|  |
| --- |
| University of manitoba |
| Graphical Components for Directed Diffusion Simulator |
| ECE 7650 Final Report |
|  |
| **Mathew Grabau, 6828495** |
| **3/23/2012**  **Presented to: Dr. K. Ferens, P. Eng.** |

|  |
| --- |
| *This document presents the design for a series of graphical components to the Directed Diffusion Simulator developed by the 2011/2012 ECE 7650 Advanced Computer Network Architecture Course. UML techniques are used to present class structures and operational concepts. The components are designed and implemented to permit stand-alone or concurrent use. As/if work continues on the simulator more components should be developed to enhance the overall simulator quality.* |

Graphical Components for Directed Diffusion Simulator

# Introduction

As computers have evolved, engineers and scientists have made excellent progress in the fields of modelling and simulation. Simulation packages/frameworks, for example the widely used Simulink™, employ visualizations to give the simulation life and allow for monitoring. Data is often presented in plots, allowing the user to identify trends and check constraints at glance. Another common element is the ability to view the raw data in some sort of tabular or grid form.

Simulation plays a large role in assessing feasibility and performance of the Wireless Sensor Networks (WSNs). This project focused on providing enhancements for a simulator

# Background

Directed Diffusion Simulator provides several classes that are framework to the. The original implementation of the Directed Diffusion Simulation (DDSim) uses console-based output in its examples. Users currently have to write extract their own data and results. By introducing a series of user interface components to the simulation, some of this work is eliminated or simplified with the use of pre-built extensible code modules. Under these conditions, the usability and overall user experience of the simulator is greatly increased. The user is also more freely available to focus on simulation.

# Design Requirements

The general design requirements for DDSim UI components were:

1. Provide a plotting mechanism for the data being collected:
   1. Node energy variance was the base reference/base implementation.
   2. Export the data to a file for use in other plotting and processing applications.
2. Yield reusable components that can be extended as other simulator users may desire.

## Work Methodology

The software components were modeled in the Enterprise Architect CASE (Computer Aided Software Engineering) tool [1]. Coding was done using the eclipse IDE [2].

## Decoupled Modules

In order to make the components usable and extendable by future users/programmers, all the components are decoupled.

# Baseline Modifications

During the initial prototyping, the number of classes in the default package grew to unsustainable levels. As a result, the classes were migrated from the default package into the following packages:

1. dd – the original
2. dd.ui – generic classes used for supporting the
3. dd.ui.log – contains the logger component related code
4. dd.ui.nodelist – contains the documentation
5. dd.ui.test – general tests for some of the components

# Detailed Designs and Implementation

## Node List

The purpose of the node listing packet (dd.ui.nostlist) is to provide a tabular view of all node data at once – a monitor of sorts. The overall component is implemented as a composition of less-complicated components. The interaction of the classes is shown in Figure 1.

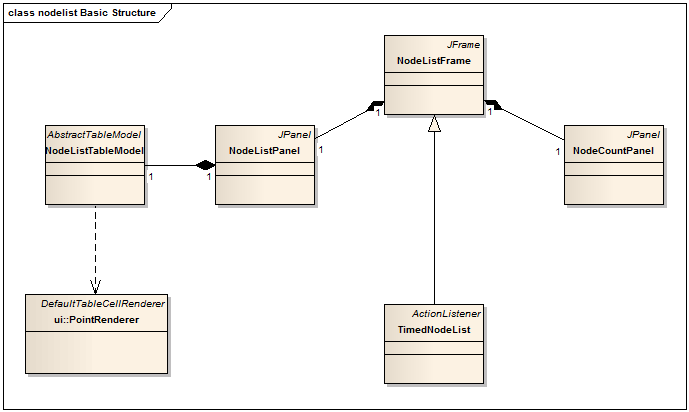


Figure 1 dd.ui.nodelist Classes

### NodeListTableModel

The class diagram for the NodeListTableModel is shown in Figure 2. Five columns were selected for the table model:

1. Node Id
2. Location
3. Number of Neighbors
4. Number of Interests
5. Energy Used

The NodeListTableModel’s core method is updateModel. This method runs through the sourceData member and updates the data Vector. When completed, the fireTableDataChanged method is called to trigger the event that will update the JTable to update [3].

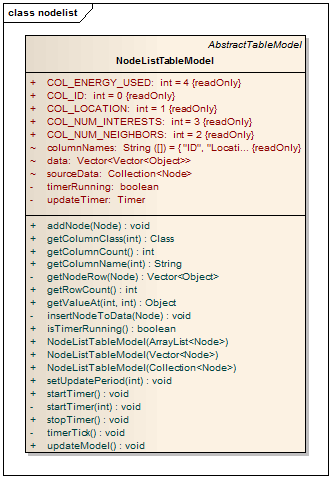


Figure 2 NodeListTableModel Class



NodeLIstPanel

NodeCountPanel

Figure 3 TimedNostList composition, while running in a simulation

### TimedNodeList

The TimedNodeList component, shown operating in Figure 3, implements useful functionality for most users:

1. NodeCountPanel – the top section of the frame shows the number of nodes that are included in the grid.
2. Pause and Resume – the buttons at the top/bottom of the frame will pause and resume the collection of data. The data will be properly synchronized when the simulation resumes.
3. Periodic updating – the component queries and updates the data based on the programmed update schedule.

The class diagram for the TimedNodeList and its parent are shown in Figure 4 and Figure 5, respectively. TimedNodeList’s startUpdateTimer(int) method allows the simulation programmer to configure the update schedule for the model.



Figure 4 TimedNodeList Class Diagram



Figure 5 NodeListFrame Class Diagram

### Results

A shown in Figure 3 the node list collects data, and testing has proved the pause and resume functions to be correct.

Early on in development there was a synchronization issue when the node data was being updated, though this was resolve

## Log Frame (dd.ui.log)

The dd.ui.log package provides a basic window for posting messages into a window with a table that provides.

### Overall Design

The design of the logger is broken down into multiple components:

1. LogTableModel – the table model required for powering the logger.
2. LogFilterComponent
   1. The JPanel derivative that presents filter settings and lets the user configure log operation.
3. FilterListener and FilterEvent
   1. The event interface to allow changes to the filter settings to be propagated to other components. It follows the general Java Swing event patterns.
4. LogFrame hosts the actual component with LogMessagesPanel being the component.
5. LogLevel and FilterSetting enumerations for indicating log and filtering settings, respectively.

The class hierarchy is shown in Figure 6. Only general discussion of the log component is given below.

### TableModel

### Supported Logging Levels

The enumeration that governs the logging levels is shown in Figure 9. The available levels allow for the logging to be disabled/ignored if desired.

### Filtering

A common feature of log viewer activity is to allow users to show only events meeting certain criteria. Available filtering levels are shown in Figure 10.



Figure 6 dd.ui.log Classes



Figure 7 dd.ui.log.LogTableModel Class



Figure 8 LogMessagesPanel component



Figure 9 dd.ui.log.LogLevel enumeration



Figure 10 dd.ui.log.FilterSetting Enumeration

### Results

The implementation of the logging with filtering was sucessful. A running instance is shown in Figure 11.s

Z:\mgrabau\Work\DirectedDiffusion\doc\images\LogFrameDemo.tiff

Figure 11 LogFrame in operation

## Plots

### General

Adding plots to the simulation allows the user to view a trace of information as the simulation progresses. Due to the complexity of implementing a robust and capable plotter from the graphics perspective, a third-party library was selected. After considering a small number of options, the JChart2D library was selected. JChart2D offers a well-documented API, robust examples, flexible output of the image and is straightforward to use, thus making it ideal for the DDSim plotting.

DDSim plotting was designed to allow for static and dynamic data collection. In testing, the dynamic model had issues with synchronization – the timer was not permitting it capture all the points being generated. Therefore, the static model is the recommended model and discussed in the remainder of this section.

### Plotting Library

The JChart2D library [4] is an LGPL [5] project that proved effective in producing quality plots. It also provided the key API for this project – it can produce a buffered image object of the plot for output to a file [6].

### dd.ui.plot Hierarchy

The class hierarchy for the dd.ui.plot packet is shown in Figure 12. There are three functional areas to this package:

1. Data Collection: static and dynamic collectors that register lists of nodes to retrieve data for plotting. These classes are the
   1. Specific implementations for collecting variance data: DynamicNodeVarianceCollector and StaticNodeVarianceCollector.
2. JFrame derivatives for displaying plots. The DynamicPlotFrame and StaticPlotFrame classes support static and dynamic data gathering. Both plots show updating/dynamic displays
3. JFileChooser support classes – ImageFilter and derivatives. These classes support the image output of the plots that are generated.

The major functional areas are discussed below.



Figure 12 dd.ui.plot Package

### StaticPlotCollector

The important interactions for the node data collection are shown in Figure 13 and Figure 14. As demonstrated in those figures, the collectData interface is the core of the implementation. The two class diagrams for the StaticNodeDataCollector and StaticNodeVarianceCollector are shown in Figure 15 and Figure 16, respectively. Note the StaticNodeCollector provides most of the functionality required, and there just overrides and constructors in the variance subclass.

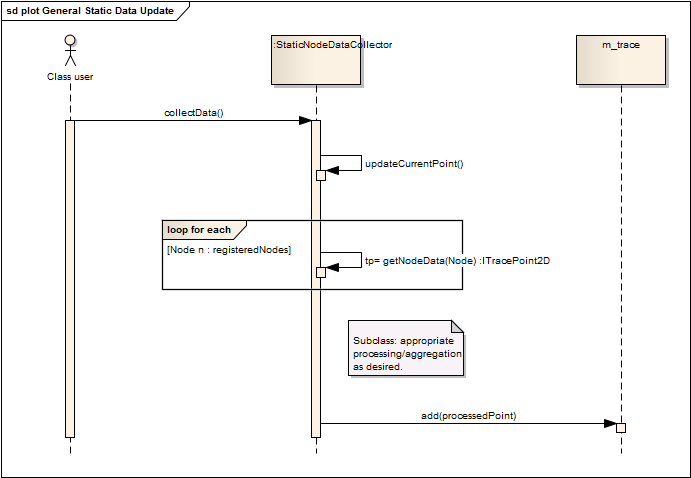


Figure 13 Sequence diagram for collecting static data

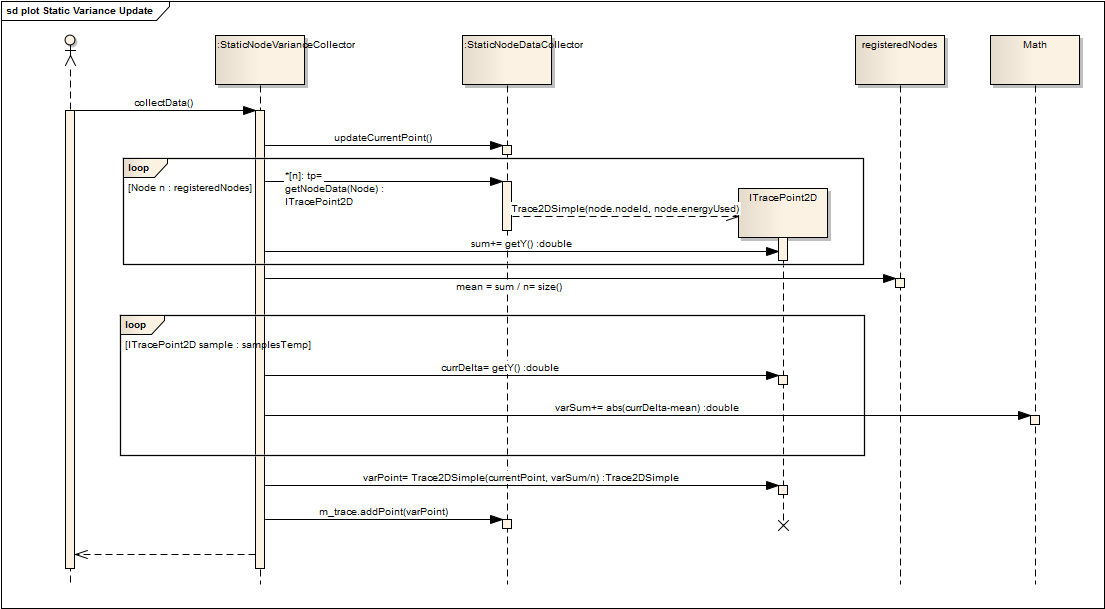


Figure 14 Static variance data gathering



Figure 15 StaticNodeDataCollector class diagram



Figure 16 StaticNodeVarianceCollector

### StaticPlotFrame

The StaticPlotFrame component is an abstract class that provides a platform for implementing plot windows. It contains several hooks for customization during setup.

Using the getTrace() method is not required under ordinary circumstances. It has been included for scenarios outside of perceived usage.

### StaticVariancePlot

The StaticVariancePlot is implemented to support the requirement to produce a variance plot of node energy. A screen shot of the plot running is shown in Figure 18. The user clicks the “Save As Image” button to save an image of the plot. The user clicks the “Export as CSV” to obtain the dataset in a comma separated value format that is both easy to parse and acceptable input to many other applications such Microsoft Excel ™.



Figure 17 StaticNodeVariancePlot class

C:\Users\Mathew Grabau\schoolwork\DirectedDiffusion\doc\images\StaticVariancePlot.tiff

Figure 18 StaticVariancePlot Sample

### Results

The plotting implementation proved successful for the static case. As mentioned previously, the dynamic model suffered from synchronization issues.

## Window Management

### Concept

The idea for the frame/window management stemmed from the scenario where multiple simulator windows are opened, as is demonstrated in the dd.NodeGraphicalTest test. A screenshot of that test in operation is shown in Figure 19.

C:\Users\Mathew Grabau\schoolwork\DirectedDiffusion\doc\images\NodeGraphicalTest.tiff

Figure 19 Multiple Windows Running During NodeGraphicalTest

When multiple windows are opened as shown, the JFrame cannot use the typical method of closing the application (JFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE)) . Programming the windows as such would cause the simulator to close when any individual window was closed.

### Design

The external class for the window manager is SimulationFrameManager, shown in Figure 20. The class is a subclass of WindowAdapter [7], and uses the windowClosing event to detect the window being closed, then check its list of registered. The sequence diagram for handling the closing process is shown in Figure 21.

Registering frames with SimulationFrameManager is done via the registerFrame(JFrame) method. The process of registering frames is shown in Figure 22.

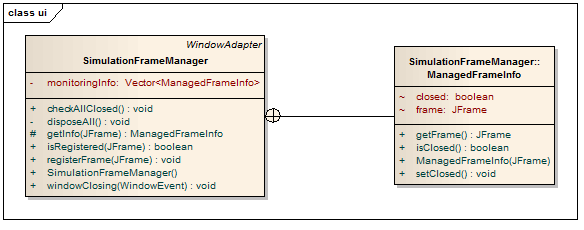


Figure 20 dd.ui.SimulationFrameManager class design

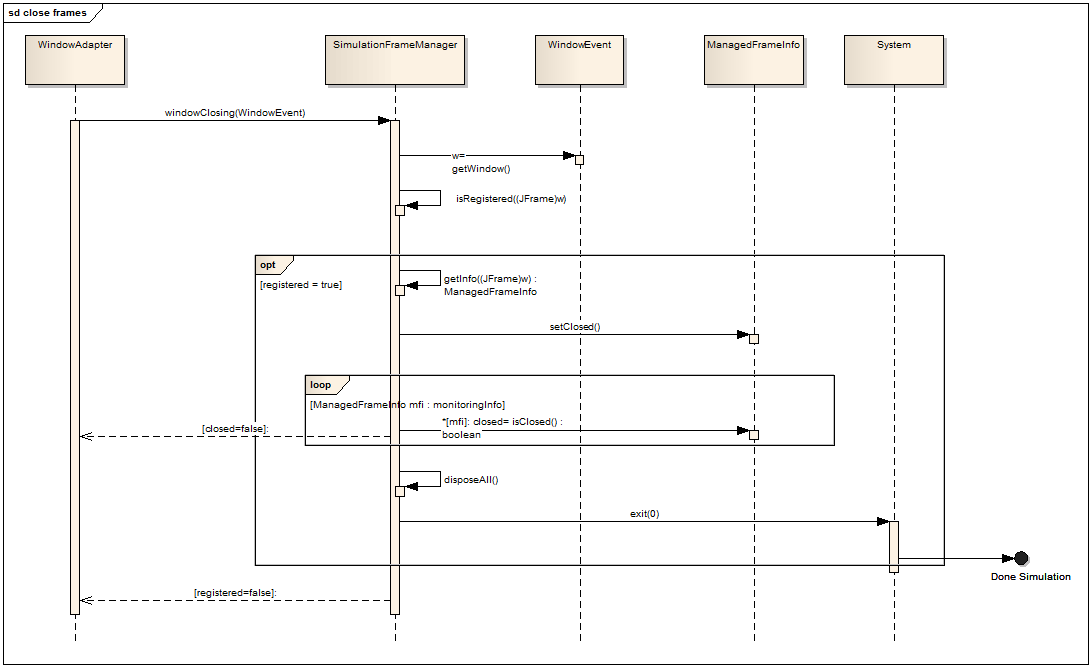


Figure 21 Closing Windows Registered with SimulationFrameManager



Figure 22 Sequence diagram for registering classes with SimulationFrameManager

## Testing Results

The SimulationFrameManager class was successful tested in dd.NodeTestGrapical. The application exits after all the windows are closed.

# Recommendations

The DDSim graphical components presented herewith in provide an excellent opportunity for future work. The activities recommended for further work during design and development activities are:

1. Packet tracer, visualizer and log – would allow the user to look back to the traffic and see a packet’s destination and contents.
2. Node map – map the locations of the nodes on a grid, and allow the user to zoom in and see the activity between the nodes. This alternative would give the DDSim a large boost in user appeal and overall experience.
3. Enhanced plotting – provide a reference implementation of combining multiple data sources into one plot.
4. Extensions to SimulationFrameManager – add means of shutting down simulator without manually closing all windows and controls for single stepping simulations.

# Conclusions

The implemented components present a start to making DDSim a high quality WSN simulator. They add a lot of functionality to the simulator that make it more usable and make it provide a better simulation experience.

With further development, the simulator has great potential to provide a valuable simulation experience for persons interested in studying and modelling Directed Diffusion based WSNs.

# References

|  |  |
| --- | --- |
| [1] | Sparks Systems, "Product Information: Sparx Systems," 2012. [Online]. Available: http://www.sparxsystems.com/products/index.html. [Accessed 2012]. |
| [2] | The Eclipse Foundation, "Eclipse Project," 2012. [Online]. Available: http://www.eclipse.org/projects/project.php?id=eclipse. [Accessed 01 2012]. |
| [3] | Oracle, "AbstractTableModel," 2010. [Online]. Available: http://docs.oracle.com/javase/1.4.2/docs/api/javax/swing/table/AbstractTableModel.html. [Accessed 03 2012]. |
| [4] | A. Westermann, "JChart2D, preciseVisualization of data," 19 06 2010. [Online]. Available: http://jchart2d.sourceforge.net/. [Accessed 03 2012]. |
| [5] | Free Software Foundation Inc., "GNU Lesser General Public License v2.1," 30 01 2012. [Online]. [Accessed 15 03 2012]. |
| [6] | A. Westermann, "Chart2D (JChart2D API documentation)," 2010. [Online]. Available: http://jchart2d.sourceforge.net/docs/javadoc/index.html. [Accessed 03 2012]. |
| [7] | Oracle, "WindowAdapter," 2010. [Online]. Available: http://docs.oracle.com/javase/1.4.2/docs/api/java/awt/event/WindowAdapter.html. [Accessed 2012]. |
| [8] | Oracle, "How to Make Frames (Main Windows)," 2012. [Online]. Available: http://docs.oracle.com/javase/tutorial/uiswing/components/frame.html. [Accessed 03 2012]. |