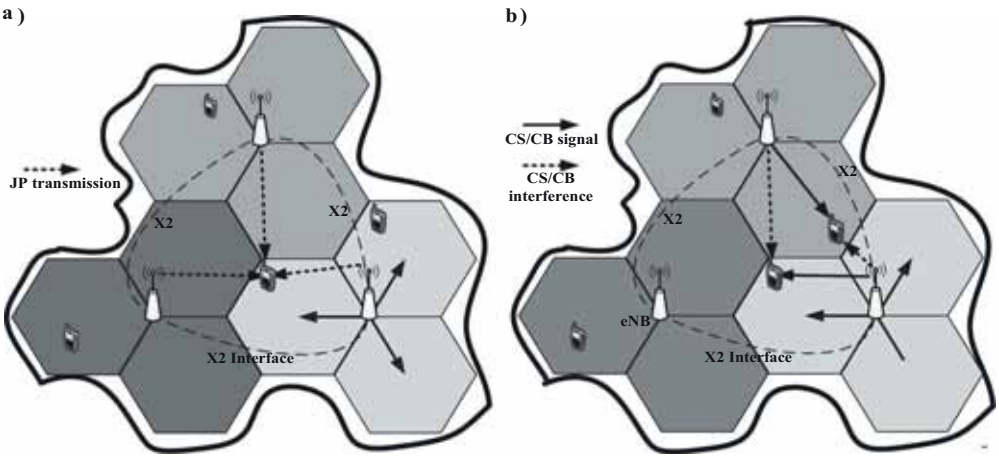

Figure 10. (A) JP transmission (B) Coordinated scheduling/beamforming in LTE-A.

```
components: Cell1  Cell2  ...
Link : Out1@Cell1   In1@Cell2
Link : Out1@Cell2   In1@Cell1
Link : Out2@Cell1   In2@Cell2
...
[Cell1]
components: eNB1  UE1 …
in : In1    in : In2  …
Out : Out1  Out : Out2 …
Link : In1@Cell1  In1@eNB1
Link : In2@Cell1  In1@UE1
...
Link : Out1@eNB1  Out1@Cell1
Link : Out1@UE1    Out2@Cell1
…
[eNB1]
Components:
eNBP1C1@eNBProcessor eNBQ1C1@eNBQueue
in : In1
in : In2
Out: Out1
Out: Out2 …
Link : Request@eNBP1C1  Re-
quest@eNBQ1C1
Link : In1@eNB1   In1@eNBQ1C1
Link : In2@eNB1   In2@eNBQ1C1
…
Link : Out1@eNBP1C1   Out1@eNB
Link : Out2@eNBP1C1   Out2@eNB
…
[eNBP1C1]
// parameters for the atomic model …
[Cell2]
…
```

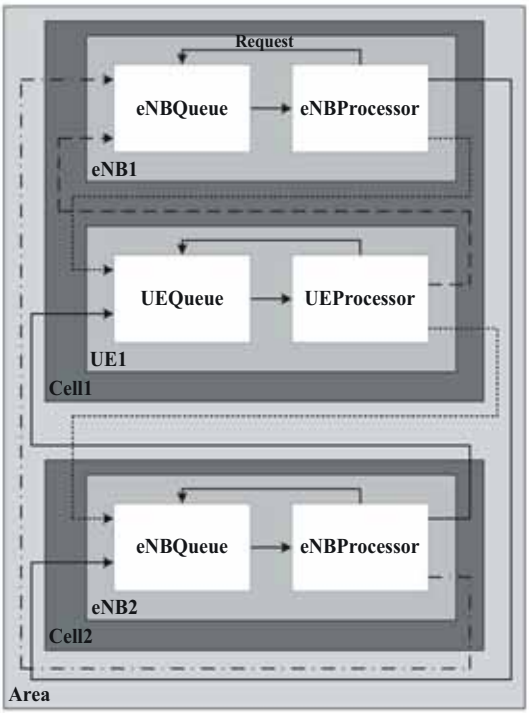

Figure 11. Simplified DEVS model hierarchy and model definition for a mobile network.

UEQueue. Moreover, if we focus on a UE and an eNB-coupled model in Figure 11, there is a connection ("Request") between their atomic models (between the processor and the queue atomic models). The processor atomic model uses this link to send out a message to the queue atomic model to say it is ready for the next task. In addition, another connection connects the only output port of the queue atomic model to the only input port of the processor atomic model.

The left side of Figure 11 shows the Model file that defines the structure of the DEVS model on the right side of this figure by using CD++. The definition of the hierarchy of the model is started by defining the Area as the top model. To do so, the components of the Area coupled model as well as the interconnections among these components are defined. As the next step, the internal structure of the components of the Area coupled model is defined one by one. This process is continued until we reach to a complete definition of our DEVS model structure.

4.2. DEVS model components' behaviour

The queue atomic model has three states: namely, Idle, Push, and Pop. The external transition function receives messages from the input ports and initiates appropriate state transitions. Furthermore, the internal transition function defines state changes according to the current state, and the time advance function controls the required timing configuration during the simulation.

Two atomic models present the behavioural of the processor of eNBs and UEs: eNBProcessor and UEProcessor in order. In this model, the UEs communicate with each other through eNBs. Therefore, UEs can send/receive data or control information to/from their serving eNB. The control packets include the base information for eNBs. The eNBs use this information to determine the UE's status. For example, according to this information, a serving eNB can determine if a certain UE will be scheduled to operate in normal mode or CoMP mode and if it gets scheduled to be in CoMP mode, which eNBs are in the coordination set. In such a case, the serving eNB must send data and control information to the other eNBs in the CoMP session. In addition, according to these control packets, an eNB can determine that a UE wants to move from one Cell Tower to another Cell Tower or even that a UE wants to leave the Area. In such a scenario, eNBs should handover both data and control packets to other eNBs. In case of data packets,

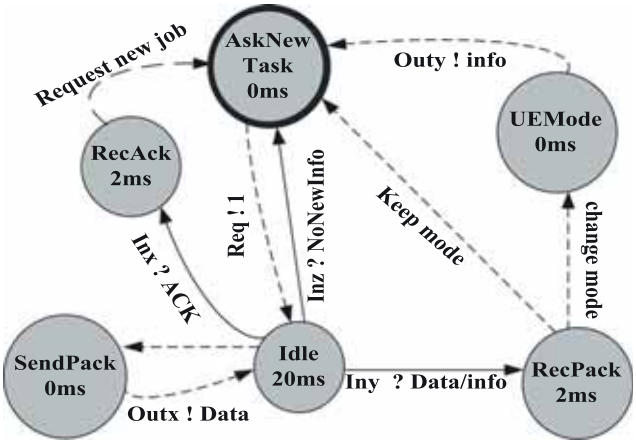

Figure 12. UE processor DEVS graph.

the source UE sends its data packet to its serving eNB, and this eNB uses the destination address to forward the packet. This packet may be handover among multiple eNBs to reach to the serving eNB of the destination UE, and then the packet can be delivered to its destination. We use atomic models to implement the operation of the eNBs and UEs processor. To do so, the external transition function receives messages from the input ports and initiates appropriate state transitions. Moreover, the internal transition function defines state changes according to the current state. In addition, the output function sends out the data/control packets over the output ports. Again, it worth to mention that the eNBs communicate over X2 interfaces, and a pair of eNB and UE communicate over radio signals. The X2 interface characteristic can have a huge effect on the overall network performance.

Now, the UEPRocessor and UEQueue atomic models can be discussed in more detail. The eNBProcessor and eNBQueue are similar. A UEQueue keeps the arrival messages to the UE coupled model. Upon receiving a request message (via "Request" port), it forwards one message (the one that has arrived sooner than the others have) to the UEProcessor. The UEProcessor is responsible to process the received messages and to perform the required actions based on the arrival messages. A simplified behaviour of a UE is implemented in an atomic model by using different states that each of these states represents a specific status of a UE. As seen in Figure 12, UEProcessor is defined using six states. The initial state is AskNewTask. There is only one transition from this state. When UEProcessor sends a request message (which, in the coupled model defined above, will be sent to the corresponding UEQueue asking for a new task), it changes its state to "Idle". In the "Idle" state, one of the following three transitions can occur. If the UE wants to generate an output packet (an acknowledgement – ACK – or a data packet), it switches to "SendPack". If a UE receives an external message carrying a data packet, there is a transition from "Idle" to "RecPack". Finally, if a UE receives an external message representing an acknowledgement packet (for a data packet sent earlier), UEProcessor changes from "Idle" to "RecAck". In the "RecAck" state, once the UEProcessor finishes the acknowledgement of the packet, it changes its state to the "AskNewTask" state. In the "RecPack" state, if the UEProcessor wants to continue operating in the current mode a state transition to the "AskNewTask" state occurs. Otherwise, it switches to "UEMode" state and it changes the UE receiving mode (that has been described in the previous paragraph) and from there it changes the state to the "AskNewTask" state. In the "SendPack" state, UEProcessor forms the output (data or ACK) packet. After that, the UE sends out the output packet and it goes back to the "Idle" state.

4.3. DEVS model behaviour and results

To implement our DEVS model, we start by defining the atomic models and testing their functionality. For example, in the case of eNBs, we build the eNB coupled model after testing the eNBProcessor and eNBQueue atomic models. To do so, we defined a Model file to define the eNB coupled model, input/output ports, its components, and the interconnection among these components. After that, we can evaluate the functionality of an eNB coupled model. We do the same for the rest of the coupled models until we build all the required models.

As mentioned earlier, a UE in a normal mode only communicates with other entities through its serving eNB. In such a scenario, the serving eNB of the source UE forwards the received packets to the serving eNB of the destination eNB through the LTE-A network. Finally, the destination UE receives the message from its serving eNB. Figure 13 illustrates a sample of such communication between UE0 and UE2. For this example, eNB0 and eNB4 are the serving eNBs of

UE0 and UE2 in order. UE0 produced data packets and UE2 has created an ACK packet for each of received data packets from eNB4.

The serving eNB of a cell-edge UE can determine if the UE is a candidate for working in CoMP mode. According to this scenario, the serving eNB can create a coordination set for that UE to improve its performance. To do so, the serving eNB should communicate with other eNBs. If the UE chose the JP method, the serving eNB should share UE's data among the eNBs of the coordination set, and after that, all the eNBs of the coordination set forwards the data to the UE simultaneously. Figure 14 shows such scheduling among the eNBs of the coordination set. As seen in this figure, the serving eNB sends out a copy of the data to non-serving eNBs of the coordination set. After that, all three eNBs sends the same copy of the data to the UE at the same time.

However, if the UE selects the CS/CB method, then the serving eNB sends the interference cancellation message to the other eNBs of the coordination set. The CoMP scheduling continues while a UE remains in a shared area, which is covered by the eNBs of the CoMP coordination set, so that the UE can benefit from high data rate and reduced interference. As seen in Figure 15, the UE communicates with its serving eNB, and it exchanges some information about the network status. The serving eNB realises that the UE can be a candidate to use the CoMP technique as the upload method. Therefore, it sends a control message to the non-serving eNBs of the UE coordination set to ask them to support the UE upload by reducing the interference with that specific UE. We assumed that the eNB processors and the X2 interfaces that connect the eNBs are fast. This means that the delay in eNB-to-eNB communication is negligible.

Again, like the Cell-DEVS model, scaling up the simulation is not a complicated task for our DEVS model. Once we define our components, we can get any number of instances that we require from our simulation. We only need to take care of the connections among the components.

In this section, we showed how DEVS can be used to define a model hierarchy. In addition, we discussed how it can easily be used to implement the timing property of a DEVS model's components. In the next section, we study Cell-DEVS application for the implementation of a WSN model.

5. Cell-DEVS model for wireless sensor networks

Wireless Sensor Networks are collections of nodes that can sense a variety of physical phenomena, partially process the raw data locally and communicate wirelessly. Each node is equipped with processing units like microcontrollers, CPUs, different types of memory, an RF transceiver, a power source (e.g., batteries or solar cells), and diverse sensors and actuators (Stankovic, 2008). One of the core features of WSN is their ability to collect information from the real world and communicating that information to more powerful logical devices that can process it and use accordingly (He et al., 2004). In recent years, WSNs have received tremendous attention in the research community, with applications in health care, transportation, battlefields, energy management, industrial process monitoring, home automation, environmental monitoring, etc (Farooq & Kunz, 2011; Wainer et al., 2013).

As the applications of WSN increasing gradually, sensor nodes become more vulnerable to many security attacks like malware. Although security has been widely studied and many techniques have been developed for classical wired networks, many of them are not well suited in WSNs. The WSNs are highly distributed,

```
Time         Port  Value
...
                           //UE0 Processor out
00:00:00:020  0  020001... //UE0 send out a message to its serving eNB (s-eNB)
00:00:00:052  0  020002...
...
                           //eNB0 Processor out
00:00:00:022  4  020001... //eNB0 forwards the received message to another eNB
00:00:00:030  7  201001...
00:00:00:054  4  020002...
00:00:00:064  7  201002...
...
                           //eNB4 Processor out
00:00:00:024  9  020001... //s-eNB of destination UE forwards received message
00:00:00:028  0  201001...
00:00:00:056  9  020002...
00:00:00:060  0  201002...
...
                           //UE2 Processor Out
00:00:00:026  4  201001... //destination UE sends back ACK for received message
00:00:00:058  4  201002...
```

Figure 13. UE communication in normal mode.

```
Time         Port Value
                              //Serving eNB
00:00:00:178  6  201004...    //To the non-serving eNB
00:00:00:178  1  201004...    //To the non-serving eNB
00:00:00:180  7  201004...    //To UE
…
                              //non-serving eNB of the coordination set
00:00:00:180  9  201004...    //To UE
…
                              //non-serving eNB of the coordination set
00:00:00:180  9  201004 ...   //To UE
...
```

Figure 14. eNB to UE message in JP method.

self-organised, resource constrained, broadcast nature of its transmission medium and unattended environment, which makes it more valuable in malware attacks. A malware residing in an active node can infect its active neighbours, which can directly communicate with this node. Several actions or states within a node, for example, computation activity in the Microcontroller Unit (MCU), data transmission, data reception, idling, and sleeping, consume the energy of a node. Among these, the most energy-consuming activity is data transmission (El-Shabani et al., 2013). Therefore, when malware is transmitted from one node to another node repeatedly, the energy of the nodes is exhausted, and more nodes become dead.

In (Nekovee, 2007), the authors developed a new model for epidemic spreading of worms. They investigated the spreading of worms in Wi-Fi-based wireless ad-hoc networks by using Monte Carlo simulations. The authors of (Wang et al., 2010) proposed a model for analysing the dynamics of worm propagation in wireless sensor networks based on epidemic theory. In (Song & Jiang, 2008), the authors analysed the process of malware propagation in WSN using cellular automata and validated the model. Our model is based on this work. In this section, we show an implementation of such a model defined using Cell-DEVS and presented the results that successfully satisfy the previous results. In the results section, we presented how the simulation results agree with the results in (song & Jiang, 2008). This open library can be used by researchers in the field of networks on how to transform similar models into the Cell-DEVS versions. This model is composed of maximum N stationary and identical sensors. The sensors are randomly deployed on a rectangular two-dimension grid composed of L × L cells. Each cell is occupied by one or no sensor node. Sensor nodes can establish wireless links with each other within a circle of radius r. In this model, a sensor node is considered into one of the following four states (Song & Jiang, 2008; Wainer et al., 2013):

- Susceptible (*S*): The nodes in state *S* have not been infected by worm yet but are vulnerable to become infected.
- Infected (*I*): The nodes in *I* have been infected by malware in the WSN and may spread the malware to its neighbours.
- Recovered (*R*): The nodes in *R* used to be infected by malware and recovered from the infected malware.
- Dead (*D*): Sensors have restrained power that decreased during wireless communication, so a node can run out of energy.

Initially, all the nodes in the lattice are considered *Susceptible*, with fixed power. Though the energy consumption of a node depends on several actions, message transmission consumed the maximum energy of a

```
Time         Port Value
                              //UE Output
00:00:00:430  0   113610...   //To the Serving eNB
…
                              //Serving eNB Input
00:00:00:432 In   113610
…
                              //Serving eNB Output
00:00:00:438  1   114010...   //To the non-serving eNB
00:00:00:438  6   114060...   //To the non-serving eNB
…
                              //The non-serving eNB of the coordination set Input
00:00:00:438  0   114010...   // Message from the serving eNB
…
                              //The non-serving eNB of the coordination set Input
00:00:00:438  0   114060...   // Message from the serving eNB
…
```

Figure 15. Control messaging among the eNBs in CS/CB method.

sensor node and the other factors fade in comparison. Therefore, in the *Susceptible* state, a node consumes energy at the low rate. A node consumes energy at the highest rate in the *Infected* state because it broadcasts the malware packets to other nodes (Wainer et al., 2013). A node completely drained out of power is moved to the *Dead* state. Cells in the grid without any sensor node are considered in *Dead* state. Figure 16 shows the transition of states of a node based on the different probabilities (Song & Jiang, 2008). In the transition diagram α, β and γ are the probabilities of the transition of states. The transition probability from susceptible state to infected state is β and from infected sate to recovered state is α. Moreover, a node can move to the dead state at a probability of γ. The details of the probability parameters will be discussed later.

In this section, we present a model of malware propagation in WSN using Cell-DEVS. The entire covered area of the network is considered as cell space and the coverage area of a node is considered as a cell. We built a Cell-DEVS model consisting of a 3D lattice (20*20*2 cells). We considered two planes: the first one describes the different deployed sensor nodes and the transition of the nodes. The second plane represents the energy status of the corresponding nodes in the first plane throughout the simulation. In this model, Moore's neighbourhood is adopted. Therefore, there are nine neighbouring cells with the origin cell.

- The states of a node discussed earlier are denoted in Cell-DEVS as below: Let S_t be a variable that denotes the states of a node at a specific time and $S_t = \{0, 1, 2, 3, 4\}$ where, the node or the cell is in *Susceptible* state within the coverage area of the network or within the cell space.
- 1: The node is in *Dead* state within the coverage area of the network or within the cell space.
- 2: The node is in *Recovered* state within the coverage area of the network or within the cell space.
- 3&4. The node is in *Infected* state within the coverage area of the network or within the cell space.

Each plane uses its own rules. Infected nodes in the sensor network try to diffuse the malware at each time step to their neighbours. All the nodes in the *Susceptible* state become infected with probability β when it received a packet containing a copy of the malware. Otherwise, the node remains in the *Susceptible* state. From the *Infected* state, a node can go to *Recovered* state by running a patch with a probability α. Sensor nodes transit to *Dead* state from any other state with a rate γ based on the restrained power of the sensors and the consumptions during communication or packet transmission among the nodes. A cell of the grid that does not contain any node also considered as dead node.

To observe the malware propagation, two types of information (state transition and energy reduction) play a role in our model. The following sample rules are used to decide how a sensor node changes its state from *Susceptible* to *Infected* state. Here the transition probability *ß* is 30% (*ß* = 0.3).

```
rule: 3 100 { (0,0,0) = 0 and stateCount(3) ≥ 1 and (random ≤ 0.3) }
rule: 4 100 { (0,0,0) = 0 and stateCount(4) ≥ 1 and (random ≤ 0.3) }
```

A node consumes energy in any state, but the highest consumption is considered in *Infected* state. The rules below show how the energy of an infected cell (node) reduces to 80% of the original.

```
rule: {(0,0,0)*.8} 100 {(0,0,0)>1 and (0,0,-1) = 3}
rule: {(0,0,0)*.8} 100 {(0,0,0)>1 and (0,0,-1) = 4}
```

We will discuss a few examples on the execution of this model showing how the malware propagates into the network as well as how the propagation of malware affects the energy consumption of a sensor node. In our first scenario, we executed malware propagation for a 20 × 20 network with radius r = 1.5. Initially, all the sensor nodes are considered with full of power and we assume that one sensor is infected with malware. The transition probability from *Susceptible* state to *Infected* state is 30% (β = 0.3). The state of a cell changes from *Infected* to *Recovered* with 1% probability (α = 0.01) if its energy level is greater than 1 (at the beginning of the simulation the energy level of a node is 100). Second plane represents the energy status of the corresponding node in every time step. If the state of the cell (sensor node) is *Infected*, then its energy is reduced to 80% of the original in the next simulation step. Likewise, the energy of the sensors in the *Susceptible* and *Recovery* states decreases to 99% and 95% of the previous value, respectively.

The simulation results shown in Figure 17 describe the malware propagation using transition probability of 30% from *Susceptible* state to *Infected* state. Figure 17(a) shows one infected node by a dark grey cell as initial state. In the next simulation step, the energy of the infected sensor decreases to 80. Similarly, the energy of the other sensors in the *Susceptible* state decreases to 99. The malware spreads and infects

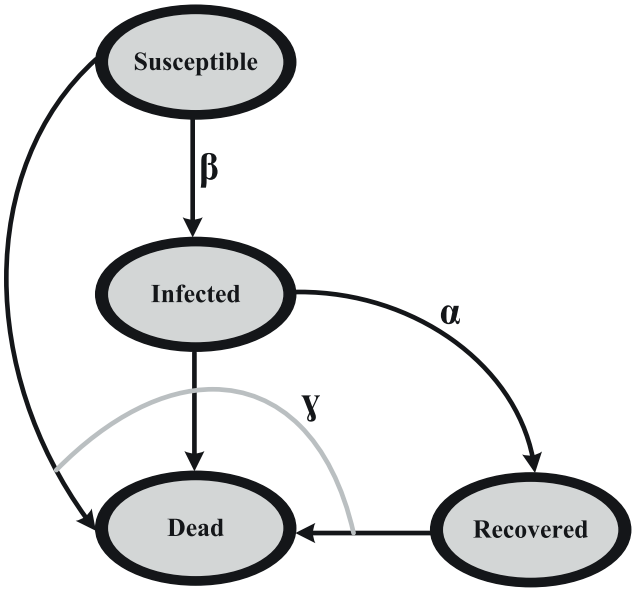

Figure 16. State transition of sensor nodes.

more sensors as it can be seen in Figure 17(b). The malware spreads further and infects more sensors gradually. Later, few of the infected sensors turn to the *Recovered* state, as it can be seen in Figure 17(d) (white cell). Note that the energy of the recovered sensors is reduced to 95 in each simulation step.

Figure 18 shows the simulation results of malware propagation using transition probability of $\beta = 0.6$ from the *Susceptible* state to *Infected* state. In this simulation scenario, we maintain other assumptions the same as the scenario shown in Figure 17 to observe how the model behaves with the change of transition probability. As we have seen in Figures 17 and 18, the malware propagation speed under the transition probability $\beta = 0.6$ is higher compared to the case in which the transition probability was 0.3, which is expected. Figures 17 and 18 also shows that the propagation of malware happens along with the defined neighbours which is spatially bounded. Both properties of the malware spread to satisfy the result in Song & Jiang, 2008.

Figure 19 shows how the nodes gradually move towards the *Dead* state by consuming the residual energy in every simulation step. In this case, we considered the energy of an infected node reduces to 80% of the original but the nodes in *Susceptible* and *Recovery* state reduce to 99% and 95% of previous energy. As we can see in Figure 19(a), after simulation advances with the propagation of malware, the first infected sensor becomes dead after running out of energy (black cell). It also proves that if an infected node does not get patch to move to the recovery state, it could go to the dead state earlier, because of the higher cumulative consumption of the residual energy. The remaining parts of Figure 19(b,c) show the progression of the dead cells with the simulation advances. This simulation results also agree with the results shown in (Song & Jiang, 2008).

Therefore, the simulation results presented above show that the spread of malware depends on the value of the parameter β as well as the state of the nodes. Furthermore, the lifetime of the sensor network depends on the energy depletion rate in each state of the nodes.

6. Conclusion

We discussed the use of the DEVS and Cell-DEVS formalisms for M&S of networking applications. In this paper, three applications were presented, and CD++ has been used as the platform for M&S of these applications. In the first application, we used cell-DEVS to model a mobile network to study users' upload process while they are using either non-cooperative algorithm or CoMP. The simulation results revealed that the cell-edge users showed better performance while they were using CoMP compared to the times that they used the other algorithm. In the second application, we used DEVS to model a mobile network that uses the CoMP method. We have represented different mobile network components using atomic and coupled models. We ran some tests to validate the modelling. In the last application, we presented a Cell-DEVS model that investigates the behaviour of malware in a wireless sensor network. Different simulations were performed, and results were presented graphically. This model showed how fast a malware could spread throughout the network. It also showed how a sensor node could run out of

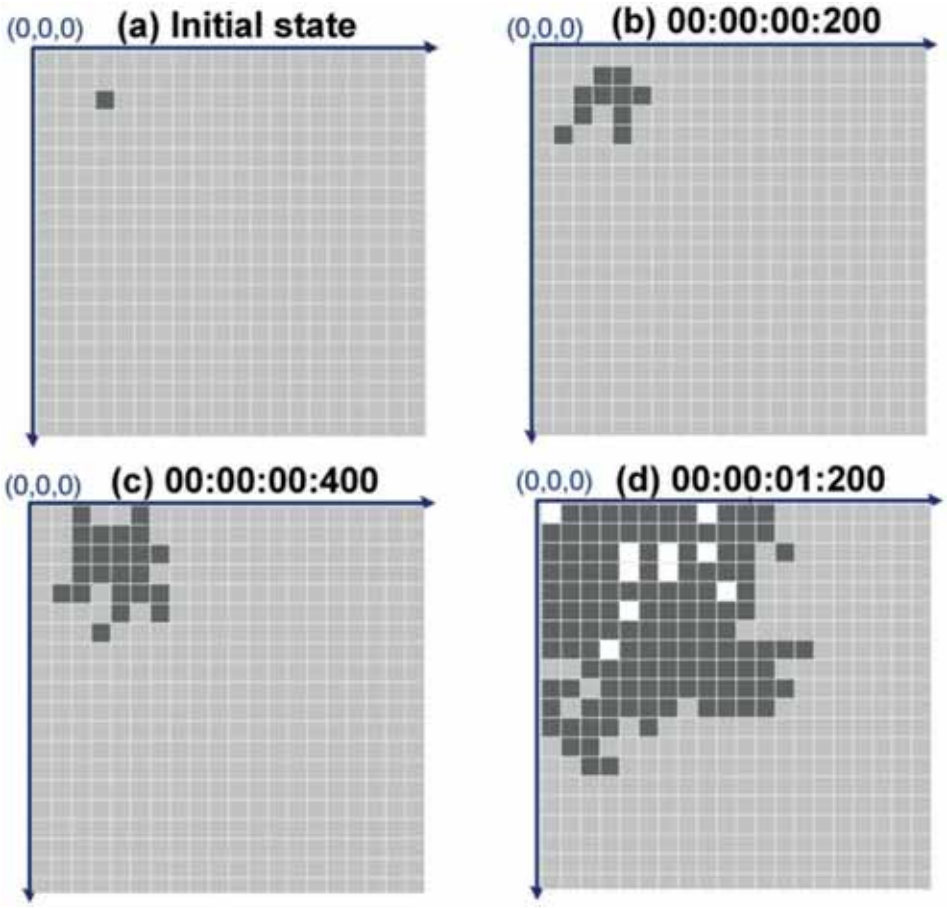

Figure 17. The effect of infected cell with β = 0.3 on its neighbours with time steps.

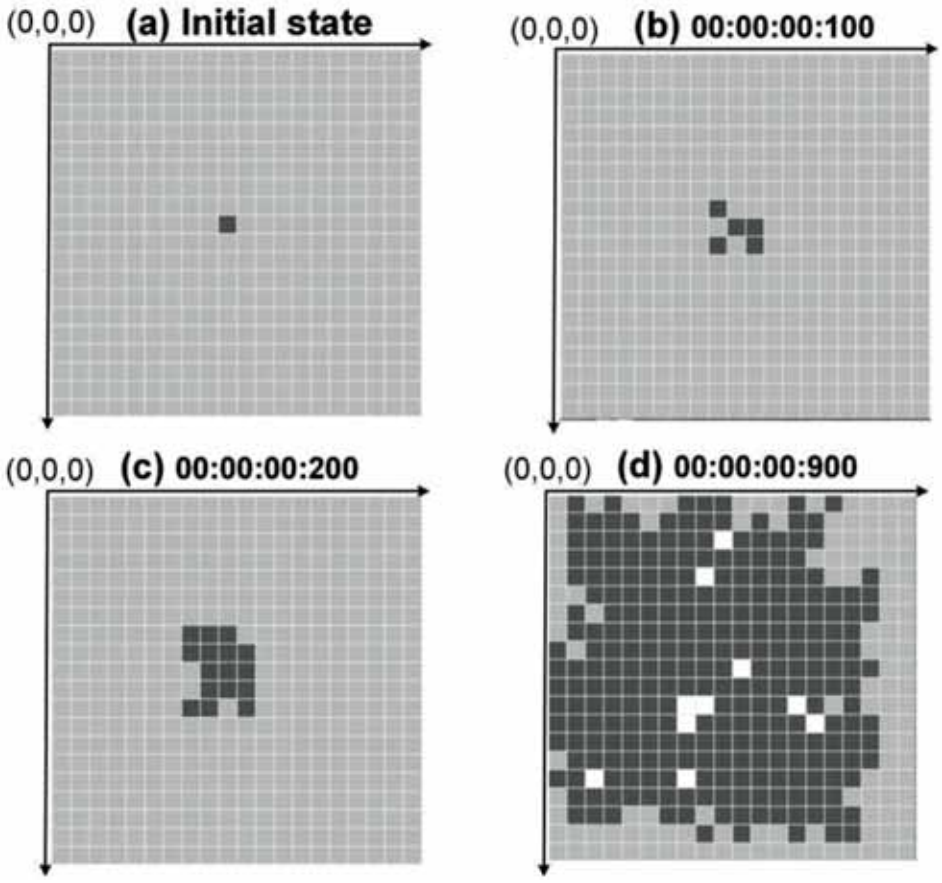

Figure 18. The effect of infected cell with β = 0.6 on its neighbours with time steps.

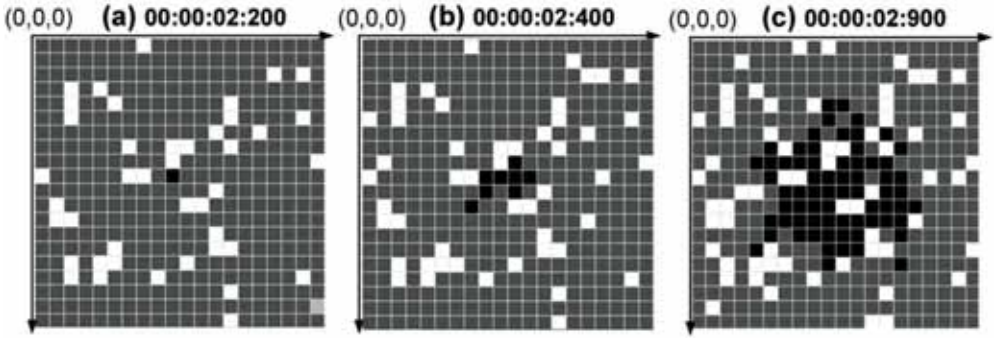

Figure 19. Dead cells progression.

energy because of malware propagation in the network. As more nodes became dead, the WSN became disabled due to the characteristics of the WSN.

These models showed how DEVS and Cell-DEVS formalisms could be used to model these kinds of problems in the wireless networking area. We could reduce the complexity of the model by applying some level of simplification, and we implemented the models using CD++ platform. However, since networking applications and problems are evolved fast, we need to design models that are more complex to be able to study new advanced methods in these areas. The modular and hierarchal nature of DEVS and Cell-DEVS gives us this opportunity to build the new features on the top of the current DEVS and Cell-DEVS models.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

The 3rd Generation Partnership Project. (2017, March). *3GPP specification series: TR36.942*. http://www.3gpp.org/DynaReport/36-series.htm.

Cellular Telecommunications Industry Association. (2016a, June). *CTIA annual wireless industry*. http://www.ctia.org/industry-data/ctia-annual-wireless-industry-survey.

Cellular Telecommunications Industry Association. (2016b). *Wireless quick facts*. http://www.ctia.org/industry-data/wireless-quick-facts.

El-Shabani, M., Moallemi, M., & Wainer, A. G. (2013). Cellular simulation of asymmetric energy requirements in wireless sensor networks. SCSC. *Proceedings of the 2013 Summer Computer Simulation Conference*. Toronto, ON.

Ericsson. (2016, June). *Ericsson mobility report*. https://www.ericsson.com/assets/local/mobility-report/documents/2016/ericsson-mobility-report-june-2016.pdf.

Farooq, M. O., & Kunz, T. (2011). Operating systems for wireless sensor networks: A survey. *Sensors*, *11*(6), 5900–5930. https://doi.org/10.3390/s110605900

Farooq, U., Wainer, A. G., & Balya, B. (2007). DEVS modeling of mobile wireless ad hoc networks. *Simulation*, *15* (3), 285–314.

Ghanem, K., Alradwan, H., Motermawy, A., & Ahmad, A. (2012, July). Reducing ping-pong handover effects in intra EUTRA networks. CSNDSP. *Proceedings of the 8th International Symposium on Communication Systems, Networks & Digital Signal Processing*. Poznan, Poland: IEEE.

Guidolin, F., Badia, L., & Zorzi, M. (2012, September). Implementation of 2 × 2 MIMO in an LTE module for the ns3 simulator. CAMAD. *Proceedings of the 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks* (pp. 281–285). Barcelona, Spain: IEEE.

Guidolin, F., Badia, L., & Zorzi, M. (2014). A distributed clustering algorithm for coordinated multipoint in LTE networks. *IEEE Wireless Communications Letters*, *3*(5), 517–520. https://doi.org/10.1109/LWC.2014.2340405

Hammoodi, I. S., Stewart, B. G., Kocian, A., & McMeekin, S. G. (2009, September). A comprehensive performance study of OPNET modeler for ZigBee wireless sensor networks. NGMAST'09. *Proceedings of the 3rd International Conference on Next Generation Mobile Applications, Services and Technologies* (pp. 357–362). Cardiff, Wales: IEEE.

He, T., Krishnamurthy, S., Stankovic, J. A., Abdelzaher, T., Luo, L., Stoleru, R., Yan, T., Gu, L., Hui, J., & Krogh, B. (2004, June). Energy-efficient surveillance system using wireless sensor networks. *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services* (pp. 270). Boston, MA.

Kamoltham, N., Nakorn, K. N., & Rojviboonchai, K. (2012, January). From NS-2 to NS-3 - implementation and evaluation. *Proceedings of the Computing, Communications and Applications Conference* (pp. 35–40). Hong Kong: IEEE.

Kazi, B. U., & Wainer, A. G. (2017). Integrated cellular framework for modeling ecosystems: Theory and applications. *Simulation*, *94*(3), 213–233. https://doi.org/10.1177/0037549717706007

Koksal, M. M. (2008, October 22). A survey of network simulators supporting wireless networks. Unpublished Manuscript, Department of Computer Science, Graduate School of Natural and Applied Sciences, Middle East Technical University.

Meenakshi, Kaushik, A., & Satvika. (2014, July). Performance comparison of routing protocols using OPNET UMTS model suite. ICSPCT 2014. *International Conference on Signal Propagation and Computer Technology* (pp. 668–672). Ajmer, India IEEE.

Nekovee, M. (2007). Worm epidemics in wireless ad hoc networks. *New Journal of Physics*, *9*(6), 189–202. https://doi.org/10.1088/1367-2630/9/6/189

NS-3 Project. (2014, December). *NS3: A discrete-event network simulator*. https://www.nsnam.org/docs/release/3.21/tutorial/html/introduction.html#.

Qiu, Q., Chen, J., Ping, L., Zhang, Q., & Pan, X. (2009, December). LTE/SAE model and its implementation in NS 2. *Proceedings of the 5ht International Conference on Mobile Ad-hoc and Sensor Networks* (pp. 299–303). Fujian, China: IEEE.

Rahman, M. M., Hsu, C., Hasib, A., & Hefeeda, M. (2014, June). Hybrid multicast-unicast streaming over mobile networks. *Proceedings of IFIP Networking Conference*. Trondheim, Norway: IEEE.

Rajankumar, P., Nimisha, P., & Kamboj, P. (2014, March). A comparative study and simulation of AODV MANET routing protocol in NS2 & NS3. *Proceedings of the International Conference on Computing for Sustainable Global Development* (pp. 889–894). New Delhi, India: IEEE.

Song, Y., & Jiang, G. P. (2008, June). Modeling malware propagation in wireless sensor networks using cellular automata. *Proceedings of the International Conference on Neural Networks and Signal Processing* (pp. 623–627). Nanjing, China: IEEE.

Stankovic, J. A. (2008, October). Wireless sensor networks. *IEEE Computer*, *41*(10), 92–95. https://doi.org/10.1109/MC.2008.441

Tavanpour, M., Mikhail, J., Moallemi, M., Wainer, A. G., Boudreau, G., & Casselman, C. (2015). Data upload in LTE-A mobile networks by using shared segmented upload. *Journal of Network*. *10*(4), 252–264.

Tavanpour, M., Moallemi, M., Wainer, A. G., Mikhail, J., Boudreau, G., & Casselman, C. (2014, July). Shared segmented upload in mobile networks using coordinated multipoint. SPECTS2014. *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems* (pp. 662–669). Monterey, CA: IEEE.

Wainer, A. G. (2006). ATLAS: A language to specify traffic models using cell-DEVS. *Simulation Modelling Practice and Theory*, *14*(3), 313–337. https://doi.org/10.1016/j.simpat.2005.07.002

Wainer, A. G. (2007). Developing a software toolkit for urban traffic modeling. *Software Practice and Experience*, *37*(13), 1377–1404. https://doi.org/10.1002/spe.809

Wainer, A. G. (2009). *Discrete event modeling and simulation: A practitioner's approach*. CRC Press, Taylor & Francis Group.

Wainer, G. A., Tavanpour, M., & Broutin, E. (2013, December). Application of the DEVS and Cell-DEVS formalisms for modeling networking applications. WSC. *Proceedings of 2013 Winter Simulation Conference* (pp. 2923–2934). Washington, D.C: IEEE.

Wang, X., Li, Q., & Li, Y. (2010). EiSIRS: A formal model to analyze the dynamics of worm propagation in wireless sensor networks. *Journal of Combinatorial Optimization*, *20*(1), 47–62. https://doi.org/10.1007/s10878-008-9190-9

Xian, X., Shi, W., & Huang, H. (2008, June). Comparison of OMNET++ and other simulator for WSN simulation. *Proceeding of the 3rd IEEE Conference on Industrial Electronics and Applications* (pp. 1439–1443). Singapore, Singapore: IEEE.

Zeigler, B. P., Praeofer, H., & Kim, T. G. (2000). *Theory of modeling and simulation: Integrating discrete event and continuous complex dynamic systems*. Academic press.

Zhu, K., Dou, Q., Shao, A., Chu, P., Xiao, Y., & Peng, Y. (2013, May). An energy-efficient routing protocol based on hotspot-aware uneven clustering and dynamic path selection. *Proceedings of the 22nd Wireless and Optical*

Communication Conference (pp. 315–320). Chongqing, China: IEEE.

APPENDIX I – FORMAL DEFINITIONS OF DEVS AND CELL-DEVS

Equation 5 formally specifies a DEVS atomic model:

$$M = \langle X, Y, S, \delta_{int}, \delta_{ext}, \lambda, t_a \rangle \tag{5}$$

Where $X = \{(p, v) \mid p \epsilon IPorts,\ v \epsilon X_p\}$ is the set of input events, where *IPorts* reveals the set of input ports, and X_p shows the set of values for the input ports. $Y = \{(p, v) \mid p \epsilon OPorts,\ v \epsilon Y_p\}$ is the set of output events, where *OPorts* reveals the set of output ports, and Y_p shows the set of values for the Output ports. S is the set of sequential states. $\delta_{int} : S \rightarrow S$ is the internal state transition function. $\delta_{ext} : Q \times X \rightarrow S$is the set of external transition function where $Q = \{(s, e) \mid s \epsilon S,\ 0 \leq e \leq ta(s)\}$ and e is the elapsed time since the last transition function. $\lambda = S \rightarrow Y$ is the output function and $ta : S \rightarrow R_0^+ \cup \infty$ is the time advance function (A. G. Wainer, 2009).

According to this definition at any given time, a DEVS model is in a statesϵS and if there is not any external event, it remains in that state for a lifetime defined by ta(s). When the state lifetime expires (e = ta(s)) the model sends the output λ(s)(if there is any output) through the output ports. After that, it performs an internal transition function to select the new state by δ_{int}(s).Based on the definition, a state transition can also happen due to the arrival of an external event. In this case, the external transition function determines the new state, given by δ_{ext}(s, e, x) where s is the current state, e is the elapsed time since the last transition and xϵX is the external event that has been received. The time advance function ta(s) can take any real value from the defined interval in the definition. It worth to mention that if ta(s) = ∞, the state is called passive which means that the atomic model stays in this state until receiving an external event. This situation can be used as a termination condition as well. In addition, a state with ta(s) = 0 is called a transient state and it leads to an instantaneous internal transition. Equation 6 formally specifies a DEVS coupled model (A. G. Wainer, 2009):

$$CM = < X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC, Select > \tag{6}$$

Where D is the set of components name. M_d is a DEVS model. *EIC* is the set of external input couplings. *EOC* is the set of

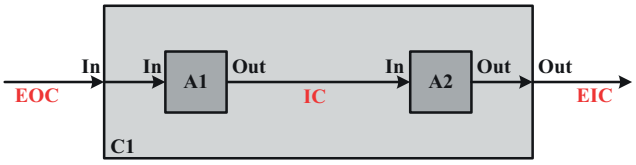

Figure 20. A DEVS-coupled model.

external output couplings. *IC* is the set of internal couplings. *Select* is the tiebreaker function in case of simultaneous internal event among the components of the coupled model (A. G. Wainer, 2009). Figure 20 shows a DEVS model with two levels, one coupled model (C1) and two atomic models (A1 and A2).

Cell-DEVS is a combination of DEVS and Cellular Automata (CA) with explicit timing delay. It is used to capture the behaviour of the system of interest that can be represented as cell spaces. In Cell-DEVS, each cell is represented as an atomic model. In addition, a procedure is defined to connect cells to their neighbouring cells. Each cell has a delay mechanism that implements the required delay for each state change event. Equation 7 formally specifies a Cell-DEVS atomic model (A. G. Wainer, 2009):

$$TDC = \langle X, Y, S, N, Type, d, \tau, \delta_{int}, \delta_{ext}, \lambda, ta \rangle \tag{7}$$

Where N shows the set of input values. d is the delay for the cell. *Type* is the kind of delay (inertial/transport/other). $\tau : N \rightarrow S$ is the local computing function. A Coupled Cell-DEVS model consists of several Cell-DEVS atomic models in a two-dimensional or three-dimensional cell space. The borders of the cell space can be either wrapped or non-wrapped. In the latter, the border cells must have special rules defined by the modeller, and in the former, the border cells at the border from one side of the cell space are considered neighbours to the cells at the border on the opposite side of the cell-space. Equation 8 formally specifies a coupled Cell-DEVS atomic model (A. G. Wainer, 2009):

$$GCTD = \langle X, Y, Xlist, Ylist, \mu, N, \{m, n\}, C, B, Z, select \rangle \tag{8}$$

Where *Xlist* is the input-coupling list. *Ylist* is the output-coupling list. μ is the neighbourhood size. N is the neighbourhood set. $\{m, n\} \in N$ is the size of the cell. C is the cell-space set. B is the set of border cells and Z is the translation function.

These formal approaches enable to prove the correctness and completeness of the simulation models. Errors found during the simulation can be fixed by analysing the specification, without considering the underlying software system. DEVS and Cell-DEVS are discrete-event formalisms, providing higher precision and speedups than the discrete-time approaches. DEVS allows the modeller to formally specify discrete-event systems using modular descriptions. This strategy allows the reuse of tested models, improving the safety of the simulations and allowing reducing the development times. As it uses a continuous-time base, precision of the models can be improved while CPU time requirements can be reduced. Higher timing precision can be obtained without using small discrete time segments (that would increase the number of simulation cycles). The

formalism is based on sound theoretical grounds, allowing for an abstract design of models that are independent from the implementation platform. The use of DEVS and Cell-DEVS provides (Farooq et al., 2007):

- A method that allows conducting formal tests.
- Seamless model sharing between different DEVS-based toolkits.
- High-performance execution of the same models in a parallel simulation environment
- Remote execution using client–server services, allowing remote interaction between users.
- The ability to execute the models on a distributed platform.
- The possibility to define models using different techniques interacting within the same environment. This could allow including non-network entities that affect network operation, providing results that are more realistic, including mobility and traffic models.
- The potential to automatically deploy models that have been evaluated on the simulation environment into the actual networking hardware, converting them into the real applications.

APPENDIX II – OVERVIEW OF CD++

As mentioned earlier, the CD++ toolkit provides a framework for programming DEVS models. A Model file is used for defining the DEVS coupled model hierarchical structure and coupling. A header file is used for defining atomic models as a class. Ports, variables, and state definitions of an atomic model can be found in this file. Users can implement definitions of functions such as $\delta_{int}, \delta_{ext}and\lambda$ in the CPP file, according to the C++ programming language convention. Therefore, the behavioural of the systems is presented through the implementation of the atomic models.

Figure 21 shows an example of DEVS models, which includes one coupled model and three atomic models. This model has two levels: in the first level there is one coupled model (C1, known as the top model), and in the second level there are three

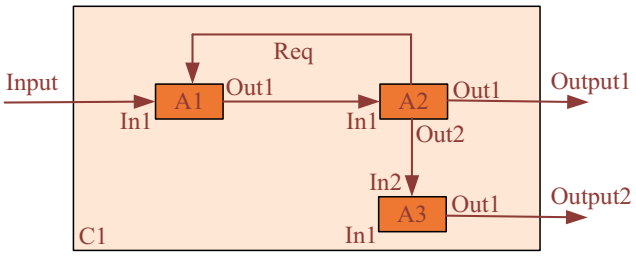

Figure 21. A coupled model with three atomic models.

atomic models (A1, A2, and A3). This model has one external input port and two output ports. The external input port is connected to the input port of A1. The 'Out1′ ports of A2 and A3 are connected to the 'Output1′ and 'Output2′ ports of the C1 coupled model, respectively. The rest of the interconnections can be seen in Figure 21.

Now, let us assume that A1 represents a simple queue behaviour. According to the Figure 21, the A1 atomic model has two input ports and an output port. A1 keeps the arrival messages (from the 'In1′ port), and it sends them to A2 one by one. To do so, A2 should send a request message to A1, whenever it is free to process a new message. Figure 22 shows the header file that defines the A1 atomic model as a class (Queue0). Moreover, Figure 23 presents a simplified behaviour of this atomic model. Finally, Figure 24 illustrates the structure of the DEVS model.

In Figure 22, we start by defining our desired queue class. We define the required methods such as initFunction, externalFunction, internalFunction, outputFunction, etc., in the protected zone. Then, we define the required ports, variables, and the needed states in the private zone. We use these states to implement the behaviour of the atomic model in the CPP file. According to the DEVS model in Figure 21, our atomic model (A1) has two input ports (In1 and Req) and one output port (out1). We use a variable (processTime) to show how much time this atomic model requires to process an input message. This parameter can be initialised either in initFunction, or it can be initialised by a value that CPP file will read from the Model file (please check the constructor method in Figure 23 and the A1 definition in Figure 24).

Figure 23 shows a CPP file that represents the behaviour of our queue atomic model. Before we start with the detail of Figure 23, we present a brief overview on the way that a CPP file works. In the CPP file, we usually implement the body of the defined methods in the Header file including but not limited to constructor, initialisation, external, internal, and output functions. The constructor method creates a sample of the desired atomic model. We use the initialisation function ("initFunction") to assign a value to the variables that required to be initialised at the beginning of the simulation. The external function of an atomic model is responsible to deal with the external messages that arrive at the atomic model. The internal function deals with the required actions that we need to take at each state. If we do not deactivate an atomic model in the external, internal, or initialisation functions, the controller goes to the output function after each execution of the external, internal, or initialisation functions. When there is no external message and the atomic model is active, the controller moves between the output and internal functions repetitively. Once an external message arrives, the controller calls the external function. We can use "passivate" method in the external, internal, or initialisation functions to deactivate an atomic model until it receives the next external message.

```
#ifndef __QUEUE0_H
#define __QUEUE0_H

#include "atomic.h"  // class Atomic
#include "string.h"
#include "list.h"

class Queue0 : public Atomic {
   public:
       Queue0( const std::string &name = "Queue0" );  //Default constructor
       virtual std::string className() const { return "Queue0"; }
       ~Queue0() {};

   protected:
       Model &initFunction();
       Model &externalFunction( const ExternalMessage & );
       Model &internalFunction( const InternalMessage & );
       Model &outputFunction( const InternalMessage & );

   private:
       const Port &In1,&Req;
       Port &Out1;
       Time processTime;
       typedef list<Value> ElementList ;
       ElementList elements;
       int Request;
       int Qlen;
       enum State{
                   Idle,       //initial state
                   Push,
                   Pop,
                 };
       State state;
};

#endif   //__QUEUE0_H
```

Figure 22. The header file of the DEVS atomic model (A1) in.Figure 21

We can use "holdIn" method in these functions to generate the desired delay before the controller calls the output function.

In Figure 23, in the constructor method, we define the ports and we also read the corresponding Model file (in Figure 24) to initialise the required parameters. Then, we use the initFunction to assign a value to the variables that required to be initialised. Moreover, we make sure that the queue is empty at the beginning of the simulation. Finally, by using the "passivate ()" method, we make the atomic model inactive but ready to be used in the simulation. In the "externalFunction", we make decision for the received messages according to their arrival port as well as the status of the queue. Here, we determine the next state of the atomic model and we may change the queue size based on the received message. We use "outputFunction" to send out the output messages through desired ports. In our case, if the atomic model state is "Pop" and the "Request" value is 1, we send the front element of the queue to the A2 through the 'out1′ port. In the "internalFunction", we deal with the different states that the A1 atomic model can have, and we program them to make sure this atomic model represents our desired queue behaviour.

```
#include "Queue0.h"      // class Queue
#include "message.h"    // class ExternalMessage, InternalMessage
#include <iostream>
#include <fstream>

Queue0::Queue0( const string &name ) :
Atomic( name ),
In1( addInputPort( "In1" ) ),
Req( addInputPort( "Req" ) ) ,
Out1( addInputPort( "Out1" ) ),
{
   string processTimeS(MainSimulator::Instance().getParameter(description(),
   "processTime"));
   if (processTimeS!="") processTime = Time(processTimeS);
}

Model &Queue0::initFunction(){
   Request = 0;
   Qlen = 0;
   state = Idle;
   elements.erase( elements.begin(), elements.end() ) ;
   passivate();
   return *this ;
}

Model &Queue0::externalFunction( const ExternalMessage &msg ){
   if( msg.port() == In1 ) // an input from the external input port{
      elements.push_back( msg.value() ) ; //Store Input value in Queue
      state = Push;
      Qlen++;
      holdIn( Atomic::active, ProcessTime);
   }
   else if ( msg.port() == Req && elements.size() > 0){
      Request = 1;
      state = Pop;
      holdIn( Atomic::active, ProcessTime);
   }
   else if ( msg.port() == Req  && elements.size() == 0){
      Request = 1; //Although Queue is empty but A2 is waiting for input.
      passivate(); // If the queue is empty then passivate.
   }
   return *this;
}

Model &Queue0::outputFunction( const InternalMessage &msg ){
   if ((state == Pop) && (Request == 1)){
      sendOutput( msg.time(), Out1, elements.front());
   }
   return *this ;
}

Model &Queue0::internalFunction( const InternalMessage & ){
   switch (state){
      case Idle:
         if (Request == 1) {state = Pop;}
         else {passivate();}
         break;

      case Push:
         state = Idle;
         break;


        case Pop:
           elements.pop_front();
           Qlen--;
           Request = 0;
           state = Idle;
           break;
     }
     return *this;
  }
```

Figure 23. The CPP file of the DEVS atomic model (A1) in Figure 21.

Figure 24 presents the Model file of the DEVS model in Figure 21. This file is used to define the structure of the DEVS model. In the Model file, we start with defining the top model (which is C1 in Figure 21). We mention all the components and the interconnection in the level one of the DEVS model. Then, we continue with defining the detail of the components of the level two. This process is repeated until we define all the components of various levels in order.

```
[top]
components : A1@queue0   A2@relatedClass1   A3@relatedClass2

in:   Input
out:  Output1  Output2

Link: Input    In1@A1
Link: Out1@A1  In1@A2
Link: Out2@A1  In1@A3
Link: Out2@A2  In2@A3
Link: Out1@A2  Output1
Link: Out1@A3  Output2

[A1]
processTime : 00:00:00:10   #pass values for the parameters

[A2]
…

[A3]
…
```

Figure 24. The model file of the DEVS model (C1) in Figure 21.