ZONE-BASED SENSING SCHEDULING APPROACH IN COGNITIVE RADIO SPECTRUM SENSING

by

Bilal ACAR

Mehmet Akif ERSOY

Submitted to the Department of Computer Engineering

in partial fulfillment of the requirements

for the degree of

Bachelor of Science

in

Computer Engineering

Boğaziçi University

November 2011

Table of Contents

[TABLE OF FIGURES 4](#_Toc309435317)

[ABSTRACT 5](#_Toc309435318)

[1. INTRODUCTION 6](#_Toc309435319)

[2. COGNITIVE RADIO SPECTRUM SENSING 7](#_Toc309435320)

[3. GENERAL DESIGN OF THE SOFTWARE 7](#_Toc309435321)

[3.1. MAIN PROGRAM 8](#_Toc309435322)

[3.2. MULTITHREADED SIMULATION MODULE 9](#_Toc309435323)

[3.2.1. Primary Traffic Generator Thread 9](#_Toc309435324)

[3.2.2. Cognitive Radio Traffic Generator Thread 11](#_Toc309435325)

[3.3. DISCERETE EVENT SIMULATION MODULE 11](#_Toc309435326)

[3.3.1. Primary Traffic Generator Simulation Entity 11](#_Toc309435327)

[3.3.2. Cognitive Radio Traffic Generator Simulation Entity 11](#_Toc309435328)

[3.4. SHARED PART OF THE MODULES 14](#_Toc309435329)

[4. DESIGN OF THE CLASSES 16](#_Toc309435330)

[4.1. ANIMATION PACKAGE 16](#_Toc309435331)

[4.1.1. Draw Area 16](#_Toc309435332)

[4.1.2. Draw Cell 16](#_Toc309435333)

[4.1.3. Plot 17](#_Toc309435334)

[4.1.4. Point Color 18](#_Toc309435335)

[4.1.5. Simulation Stats Table 19](#_Toc309435336)

[4.2. COMMUNICATION ENVIRONMENT PACKAGE 19](#_Toc309435337)

[4.2.1. Cell 19](#_Toc309435338)

[4.2.2. Wireless Channel 22](#_Toc309435339)

[4.3. DISCRETE EVENT SIMULATION PACKAGE 25](#_Toc309435340)

[4.3.1. CR DES Scheduler 25](#_Toc309435341)

[4.3.2. DES Primary Traffic Generator 26](#_Toc309435342)

[4.3.3. Primary Traffic Generator SimEnt 27](#_Toc309435343)

[4.4. MULTI THREADED SIMULATION PACKAGE 27](#_Toc309435344)

[4.4.1. CR Sensor Thread 28](#_Toc309435345)

[4.4.2. Primary Traffic Generator 29](#_Toc309435346)

[4.4.3. Primary Traffic Generator Thread 29](#_Toc309435347)

[4.5. NODES PACKAGE 30](#_Toc309435348)

[4.5.1. Node 30](#_Toc309435349)

[4.5.2. CR Base 31](#_Toc309435350)

[4.5.3. CR Node 33](#_Toc309435351)

[4.5.4. Primary Traffic Generator Node 37](#_Toc309435352)

[4.6. SIMULATION RUNNER PACKAGE 38](#_Toc309435353)

[4.6.1. Pareto Distribution 39](#_Toc309435354)

[4.6.2. Simulation Runner 39](#_Toc309435355)

[5. CONCLUSION 41](#_Toc309435356)

[BIBLIOGRAPHY 41](#_Toc309435357)

# TABLE OF FIGURES

[Figure 1 Main Flow Chart of The Software 9](#_Toc309435279)

[Figure 2 Primary Traffic Generator Thread 10](#_Toc309435280)

[Figure 3 Cognitive Radio Thread 12](#_Toc309435281)

[Figure 4 Primary Traffic Generator Simulation Entity 13](#_Toc309435282)

[Figure 5 Primary Cognitive Radio Simulation Entity 14](#_Toc309435283)

[Figure 6 Operations of Communication Schedule Advertiser 15](#_Toc309435284)

[Figure 7 Structure of CR Cell 20](#_Toc309435285)

[Figure 8 Frame Structure 25](#_Toc309435286)

# ABSTRACT

Project Name : Zone-Based Sensing Scheduling Approach in Cognitive Radio

Spectrum Sensing

Project Team : Bilal ACAR

Mehmet Akif ERSOY

Term : 2011/12 I. Semester

Keywords : Wireless Communications, Cognitive Radio

Summary :

**Cognitive radio (CR)** improves spectrum efficiency by enabling CR users to opportunistically reuse the idle spectrum bands of licensed users, i.e., primary users. To avoid causing interference to the primary users, spectrum sensing, which detects idle licensed bands, is one of the most important issues.

**Spectrum Sensing**: detecting the unused spectrum and sharing it without harmful interference with other users. It is an important requirement of the Cognitive Radio network to sense spectrum holes. Detecting primary users is the most efficient way to detect spectrum holes. Spectrum sensing techniques can be classified into three categories:

* *Transmitter detection*: cognitive radios must have the capability to determine if a signal from a primary transmitter is locally present in a certain spectrum. There are several approaches proposed:
  + matched filter detection
  + energy detection
  + cyclostationary feature detection
* *Cooperative detection*: refers to spectrum sensing methods where information from multiple Cognitive radio users is incorporated for primary user detection.
* *Interference based detection*.

In this project we will implement a simulator to analyze and optimize the efficiency of ***Zone-Based Sensing Scheduling in Cognitive Radio***. The project will construct a CR cell and zones in this cell. Then it will generate primary and secondary users both trying to communicate in the cell. It will measure the performance of the system with respect to various parameter sets. Here we assume that the secondary users use energy detection method for spectrum sensing.

# INTRODUCTION

In today’s world, wireless technologies are the basic mean of the communication. Obviously these technologies utilize available wireless channel bands which can be divided into two main groups. These groups are licensed bands and unlicensed bands. Most of the available spectrum is already allocated to current wireless technologies to communicate. However, these bands are not well utilized. So, even though there is available bandwidth to communicate for other emerging technologies since the bands are already allocated to previous technologies, these new technologies cannot have their own frequency bands.

Cognitive Radio (CR) has emerged as a solution for this problem. CR does not require its own licensed band instead it uses unutilized available portion bandwidth of other current technologies. CR’s communication is based on detecting spectrum holes of licensed or unlicensed users’ bandwidths. That is, it uses other technologies’ bandwidths while they are not using it themselves. During the communication of CR if a licensed (primary) user tries to use its own bandwidth, CR user changes its communication parameters, such as communication frequency, medium access protocol, to communicate from another available band. This whole communication cycle of CR contains four phases:

* *Spectrum Sensing:* A CR monitors the available spectrum bands, captures their information, and detects the spectrum holes.
* *Spectrum Decision:* Based on the spectrum availability, CR users can determine a channel. This operation not only depends on spectrum availability, but it is also determined based on internal (and possibly external) policies.
* *Spectrum Sharing:* Multiple CR users try to access the spectrum. CR network access should be coordinated in order to prevent multiple users colliding in overlapping portions of the spectrum.
* *Spectrum Mobility:* CR users are regarded as “visitors” to the spectrum. If primary users need a specific portion of the spectrum, then the CR users must continue in another vacant portion of the spectrum.

Among these phases spectrum sensing is one of the most important, difficult, and costly task. Because in case of miss detection both primary users’ and CR users’ communication will be garbage. On the other hand, in case of false alarm CR users will not use available channel although there is no primary user uses it, which will cause underutilization of available bandwidth. Due to these problems, in our project we try to optimize performance of spectrum sensing.

# COGNITIVE RADIO SPECTRUM SENSING

In spectrum sensing acknowledgment of spectrum sensing results from each of the cognitive users suffices a broad bandwidth since all of them will try to announce their own measurements at the same time.

Moreover, all collaborative sensing of spectrum in whole cell would cause some inefficient situations in spectrum sharing. For example, while in some region of the cell a frequency is not available, in some other regions it may be available. However, since spectrum sensing is done collaboratively in the entire cell it will be decided that if one frequency is not available in some region of the cell, it is not available anywhere. This situation will cause underutilization of available spectrum, and so inefficiency in bandwidth allocation.

In our project we develop a simulator to measure the performance of zone-based spectrum sensing approach and to optimize it. In zone based approach a cognitive radio cell is divided into sectors, sectors are divided into slices, slices are divided into sections. This final division constructs zones in the cell.

In each zone all CR users broadcasts their sensing measurements with low power. The first CR user who makes its broadcast declares itself as *leader* of zone. Since all CR users in the zone can hear each other, the leader will have the information about spectrum sensing measurements of other users in the zone. Therefore, the leader alone will have the information of sensing results in the entire zone. Hence, only leader’s acknowledgement to CR base about sensing result is necessary to acknowledge the base station. Consequently, instead of acknowledgement from all users in the zone only by making leader acknowledge the CR base station, the bandwidth requirement of spectrum sensing result acknowledgement will be minimized.

The zone-based spectrum sensing approach also produces a solution to second problem mentioned above. Since the sensing results are reported for each zone separately the CR base station can keep track of available frequencies for each zone separately. By doing so, it can decide a frequency as available in one zone while it is not in another. So, it can utilize that frequency also by assigning it to available zone.

# GENERAL DESIGN OF THE SOFTWARE

The simulation software we developed so far consists of two main modules. Each of these modules consists of two parts. Also, there is a shared part that both modules are commonly using. In the following sections, we will describe these modules and parts.

## 3.1. MAIN PROGRAM

The main structure of the program is consists of parsing input from the user, calling associated module according to the user’s input, and collecting and displaying output statistics. Program parses 5 main groups of input to start simulation. These groups are:

* *Main Options:*
  + How many primary users will be located in the Cognitive Radio cell
  + What is the channel model to compute the SNR distribution model
  + Duration of the simulation in terms of minutes
  + Maximum SNR value to compute distance relative SNR values
  + SINR threshold to declare a frequency as vacant
  + Random number seed, constant or random
  + Option to display animation or not
  + Option to plot SNR values or not
* *Traffic Options:*
* Traffic model of both primary and secondary users
* Expected number of calls per hour
* Expected call duration in terms of minutes
* *Frame Options: (All of duration are in terms of milliseconds)*
* Duration of sensing schedule advertisement
* Number of sensing slots and their durations
* Duration of sensing result reporting
* Duration of communication schedule advertisement
* Duration of communication
* *Frequency Options:*
* Number of available channels
* Bandwidth of each channel in terms of KHz
* *Zone Options:*
* Number of sectors in a cell
* Number of slices in each sector
* Number of sections in each slice
* Radius of the cell in terms of 100 meters
* Number of zones to simulate
* Position of the zones in the cell
* Number of CR users in zones

The program chooses a module according to user whether wants to display animation or not. General flow chart of the program is shown in the Figure 1.

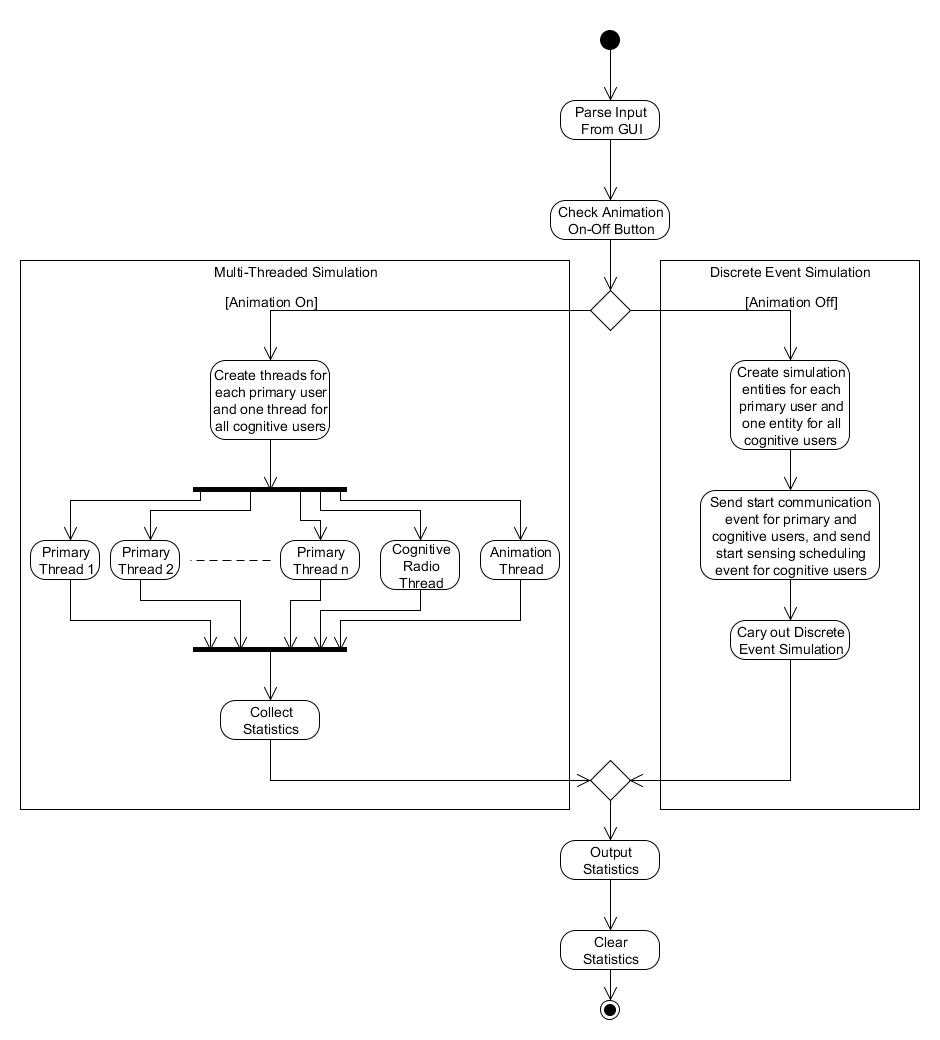


Figure 1 Main Flow Chart of The Software

## 3.2. MULTITHREADED SIMULATION MODULE

This module handles the simulation with animation. It runs two types of threads to perform the simulation. In the following subsections details of these threads are explained.

### 3.2.1. Primary Traffic Generator Thread

This thread handles operation of a primary traffic generator node in the simulation. It simply stays idle for a random amount of time. Then, it tries to occupy a free frequency to communicate and after random call duration it releases the frequency it occupied and restarts its cycle unless the simulation is finished. Also, at each time before it starts to communicate it changes its position randomly with respect to its idle duration. Flow chart of this thread’s operations is shown in Figure 2.

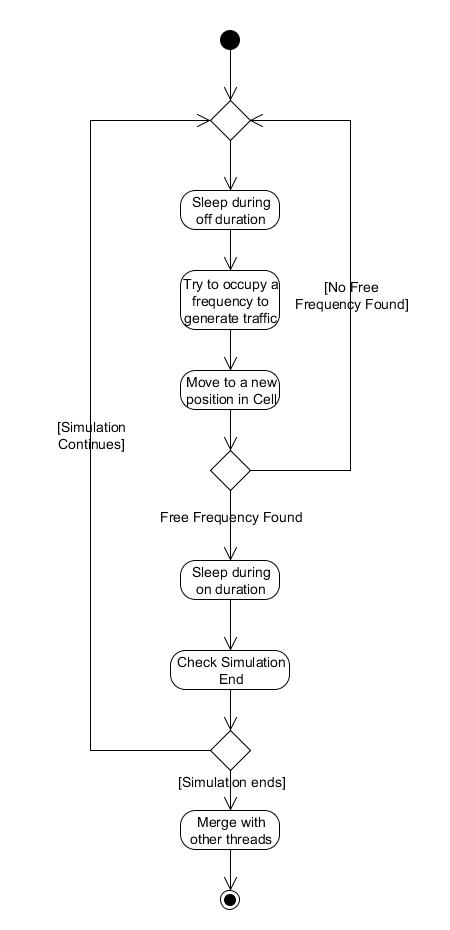


Figure 2 Primary Traffic Generator Thread

### 3.2.2. Cognitive Radio Traffic Generator Thread

This thread handles basic frame operations of all cognitive radio users simultaneously assuming that they are all synchronized. It keeps track of the current frame numbers and if a CR node has a communication event in this frame it handles it. It checks the current communication status of the node, if the node is communicating, it ends its communication; otherwise it sets the node as ready to communicate so that communication schedule advertiser can assign a frequency to that node if possible.

After handling communication events, it schedules sensing frames for all nodes, collects their sensing result, schedule communication frequencies for nodes that are ready to communicate or nodes that are collided in the previous frame. Finally it communicates the nodes and restarts it frame cycle unless the simulation is finished. Flow chart of this thread’s operations is in Figure 3.

## 3.3. DISCERETE EVENT SIMULATION MODULE

This module handles the simulation without animation. It uses a Discrete Event Simulation Framework explained in [1]. It constructs two types of simulation entities to perform the simulation. In the following subsections details of these simulation entities are explained.

### 3.3.1. Primary Traffic Generator Simulation Entity

This simulation entity handles operation of a primary traffic generator node in the simulation. It uses three types of events to send and receive between simulation entities. Then, it takes actions according to receiving events. The descriptions of these events are as follows:

* *Communication Start Event:* Upon receiving of this event, primary node’s communication is started. Random call duration for this node is assigned and end communication event for this node is sent.
* *Communication End Event:* Upon receiving of this event, primary node’s communication is ended and wait event for this node is sent.
* *Wait Event:* Upon receiving of this event random idle duration for this node is assigned and start communication event for this node is sent.

Flow chart of this thread’s operations is shown in Figure 4.

### 3.3.2. Cognitive Radio Traffic Generator Simulation Entity

This simulation entity handles operation of a CR traffic generator node in the simulation. It uses seven types of events to send and receive between simulation entities. Then, it takes actions according to receiving events. The descriptions of these events are as follows:

* *Sense Schedule Advertise Event:* Upon receiving of this event, CR base assigns CR nodes frequencies to sense and entity sends a sensing slot event to itself.
* *Sensing Slot Event:* Upon receiving of this event, CR sense the frequencies that are assigned to themselves and entity send a sensing slot event to itself if current slot is

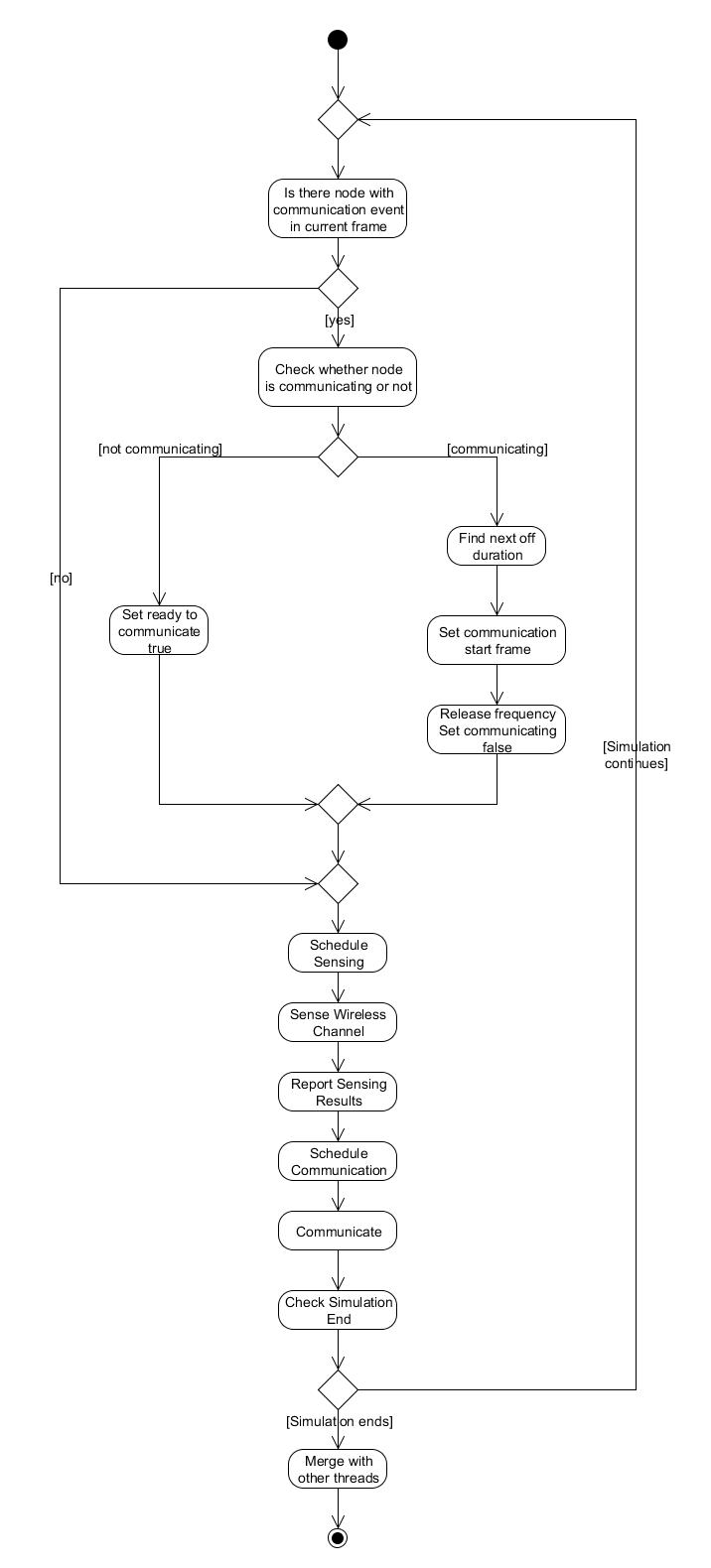


Figure 3 Cognitive Radio Thread

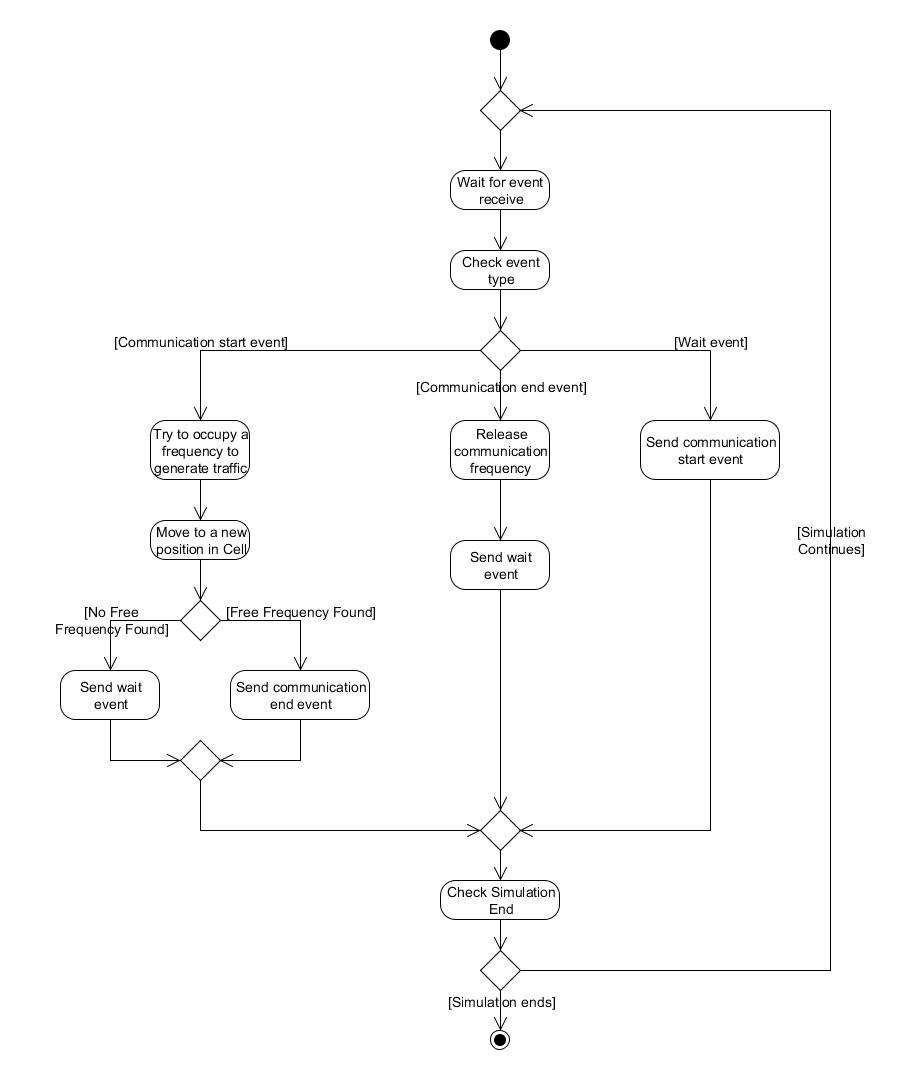


Figure 4 Primary Traffic Generator Simulation Entity

not the last slot or sends sensing result advertise event if the current slot is the last slot.

* *Sensing Result Advertise Event:* Upon receiving of this event, CR nodes report their sensing measurements to the CR base and this entity sends a communication schedule event to itself.
* *Communication Schedule Advertise Event:* Upon receiving of this event, CR base assigns communication frequencies to the nodes, that collided in the previous frame or that want to start communication in this frame, if possible.
* *Communicate Event:* Upon receiving of this event, CR nodes that are currently communicating generates traffic and report SINR values of the frequency that they are communicating in order to recognize collisions with primary nodes. Then this entity sends itself sense schedule advertise event.
* *Start Communication Event:* Upon receiving of this event, the node associated to this event is declared as ready to communicate.
* *End Communication Event:* Upon receiving of this event, random idle duration for the node associated with the arriving event is found and that nodes communication start frame is set. Also the communication frequency of the node is released. Finally, a communication start event for this node is sent to this entity.

Flow chart of this thread’s operations is shown in Figure 5.

## 3.4. SHARED PART OF THE MODULES

The main shared part of the software is communication schedule advertiser for CR nodes. This part finds free frequencies for each zone and checks whether there is any collided CR node. If so, it first tries to hand-off these collided nodes. After that operation if there are still free frequencies, it assigns them to the nodes that want to start communication in this frame.

Flow chart of this thread’s operations is shown in Figure 6.

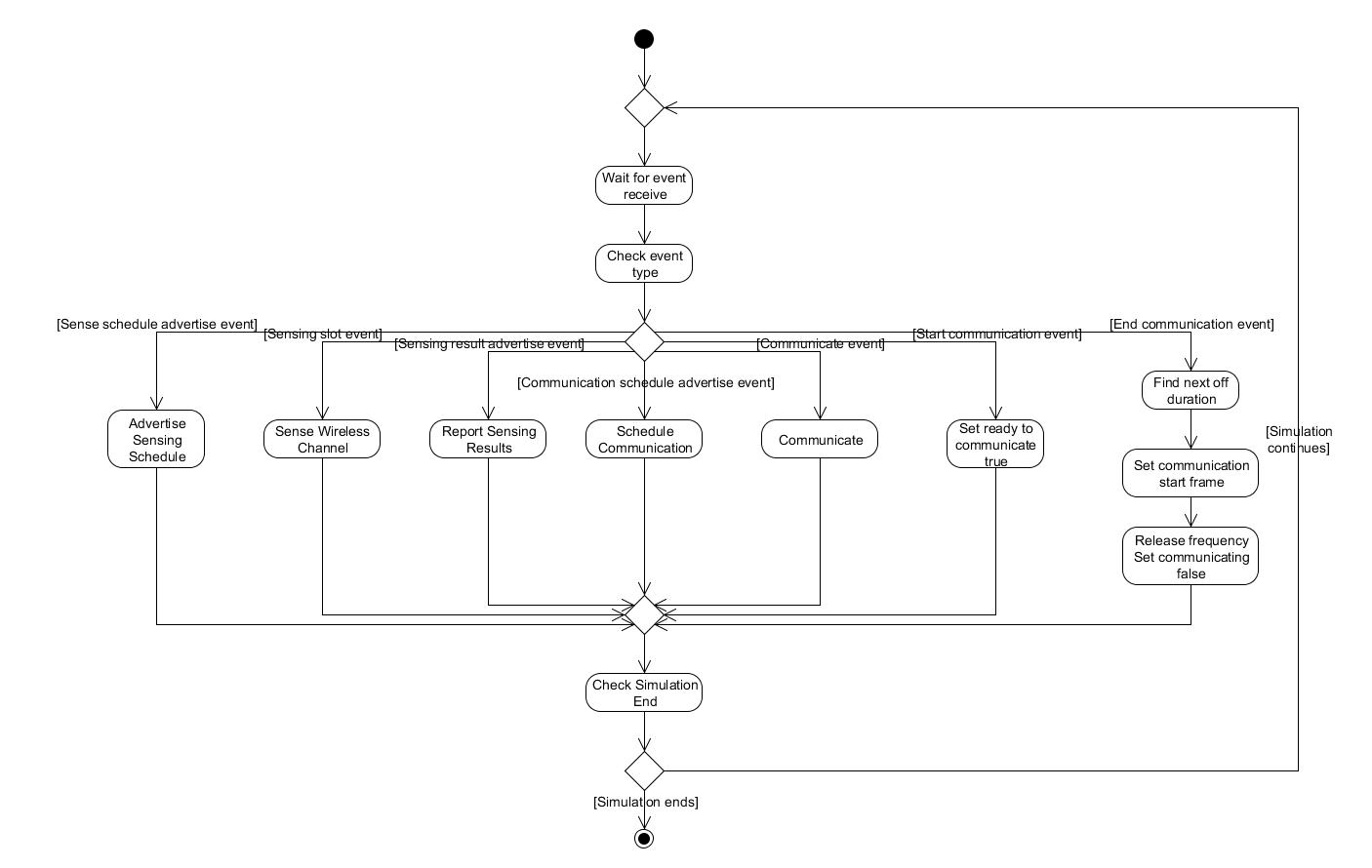


Figure 5 Primary Cognitive Radio Simulation Entity

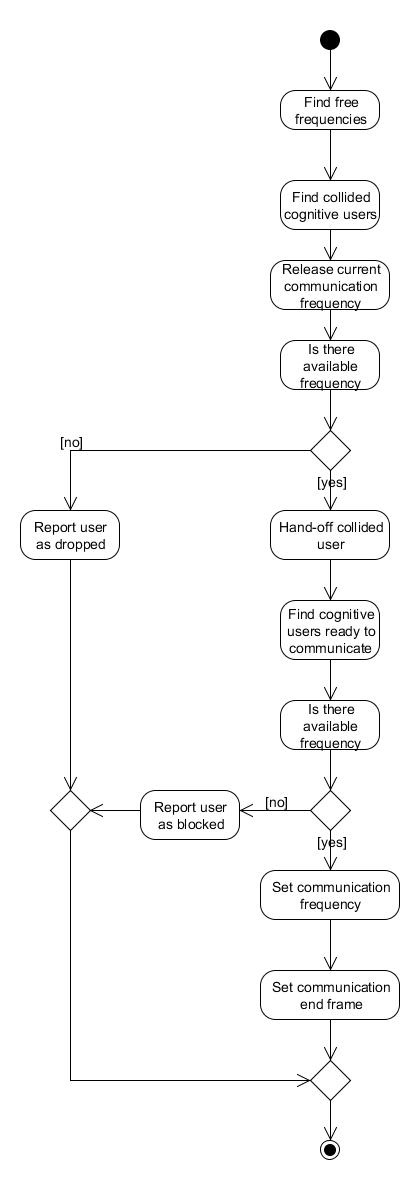


Figure 6 Operations of Communication Schedule Advertiser

# DESIGN OF THE CLASSES

The project consists of 19 classes which are grouped into 6 packages. Details of these packages and classes are explained in the following sections.

## 4.1. ANIMATION PACKAGE

### 4.1.1. Draw Area

This class extends JPanel class by adding additional functionalities for drawing basic cell structure, primary nodes, and secondary nodes. Basically, this class handles the animation window.

*public class* ***Animation.****DrawArea extends javax.swing.JPanel*

|  |  |
| --- | --- |
| Constructors | **public DrawArea(int cellRadius, int numberOfSectors, int numberOfAlpha, int numberOfDSections, int numberOfCrNodes, int numberOfPriNodes)**  Creates an animation window.  **Parameters**  cellRadius Radius - of the cell  numberOfSectors Number - of sectors in the cell  numberOfAlpha Number - of alpha slices in a sector  numberOfDSections Number of distance sections in a alpha slice  numberOfCrNodes Number - of CR nodes in the zone  numberOfPriNodes Number - of Primary nodes in the cell |
| Methods | **public void paintComponent(Graphics g)**  Overrides function from JPanel.  **public void paint(Graphics g)**  Overrides function from JPanel.  **public void paintPrimary(Integer id, PointColor p)**  This method paints a primary node with the given id, position, and color.  **Parameters**  id ID - of the primary node  p Position - of the CR node  **public void paintCR(Integer id, PointColor p)**  This method paints a CR node with the given id, position.  **Parameters**  id ID - of the CR node  p Position - of the CR node |

### 4.1.2. Draw Cell

This class creates a JFrame object and adds a DrawArea object to it. Initially it does not put any CR node or primary node in to the cell. It only draws the Cell and the zones in which the CR nodes will be deployed. It provides an interface for adding primary and CR nodes to the cell or changing the properties of currently existing ones.

*public class* ***Animation.DrawCell*** *implements java.lang.Runnable*

|  |  |
| --- | --- |
| Constructors | **public DrawCell(int radius, int numberOfSectors, int**  **numberOfAlpha, int numberOfDSections, int numberOfCrNodes,**  **int numberOfPriNodes)**  Constructs a cell structure to paint  **Parameters**  radius Radius of the cell  numberOfSectors Number of sectors in the cell  numberOfAlpha Number of alpha sections in a sector  numberOfDSections Number of distance sections in a alpha slice  numberOfCrNodes Number of CR nodes in the zone  numberOfPriNodes Number of primary nodes in the cell |
| Methods | **public void run()**  Overrides function from Runnable.  **public void draw()**  Draws the initial cell  **public static void paintPrimaryNode(Node n, Color c)**  Adds a Primary node to the cell with given color  **Parameters**  n - Node to be added  c - Nodes color: RED for transmitting or BLACK for sleeping  **public static void paintCrNode(Node n, Color c)**  Adds a CR node to the cell  **Parameters**  n - Node to be added  c - Color of the node  **public boolean isFinished()**  Returns whether the thread is finished or not  **Returns**  finished  **public void terminate()**  Terminates the thread |

### 4.1.3. Plot

This class plots a graph with provided x and y values. There can be many x values and associated y values. It can plot all of them on different windows or on the same window. It can also group different combinations. In this project it is used to draw the plots of SNR and SINR values versus time for each frequency. For plotting it uses jPlot[[1]](#footnote-2).

*public class* ***Animation.Plot***

|  |  |
| --- | --- |
| Constructors | **public Plot(int numberOfXs, ArrayList yPerX)**  Creates a new plotter object with no x value and y value.  **Parameters**  numberOfXs Number of different x values  yPerX How many y belongs to each x |
| Methods | **public boolean addPoint(int xPos, double xVal, ArrayList yVals)**  Adds an x value and its corresponding y values. This methods insert  values in a way that will keep x values array always sorted  **Parameters**  xPos Which x value collection the values belong  xVal x value to be added  yVals y values to be added  **Returns**  True if values added false otherwise.  **public void plot(String title, int xPos, int yPos)**  Creates the given x versus the given y value file to be used to plot its  graph. This method returns immediately if the y values is not valid.  **Parameters**  title title of the graph  xPos x values to be plotted  yPos y values to be plotted  **public void plot(String title, ArrayList xs, ArrayList ys, ArrayList**  **names)**  Plots the specified graphs onto the same window. It can plot 36 different  graphs onto the same window. It plots each of the graphs with different  colors and data point shapes. That is, no two different graphs will have  the same color and same data point shapes.  **Parameters**  title Title of the graphs  xs x values to plotted  ys y values to plotted  names Names of the graphs  **Throws**  IndexOutOfBoundsException  **public void plotAll()**  Plots all available x versus y values onto different graphs on different  windows  **public void plotAllX(int xPos)**  Plots all available y values belongs to given x value onto different  graphs on different windows  **Parameters**  xPos x value to be plotted  **public void plotAll(ArrayList names)**  This method plots corresponding y values of different x values on the  same window. It plots all possible graphs.  **Parameters**  names Names of the graphs that will be plotted on the same window |
|  |  |

### 4.1.4. Point Color

This class holds the position of a node on the animation screen. It also keeps its color and radius.

*public class* ***Animation.PointColor***

|  |  |
| --- | --- |
| Constructors | **public PointColor(int x, int y, int r, Color c)**  Constructs a PointColor with specied coordinate and color  **Parameters**  x x coordinate of the point  y y coordinate of the point  r Radius of the point  c Color of the point  **public PointColor(Point2D.Double p, int r, Color c, int unit)**  Constructs a PointColor from Point2D by scaling its coordinate and with  a specified color  **Parameters**  p Point2D to be converted  r Radius of the point  c Color of the point  unit Scale unit |
| Methods | **public Animation.PointColor convertCoordinate(int cellRadius)**  Convert this PointColor object from regular x-y coordinate system to  JFrame x-y coordinate system  **Parameters**  cellRadius Radius of the cell  **Returns**  Returns this object for cascading |

### 4.1.5. Simulation Stats Table

This class generates a statistic table for CR nodes and primary nodes. In the table, there are some statistics for each CR node, such as, number of call attempts, number of calls, number of frames communicated, number of blocks, number of drops, number of forced handoffs, and number of collisions. Also, there are two statistics for primary users: number of calls and total communication time.

*public class* ***Animation.SimulationStatsTable*** *extends javax.swing.JFrame*

|  |  |
| --- | --- |
| Constructors | **public SimulationStatsTable(String[][] crStats, String[][] priStats,**  **JFrame parent)**  Creates a statistic table for the simulation results. It displays two tables  with CR node statistics and Primary node statistics on them.  **Parameters**  crStats CR node statistics  priStats Primary node statistics  parent Parent of the this frame |

## 4.2. COMMUNICATION ENVIRONMENT PACKAGE

This package includes two classes which are basically constructing the communication environment.

### 4.2.1. Cell

The basic structure of the cell is as follows:

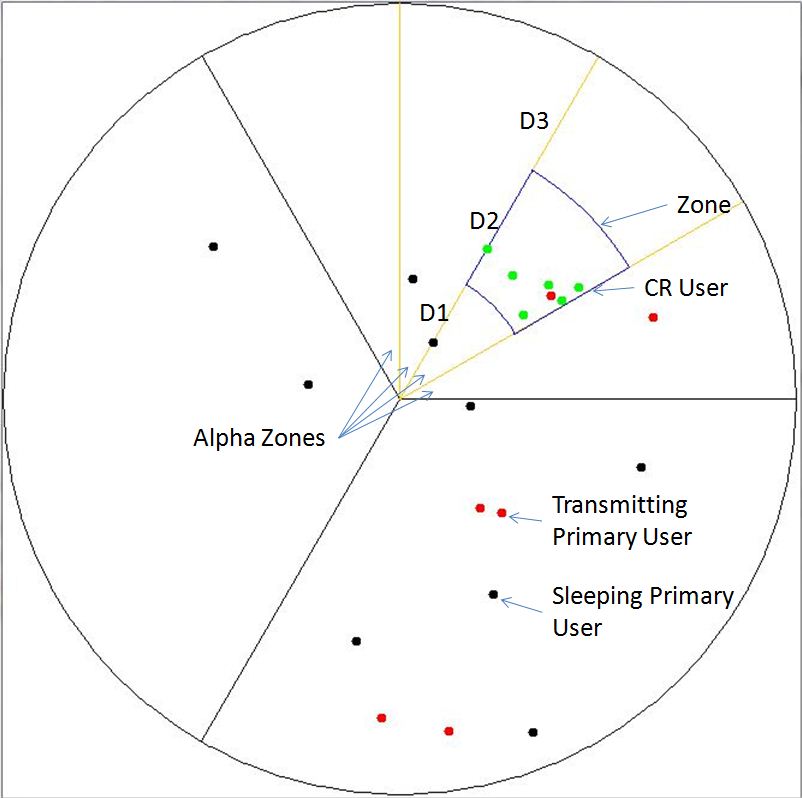


Figure 7 Structure of CR Cell

This class handles deployments of CR users and primary users into the cell. Initially, it deploys primary users to a random position in the cell, and it deploys CR users to a random position in a given zone. Whenever a primary user wants to talk, there is a method which deploys this primary user into a random position in the cell with respect to the primary user’ idle time.

This class also responsible from base station of the cell, number of sectors in the cell, number of slices in a sector, and number of zones in a slice.

*public class* ***CommunicationEnvironment.Cell***

|  |  |
| --- | --- |
| Constructors | **public Cell(CRBase baseStation, double radius, int**  **number\_of\_sectors, int alpha, ArrayList set\_of\_d)**  Constructor of the Cell  **Parameters**  baseStation CRBase  radius Radius of the Cell  number\_of\_sectors Number of sectors in the cell.  alpha Corresponding angle for a zone at the baseStation  set\_of\_d List of distances |
| Methods | **public static java.awt.geom.Point2D.Double deployNodeinCell()**  Finds a random position for a node in the Cell.  **Returns**  Position of the node.  **public static java.awt.geom.Point2D.Double deployNodeinZone(int**  **sector\_number, int angle\_number, int distance\_number)**  Finds a random position for a node in a specified zone.  **Note:** All the parameter values start from zero.  **Parameters**  sector\_number Sector number of the node.  angle\_number Angle number in the sector of the node.  distance\_number Distance of the zone to the center.  **Returns**  Position of the node.  **public static java.awt.geom.Point2D.Double**  **deployNodeInRouteCircle(Node node, double routeRadius)**  Deploys a node within a range it could have been relocated. This method is  designed for primary node relocation only. It is assured that the node will  never leave the cell.  **Parameters**  node Node to relocate  routeRadius Range of node's relocation  **Returns**  New location of the node  **public static void setBaseStation(CRBase baseStation)**  Sets a CRBase as a base station to the Cell.  **Parameters**  baseStation Base station  **public static CRBase getBaseStation()**  Gets the current baseStation of the Cell.  **Returns**  Base Station  **public void setPosition(Point2D.Double position)**  Sets a new position for the baseStation.  **Parameters**  position New Position of the baseStation  **public java.awt.geom.Point2D.Double getPosition()**  Gets the current position of the baseStation.  **Returns**  Position of the baseStation  **public static void setAlpha(int alpha)**  Sets a new alpha value for the Cell.  **Parameters**  alpha Degree of each slice in a sector  **public static int getAlpha()**  Gets the current alpha value of the Cell.  **Returns**  Alpha  **public static void setNumber\_of\_sectors(int number\_of\_sectors)**  Sets a new number of sectors value for the Cell.  **Parameters**  number\_of\_sectors Number of sectors in the Cell  **public static int getNumber\_of\_sectors()**  Gets the current number of sectors value of the Cell.  **Returns**  Number of sectors  **public static void setRadius(double radius)**  Sets a new radius value for the Cell.  **Parameters**  radius Radius of the Cell.  **public static double getRadius()**  Gets the current radius value of the Cell.  **Returns**  Radius of the Cell  **public static void setSet\_of\_d(ArrayList set\_of\_d)**  Sets a new distance list for the Cell.  **Parameters**  set\_of\_d Distance list  **public static java.util.ArrayList getSet\_of\_d()**  Gets the current distance list of the Cell.  **Returns**  Distance list |
| Fields | **public static set\_of\_d**  List of all distances between the baseStation and the zones which are in the  same sector and have the same angle interval with the baseStation.  Distances must be in the ascending order. |

### 4.2.2. Wireless Channel

This class provides available channels to both primary and secondary users. It keeps the information about which frequency is occupied by which node. It calculates SNR value for a CR node for a specified frequency according to whether the frequency is occupied by a primary user or not. It also calculates SINR value between two given CR nodes (one of these nodes can be CR base station) for a given frequency.

*public class* ***CommunicationEnvironment.WirelessChannel***

|  |  |
| --- | --- |
| Constructors | **public WirelessChannel(int channelModel, int**  **numberOfFrequencies, double maxSNR, double sinrThreshold,**  **double meanOffDuration, double meanOnDuration, int**  **trafficModel, double unitTime)**  Creates a wireless channel with the given model. It creates  numberOfFrequencies amount frequency. Initially there is no node in  the channel.  **Parameters**  channelModel 0 for Simple ch., 1 for Lognormal ch.  numberOfFrequencies Number of frequencies in the channel  maxSNR max SNR value of the channel  sinrThreshold SINR threshold for CR nodes to be able to  communicate without collision  meanOffDuration   * *If Poisson traffic model:* Mean number of calls per unit time * *If ON-OFF traffic model:* Mean OFF period duration of a node in terms of time units   meanOnDuration   * *If Poisson traffic model:* Expected value for duration of a call in terms of time units * *If ON-OFF traffic model:* Expected value for duration of a ON period in terms of time units   trafficModel Model for traffic generation  unitTime Scale of msec during animation |
| Methods | **public void registerNode(Node n)**  Registers a node to the channel.  **Parameters**  n node that will be registered  **public double generateSNR(Node sensor, int frequency)**  Finds an SNR value according to the channel model.  **Parameters**  sensor Node to assign SNR value  frequency Frequency to which the sensor senses  **Returns**  SNR value  **public double generateSINR(Node transmitter, Node receiver, int**  **freq)**  Finds a SNR value according to the channel model.  **Parameters**  transmitter Node transmitting the signal  receiver Node to assign SNR value  freq Frequency which will be used during the  communication between transmitter and receiver  **Returns**  SNR value at receiver caused by transmitter  **public void occupyFrequency(int frequency, Node n)**  Lets a primary user to occupy a frequency. That is, the primary user  starts transmission.  **Parameters**  frequency In which the user transmits  n Node that occupies the frequency  **public boolean isOccupied(int freq, int nodeType)**  Re turns whether a frequency is occupied by a given type of node  **Parameters**  freq ID of frequency  nodeType Type node, CR or Primary  **Returns**   * ***True*** if given frequency is occupied by a given type of node * ***False*** otherwise   **public void releaseFrequency(int frequency, Node n)**  Lets a primary user to release a frequency. That is, the primary user  finishes transmission.  **Parameters**  frequency In which the user transmits  n Node which releases the frequency  **public int freeFrequency()**  Finds a free frequency and returns its ID  **Returns**  ID of frequency. -1 if no free frequency  **public static double dbToMag(double db)**  Computes the magnitude of a given dB  **Parameters**  db dB value to be computed  **Returns**  magnitude equivalent of db  **public static double magTodb(double mag)**  Computes the dB of a given magnitude  **Parameters**  mag Magnitude value to be computed  **Returns**  dB equivalent of mag  **public int numberOfFreq()**  Computes the number of frequencies  **Returns**  Number of frequencies  **public double getMeanOnDuration()**  Returns the mean on (active) duration of all (both primary secondary)  users.  **Returns**  Mean on duration  **public double getMeanOffDuration()**  Returns the mean off (inactive) duration of all (both primary secondary)  users.  **Returns**  Mean off duration  **public int getTrafficModel()**  Returns the traffic model of all (both primary secondary) users.  **Returns**  Traffic Model  **public java.util.ArrayList getFreq(int freq)**  Returns both CR node and primary node using a given frequency  **Parameters**  freq ID of frequency to find its current users  **Returns**  Arraylist that contains current users of the given frequency |
| Fields | **public static final PRIMARY**  Integer value for Primary node  **public static final CR**  Integer value for CR node  **public static final SIMPLECH**  Simple Channel Model such that SNR = maxSNR / e^f(distance)  **public static final LOGNORMALCH**  Lognormal channel model  **public static final NOFREEFREQ**  There is no available frequency right now  **public maxSNR**  Max SNR value of the channel  **public sinrThreshold**  Minimum SINR threshold to be able to communicate  **public uniform**  Uniform distribution to accomplish frequency assignments  **public static final POISSON**  Poisson traffic model  **public static final ON\_OFF**  On-Off traffic model  **public static unitTime**  Scale of msec during animation |

## 4.3. DISCRETE EVENT SIMULATION PACKAGE

These classes are responsible for primary traffic generation and monitoring its effects on the CR nodes when the animationOffButton is selected. While the primary users produce a specified traffic, CR nodes sense the channels and communicate by using available channels.

### 4.3.1. CR DES Scheduler

This class handles the frame structure of the CR nodes. Whenever an event comes it calls the related functions and at the end of each part in the frame structure, it calls the next event according to the frame structure order. The main frame structure is as follows:

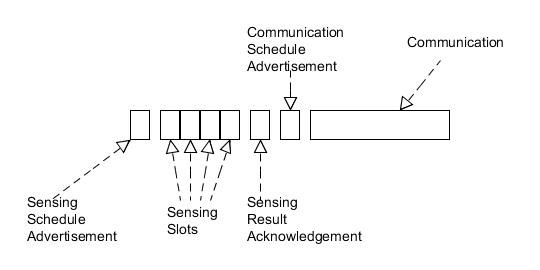


Figure 8 Frame Structure

*public class* ***DESSimulation.CRDESScheduler*** *extends SimEnt*

|  |  |
| --- | --- |
| Constructors | **public CRDESScheduler(double simulationDuration, double unitTime, int numberOfSlots, double slotDur, double senseScheduleAdvertisement, double commScheduleAdvertisement, double commDur, double senseResultAdvertisement)**  Creates a DES scheduler that performs frame action for CR sensor nodes  **Parameters**  simulationDuration Duration of the simulation in unit time  unitTime Unit of time in milliseconds  numberOfSlots Number of sensing slots in the frame  slotDur Duration of the sensing slots in terms of unit time  senseScheduleAdvertisement Duration of the sensing schedule  advertisement in terms of unit time  commScheduleAdvertisement Duration of the communication schedule  advertisement in terms of unit time  commDur Duration of the communication in terms of  unit time  senseResultAdvertisement Duration of the sensing result advertisement  in terms of unit time |
| Methods | **public void start()**  Starts the Simulation  **public void recv(SimEnt src, Event ev)**  Main frame operation  **Parameters**  src Source of the event  ev Occured Event  **protected void destructor()**  Sets and displays statistics about simulation and reset the GUI on the  program.  **public void terminate()**  Terminates the simulation  **public boolean isFinished()**  Returns whether the simulation is finished or not.  **Returns**  True if simulation is finished, false otherwise.  **public void deliveryAck(EventHandle h)**  Acknowledge event delivery  **Parameters**  h Handle of the event  **public double getRemainingSimulationDuration()**  Returns remaining simulation duration  **Returns**  Remaining simulation duration  **public double getSimulationDuration()**  Returns total simulation duration  **Returns**  Total simulation duration  **public void sendEndCommEvent(int crnode\_id)**  Sends a communication end event for a given CR node  **Parameters**  crnode\_id ID of the CR node  **public void sendStartCommEvent(int crnode\_id)**  Sends a communication start event for a given CR node  **Parameters**  crnode\_id ID of the CR node |

### 4.3.2. DES Primary Traffic Generator

This class holds primary traffic generation model and registered primary nodes. Also, it logs and prepares the statistics of the simulation related to primary nodes.

*public class* ***DESSimulation.DESPrimaryTrafficGenerator***

|  |  |
| --- | --- |
| Constructors | **public DESPrimaryTrafficGenerator()**  Creates a primary traffic generator with no node registered to it and with  the given model. |
| Methods | **public void registerNode(PrimaryTrafficGeneratorNode n)**  Registers a node and creates a Simulation entity for it to generate traffic  **Parameters**  n node to be registered  **public void start()**  Starts the simulation  **public java.lang.String[][] logStats()**  Logs and creates statistics of the simulation related to primary nodes  **Returns**  Primary node statistics of the simulation |

### 4.3.3. Primary Traffic Generator SimEnt

This class handles events related to one primary node's traffic generation. Whenever a primary user starts to talks, it finds the communication duration and sends a communication end event when this time passes. Likewise, if a primary node is stopped talking, it finds next communication time and sends a communication start event when this time passes.

*public class* ***DESSimulation.PrimaryTrafficGeneratorSimEnt*** *extends SimEnt*

|  |  |
| --- | --- |
| Constructors | **Public PrimaryTrafficGeneratorSimEnt**  **(PrimaryTrafficGeneratorNode node, double meanOnDuration,**  **double meanOffDuration)**  Creates a simulation entity for the given node with given probabilistic  mean values. It checks the traffic model from  DESPrimaryTrafficGenerator class.  **Parameters**  node Node associated to this simulation entity  meanOnDuration   * *If Poisson traffic model:* Expected value for duration of a call in terms of time units * *If ON-OFF traffic model:* Expected value for duration of a ON period in terms of time units   meanOffDuration   * *If Poisson traffic model:* Mean number of calls per unit time * *If ON-OFF traffic model:* Mean OFF period duration of a node in terms of time units |
| Methods | **public void recv(SimEnt src, Event ev)**  Main event handler  **Parameters**  src Source of the event  ev Occured event  **public void deliveryAck(EventHandle h)**  Acknowledge event delivery  **Parameters**  h Handle of the event |

## 4.4. MULTI THREADED SIMULATION PACKAGE

These classes are responsible for primary traffic generation and monitoring its effects on the CR nodes when the animationOnButton is selected. While the primary users produce a specified traffic, CR nodes sense the channels and communicate by using available channels.

### 4.4.1. CR Sensor Thread

This class handles the frame structure of the CR nodes. The main assumption about this class is all CR nodes are synchronized. It uses CRBase to advertise sensing and communication schedules and uses CRNodes registered to channel to sense the frequencies, advertise sensing results, and communicate. The main frame structure is the same with the one in the CR DES Scheduler.

*public class* ***MultiThreadedSimulation.CRSensorThread*** *implements java.lang.Runnable*

|  |  |
| --- | --- |
| Constructors | **public CRSensorThread(double simulationDuration, double**  **unitTime, int numberOfSlots, double slotDur, double**  **senseScheduleAdvertisement, double commScheduleAdvertisement**  **, double commDur, double senseResultAdvertisement)**  Creates a thread that performs simulation action for CR sensor nodes  **Parameters**  simulationDuration Duration of the simulation in unit time  unitTime Unit of time in milliseconds  numberOfSlots Number of sensing slots in the frame  slotDur Duration of the sensing slots in terms  of unit time  senseScheduleAdvertisement Duration of the sensing schedule  advertisement in terms of unit time  commScheduleAdvertisement Duration of the communication schedule  advertisement in terms of unit time  commDur Duration of the communication in terms of  unit time  senseResultAdvertisement Duration of the sensing result advertisement  in terms of unit time |
| Methods | **public void run()**  Main thread operation  **public boolean isFinished()**  Returns whether the thread is finished or not  **Returns**  finished  **public void terminate()**  Terminates the thread  **public double getRemainingSimulationDuration()**  Returns remaining simulation duration  **Returns**  Remaining simulation duration  **public double getSimulationDuration()**  Returns total simulation duration  **Returns**  total simulation duration  **public double getUnitTime()**  Return the ms per unit time  **Returns**  Unit time  **public double getFrameDuration()**  Returns the frame duration in terms of msec  **Returns**  Frame Duration  **public void setCommunationDuration(int crnode\_id)**  Sets the ending frame of communication of the given CR node  **Parameters**  crnode\_id ID of the CR node  **public void setInactiveDuration(int crnode\_id, boolean dropped)**  Sets the starting frame of communication of the given blocked or  dropped CR node  **Parameters**  crnode\_id ID of the CR node  dropped Indicates whether the node dropped or not |

### 4.4.2. Primary Traffic Generator

This class provides necessary probability distribution classes and semaphores for primary traffic generation by many threads. That is, PrimaryTrafficGenerationThread objects uses fields of this class for traffic generation and necessary critical section processes. Also initiations of these objects are accomplished by registering PrimaryTrafficGenerationNode objects to an instance of this class. This class also provides an interface to terminate all working PrimaryTrafficGenerationThreads. It also logs and prepares the statistics of the simulation related to primary nodes.

*public class* ***MultiThreadedSimulation.PrimaryTrafficGenerator***

|  |  |
| --- | --- |
| Constructors | **public PrimaryTrafficGenerator()**  Creates a primary traffic generator with no node registered to it |
| Methods | **public void registerNode(PrimaryTrafficGeneratorNode n)**  Registers a node and creates a thread for it to generate traffic  **Parameters**  n node to be registered  **public void terminateAllThreads()**  Terminates all associated threads  **public java.lang.String[][] logStats()**  Logs and creates statistics of the simulation related to primary nodes  **Returns**  Primary node statistics of the simulation |

### 4.4.3. Primary Traffic Generator Thread

This class produces a specified traffic model for an individual PrimaryTrafficGenerationNode. For now, they can only produce Poisson traffic. All of the threads uses the same random number generator by doing so the overall inter arrival time and call duration of primary users follows exponential distributions. Object of this class simply generates a random number inter arrival time, sleeps for that amount, selects a random free frequency, occupy it, generates a random call duration, sleeps for that amount, and so on.

*public class* ***MultiThreadedSimulation.PrimaryTrafficGeneratorThread*** *implements java.lang.Runnable*

|  |  |
| --- | --- |
| Constructors | **public PrimaryTrafficGeneratorThread**  **(PrimaryTrafficGeneratorNode n, double meanOnDuration,**  **double meanOffDuration)**  Creates a primary traffic generator thread associated with node n  **Parameters**  n Primary node associated with this thread  meanOnDuration   * *If Poisson traffic model:* Expected value for duration of a call in terms of time units * *If ON-OFF traffic model:* Expected value for duration of a ON period in terms of time units   meanOffDuration   * *If Poisson traffic model:* Mean number of calls per unit time * *If ON-OFF traffic model:* Mean OFF period duration of a node in terms of time units |
| Methods | **public void run()**  Main thread operation  **public boolean isFinished()**  Returns whether the thread is finished or not  **Returns**  finished  **public void terminate()**  Terminates the thread |

## 4.5. NODES PACKAGE

This package has classes which are concerning with primary users, the actual owners of the system, which generate traffic in the wireless channel and cognitive radio users which try to find an available frequency to talk by sensing the channel within assigned time interval, and they sense only assigned frequencies to them. Also, this package includes the CR Base, base station of the cell, class which is also a node.

### 4.5.1. Node

This class is an abstract class that has all common fields of all node classes. It has velocity, position and id fields for nodes, and it has all the get and set functions about these fields.

*public abstract class* ***Nodes.Node***

|  |  |
| --- | --- |
| Constructors | **public Node()**  Default constructor |
| Methods | **public java.awt.geom.Point2D.Double getPosition()**  Gets the current position of the Node  **Returns**  position of the node  **public double getVelocity()**  Gets the current velocity of the Node  **Returns**  Velocity of the node  **public void setPosition(Point2D.Double position)**  Sets a new position for the node  **Parameters**  position Position of the node  **public void setVelocity(double velocity)**  Sets a new velocity for the node  **Parameters**  velocity Velocity of the node  **public void setId(int id)**  Sets an id for the node.  **Parameters**  id Id of the node  **public int getId()**  Returns the current id.  **Returns**  Id of the node |
| Fields | **protected position**  Position of the node  **protected velocity**  Velocity of the node  **protected id**  Id o the Node |

### 4.5.2. CR Base

This class includes the fields and functions of the base station in the cell. There is a method called deploy\_freq in this class which returns frequency list which consists of frequency per CR node number of consecutive frequencies. Also, there is a method named assignFrequencies which assigns new frequencies to crnodes by using deploy\_freq and setFrequencyList of CRNode class and it updates the frequency\_list which basically keeps the number of listeners(crnodes) for each frequency.

There is another method called communicationScheduleAdvertiser.

* Firstly, this method calls findFreeFrequencies method which computes SNR value of the base station by computing the max distance between CR nodes and CR base for each zone, after that it computes the threshold value for collision decisions by using magTodb and dbTomag methods of Wireless Channel class. By using threshold value, it finds all available frequencies and adds them to free\_freq list , again, for each zone.
* Secondly, it calls handoffCollidedUsersInZones method to make forced handoffs for CR nodes which are collided in the previous frame. It releases all of collided CR nodes’ communication frequencies to avoid further collisions with the primary nodes which are also use the same frequency to communicate. Then, it tries to find new free frequencies for the collided CR nodes and updates numberOfDrops or numberOfForcedHandoff for CR nodes with respect to the result.
* Lastly, it finds all of the CR nodes(separate for each zone) which want to communicate and tries to find them available frequencies. It updates numberOfDrops for CR nodes.

Note: This method finds free frequencies for the collided CR nodes and after that, it finds frequencies for the CR nodes which are just wanted to talk. So, collided CR nodes have more priorities over other CR nodes.

*public class* ***Nodes.CRBase*** *extends Node*

|  |  |
| --- | --- |
| Constructors | **public CRBase(Point2D.Double pos, int id, int**  **number\_of\_freq\_per\_crnode)**  Creates a CRBase at the given position.  **Parameters**  pos Position of the Base station  id Id of the CRBase  number\_of\_freq\_per\_crnode The number of the frequencies that a  CRNode is going to listen in a frame. |
| Methods | **public java.util.ArrayList deploy\_freq(boolean startFromFirst)**  Takes number\_of\_freq\_per\_crnode frequencies from frequency list by  paying attention to the order of the frequencies.  **Parameters**  startFromFirst If true the deployed frequency values will start from  the first frequency  **Returns**  number\_of\_freq\_per\_crnode frequencies  **public void assignFrequencies()**  Assigns frequencies to the crnodes to be listened in the sensing slots and  also updates frequency list.  **public void communicationScheduleAdvertiser()**  First, finds the threshold value for the collision purposes. Second, finds  available(free) frequencies by using threshold value. Third, deploys  these free frequencies to the crnodes to communicate at the next frame.  **public void setLast\_averageSnr(ArrayList current\_averageSnr)**  Updates average\_SNR values. Assigns the previous current\_averagesnr  value to the last\_averagesnr.  **Parameters**  current\_averageSnr Most up-to-date SNR value.  **public java.util.ArrayList getFrequency\_list()**  Returns the frequency\_list  **Returns**  frequency\_list  **public void registerZone(int sector, int alpha, int d, int crnodes)**  Takes parameters of a zone and registers this zone into registeredZones  **Parameters**  sector Sector of the zone  alpha Alpha number of the zone  d Distance section of the zone  crnodes Total number of crnodes in that zone  **public java.awt.geom.Point2D.Double deployNodeinZone(int id)**  Takes id of crnode then finds that crnode's zone, after this, calls  deployNodeinZone function with the corresponding zone parameters.  **Parameters**  id ID of CRNode  **Returns**  Point of the CRNode  **public int findZone(int id)**  Finds zone ID of a given CR node  **Parameters**  id ID of CR node  **Returns**  Zone ID of the given CR node  **public double farthestZoneDistance()**  Finds farthest point of the all registered zones to base station  **Returns**  Distance of the farthest point |
| Fields | **public registeredZones**  Keeps sector number, alpha number, d number, and number of CR nodes  of a zone  **public static final SECTOR**  Index of sector number in zone array  **public static final ALPHA**  Index of alpha number in zone array  **public static final D**  Index of d number in zone array  **public static final CRNODES**  Index of number of CR nodes in zone array |

### 4.5.3. CR Node

This class extends Node class. It has a “sense” method which simply updates the SNR value of a given frequency and helps to compute average SNR of that frequency by adding the new SNR value to the old record and updates the old record with new one. While updating the SNR value of a frequency, it uses the generateSNR method of the Wireless Channel class by sending the CR node(itself) and the frequency to that method. It has logSnrValues, logAverageSnr and some other methods to save some SNR informations into our log file.

There is another important method in this class, communicate, which checks if there is a collision or not and writes the conclusion to log file in both cases. This method computes sinr values of frequencies and compares them with sinrThreshold value in wireless channel class and then makes a decision about collisions.

There are also nextOnDuration/nextOffDuration functions which simply return the next on/off duration in terms of number of frames according to the traffic model for Multithreaded Simulation. Similarly, it has nextOnDurationDES/ nextOffDurationDES functions which do the same job with the above functions but these are for the discrete event simulation.

*public class* ***Nodes.CRNode*** *extends Node*

|  |  |
| --- | --- |
| Constructors | **public CRNode(int id, Point2D.Double pos, double vel)**  Creates a CRNode with the given frequencies, position and velocity  values.  **Parameters**  id ID of this CR node  pos Position of the CRNode  vel Velocity of the CRNode |
| Methods | **public void sense(int freq)**  Updates the SNR value of the frequency.  **Parameters**  freq Number of the frequency in snrValues list.  **public java.util.HashMap getSnrValues()**  **Returns**  Snr values of each frequencies which are assigned to this node.  **public static void initializeAverageSnr(int**  **total\_number\_of\_frequencies, int numberOfZones)**  It creates the averageSnr arraylist and initially add zeros to the elements.  **Parameters**  total\_number\_of\_frequencies Total number of frequencies  numberOfZones Number of zones currently simulating  **public void logSnrValues()**  Writes the id of the CRNode, position of the CRNode and snrValues of  the CRNode to the log file, respectively.  **public static void logAverageSnr(double time)**  Calculates average SNR values then writes these values to the log file and  then resets the average SNR values.  **Parameters**  time Current time  **public static void createLogFile(String file\_name)**  Creates the log file.  **Parameters**  file\_name Name of the log file  **public static void writeLogFile(String log\_string)**  Writes the input string to the log file.  **Parameters**  log\_string String  **public static void closeLogFile()**  Closes the log file.  **public void setFrequencyList(ArrayList frequencies)**  Sets frequency list to listen in the sensing slots.  **Parameters**  frequencies Frequency list  **public void setCommunication\_frequency(int**  **communication\_frequency)**  Sets a frequency to communicate  **Parameters**  communication\_frequency Communication frequency  **public int getCommunication\_frequency()**  Returns the current communication frequency of this CR node  **Returns**  Current communication frequency  **public void releaseCommunication\_frequency()**  Releases its communication frequency  **public static void communicate(double time, boolean lastReport)**  Checks whether a collision occured or not if there is an assigned  frequency to the crnode for that frame and resets the collision value if it  is the last report.  **Parameters**  time Time of the simulation  lastReport True if it is the last report otherwise false  **public static void setTotalNumberOfFrames(int**  **totalNumberOfFrames)**  Sets the total number of frames for a simulation.  **Parameters**  totalNumberOfFrames Number of Frames  **public static int getTotalNumberOfFrames()**  Returns total number of frames.  **Returns**  Total number of frames  **public static java.lang.String[][] logStats()**  Calculates and writes CrNode statistics to the log file.  **Returns**  CrNode statistic values  **public double nextOnDurationDES(double frameDuration)**  Finds the next on duration according to the traffic model for DES  **Parameters**  frameDuration Duration of one frame  **Returns**  On duration  **public double nextOffDurationDES(double frameDuration)**  Finds the next off duration according to the traffic model for DES  **Parameters**  frameDuration Duration of one frame  **Returns**  Off duration  **public int nextOnDuration(double frameDuration)**  Finds the next on duration in terms of number of frames according to the  traffic model for Multithreaded Simulation  **Parameters**  frameDuration Duration of one frame  **Returns**  On duration  **public int nextOffDuration(double frameDuration)**  Finds the next off duration in terms of number of frames according to the  traffic model for Multithreaded Simulation  **Parameters**  frameDuration Duration of one frame  **Returns**  Off duration  **public boolean getCommOrNot()**  Returns whether this CR node is currently communicating or not  **Returns**   * ***True*** if node is currently communicating * ***False*** otherwise   **public void setReadytoComm(boolean readytoComm)**  Sets whether this node can start communication in this frame or not  **Parameters**  readytoComm Indicates this node can wants to start  communicating in this frame if possible  **public boolean getReadytoComm()**  Returns whether this CR node wants to start communicating in this  frame or not  **Returns**   * ***True*** if node wants to start communicating in this frame * ***False*** otherwise   **public boolean getIsCollided()**  Returns whether this CR node is collided in previous frame or not  **Returns**   * ***True*** if node collided in previous frame * ***False*** otherwise   **public void setIsCollided(boolean iscollided)**  Sets whether this node is collided in this frame or not  **Parameters**  iscollided Indicates this node collided in this frame or not  **public int getNumberOfForcedHandoff()**  Returns the number of force handoffs this node has made  **Returns**  Number of forced handoffs  **public void setNumberOfForcedHandoff(int**  **numberOfForcedHandoff)**  Sets the number of forced handoffs this node has made  **Parameters**  numberOfForcedHandoff Number of forced handoffs  **public int getNumberOfDrops()**  Returns how many times this node has dropped  **Returns**  Number of drops  **public void setNumberOfDrops(int numberOfDrops)**  Sets how many times this node dropped  **Parameters**  numberOfDrops Number of drops  **public int getNumberOfBlocks()**  Returns how many times this node has blocked  **Returns**  Number of blocks  **public void setNumberOfBlocks(int numberOfBlocks)**  Sets how many times this node blocked  **Parameters**  numberOfBlocks Number of blocks |
| Fields | **public numberOfBlocks**  Count of how many times this CR node is blocked  **public final startCommEvent**  An event to start communication for this CR node  **public startEventHandle**  Handle of the start communication event of this CR node  **public final endCommEvent**  An event to end communication for this CR node  **public endEventHandle**  Handle of the end communication event of this CR node |

#### 4.5.3.1. Start Communication Event

This class handles communication start of CR nodes

*public static class* ***Nodes.CRNode.StartCommunicationEvent*** *implements* ***Event***

|  |  |
| --- | --- |
| Constructors | **public CRNode.StartCommunicationEvent(int crnode\_id)**  Constructor of this event  **Parameters**  crnode\_id ID of the associated CR node |
| Methods | **public void entering(SimEnt locale)**  Method to handle entering of this event to a simulation entity  **Parameters**  locale Simulation entity that this event entering |
| Fields | **public id**  ID of the associated CR node |

#### 4.5.3.2. End Communication Event

This class handles communication end of CR nodes

*public static class* ***Nodes.CRNode.EndCommunicationEvent*** *implements* ***Event***

|  |  |
| --- | --- |
| Constructors | **public CRNode.EndCommunicationEvent(int crnode\_id)**  Constructor of this event  **Parameters**  crnode\_id ID of the associated CR node |
| Methods | **public void entering(SimEnt locale)**  Method to handle entering of this event to a simulation entity  **Parameters**  locale Simulation entity that this event entering |
| Fields | **public id**  ID of the associated CR node |

### 4.5.4. Primary Traffic Generator Node

This class is also an extended class of Node class. This class handles basic operations of Primary nodes, such as set a new position, generate traffic (finds free frequency and occupies it). It also logs output statistics of primary nodes.

*public class* ***Nodes.PrimaryTrafficGeneratorNode*** *extends Node*

|  |  |
| --- | --- |
| Constructors | **public PrimaryTrafficGeneratorNode(Point2D.Double pos, double**  **vel, int id)**  Constructor of the PrimaryTrafficGeneratorNode  **Parameters**  pos Position of the node  vel Velocity of the node  id ID of this node |
| Methods | **public void setRandomPosition(double offDuration)**  Sets a new position for the primary traffic generator node.  **Parameters**  offDuration Previous off duration  **public int generateTraffic(double offDuration)**  Finds a free frequency and occupies it. This method is synchronized.  That is only one thread at a time can run it  **Parameters**  offDuration Previous off duration  **Returns**  ID of the occupied frequency  **public int getCommunicationFreq()**  Returns the current communication frequency of this Primary node  **Returns**  Communication frequency  **public int getNumberOfCallAttempts()**  Returns how many times this Primary node attempted to communicate  **Returns**  Number of call attempts  **public int getNumberOfDrops()**  Returns how many times this Primary node is dropped  **Returns**  Number of drops  **public double getComunicationDuration()**  Returns the total communication duration of this primary node  **Returns**  Communication duration  **public void incrementTotalCommunicationDuration(double**  **commDur)**  Increments communication duration of this Primary node  **Parameters**  commDur Last communication duration  **public double getRoutingRadius()**  Returns routing radius of a primary node  **Returns**  Routing radius  **public static java.lang.String[][] logStats(HashMap**  **registeredNodes)**  Logs output statistics of Primary nodes  **Parameters**  registeredNodes List of registered Primary nodes  **Returns**  Primary node statistics values |

## 4.6. SIMULATION RUNNER PACKAGE

This package consists of Simulation Runner class which is the main class in our simulation, and Pareto Distribution class. The forthcoming distributions for the simulation will be implemented in this package.

### 4.6.1. Pareto Distribution

This class implements Pareto distribution by using Uniform distribution class of CERN package. However, pareto distribution is not usable in the simulation system, for now.

*public class* ***SimulationRunner.ParetoDistribution***

|  |  |
| --- | --- |
| Constructors | **public ParetoDistribution(double shapeParameter, double**  **minValue, RandomEngine randomEngine)**  Creates a Pareto distribution  **Parameters**  shapeParameter Shape parameter of the distribution  minValue Minimum value of the distribution  randomEngine Random number generator  **public ParetoDistribution(double meanValue, RandomEngine**  **randomEngine)**  Creates a Pareto distribution with given expected value.  Note: Minimum value of the distribution is assumed to be 1, and so  mean value cannot be less than 2.  **Parameters**  meanValue Expected value of the distribution  randomEngine Random number generator |
| Methods | **public double cdf(double x)**  Returns the cumulative distribution function  **Parameters**  x Point to evaluate the value of the function  **Returns**  Cumulative distribution function  **public double nextDouble()**  Returns a random number from the distribution  **Returns**  A random number form the distribution  **public double pdf(double x)**  Returns the probability density function  **Parameters**  x Point to evaluate the value of the function  **Returns**  Probability density function  **public void setMinValue(double minValue)**  Sets the minimum value of the distribution.  **Parameters**  minValue New minimum value  **public void setShapeParameter(double shapeParameter)**  Sets the shape parameter value of the distribution.  **Parameters**  shapeParameter New shape parameter value |

### 4.6.2. Simulation Runner

The main class of the project is SimulationRunner class. This class initializes the GUI components of the project. Then it parses the input provided by the user. It initiates objects of Cell, WirelessChannel, CRBase, DrawCell, and Plot classes according to that input. If the animation button is selected then, it starts Cr sensor threads and primary traffic generator threads. Otherwise, it starts CrDesSchedular and priTrafGenDes. If the animation off button is selected, simulation is done in a way of discrete simulation event. In each case, Cr nodes will produce Poisson traffic.

*public class* ***SimulationRunner.SimulationRunner*** *extends javax.swing.JFrame*

|  |  |
| --- | --- |
| Constructors | **public SimulationRunner()**  Constructor of the class. This method initializes the GUI. |
| Methods | **public static void main(String[] args)**  **Parameters**  args the command line arguments  **public void startSimulation()**  Initializes the main simulation threads or schedulers  **public static void clear()**  Clears the data of the simulation |
| Fields | **public static wc**  Main wireless channel which all types of nodes are accessing  **public static cell**  Cognitive radio cell structure  **public static priTrafGen**  Primary traffic generator thread for wireless channel frequencies  **public static crSensor**  Thread responsible for frame structure of CR nodes  **public static crDesScheduler**  DES Framework event scheduler for constructing frame structure of CR  nodes  **public static priTrafGenDes**  DES Framework Primary traffic generator for wireless channel  frequencies  **public static crBase**  Base station of CR cell  **public static crNodes**  CR nodes which sense the wireless channel  **public static priTrafGenNodes**  Primary traffic generator nodes which cause traffic in wireless channel  **public static randEngine**  Random number generator for all random number generation operations  in the simulation  **public static plot**  Plots the time versus average SNR and SINR values graphs  **public static runner**  Currently running SimulationRunner instance  **public static terminateSimulation**  Button to terminate an ongoing simulation  **public static progressBar**  Progress bar to show progress of the simulation  **public static animationOnButton**  **public static animationOffButton**  **public static plotOnButton**  **public static plotOffButton** |

# CONCLUSION

The problem that we concerned is stated and a solution method is developed. Coding is started to simulate the system described and optimize the problem. Next we will complete full operations of the simulator, construct an analytical model, optimize that model, and compare the results.

The operations that we will complete about the simulator are:

* Sensing capabilities for CR nodes will be added.
* Detection with binary hypothesis testing will be added.
* Probability of false alarm and miss detection calculation will be added.
* Other channel models will be implemented.
* Other traffic models will be implemented.
* Throughput calculation will be added.

Thank you for reading this document. You can reach further information and updates about our project from Google Code (<http://code.google.com/p/cmpe492-cr-sensing-simulation/>) or Ohloh (<https://www.ohloh.net/p/cr-sensing>).

For more information you can send email to [bilal.acar.a@gmail.com](mailto:bilal.acar.a@gmail.com) or to [makifersoy2006@gmail.com](mailto:makifersoy2006@gmail.com).

# BIBLIOGRAPHY

[1] B. Khan, A. A.-F. (2010). *Network Modelling and Simulation.* WILEY.

1. http://tcptrace.org/jPlot/ [↑](#footnote-ref-2)