2017

Graphical User Interface (GUI) Development for an Optical Communication Simulator

Sratsav Ram Shrestha
Minnesota State University, Mankato

Graphical User Interface (GUI) Development for an Optical Communication Simulator

By

Sratsav Ram Shrestha

A Thesis Submitted in Partial Fulfillment of the Requirements for Master of Science

In

Electrical Engineering

Minnesota State University

Mankato, Minnesota

November 2017

Date: 11/9/2017

Title: Graphical User Interface (GUI) Development for an Optical Communication Simulator

Student's Name: Sratsav Ram Shrestha

This thesis has been examined and approved by the following members of the student's committee.

Dr. Qun Zhang (Chairperson)

Dr. Nannan He

Dr. Puteri S. Megat Hamari

ACKNOWLEDGEMENT

I would like to thank my advisor Dr. Qun Zhang for his enormous support and guidance during my research and thesis work. I greatly appreciate Dr. Zhang for allowing me to be part of his research team. I have learnt a lot working under him. His dedication to his work has greatly influenced me and motivated me to work hard in every aspect of my life.

I would also like to thank my parents Shree Ram Shrestha and Meera Shrestha for their continuous encouragement and love.

I would like to dedicate my thesis to my Father

ABSTRACT

Modeling and simulation tools have been an integral part of engineering world for a long time. Various Electronic Design Automation (EDA) tools have been extensively used in various industries and research to evaluate the performance of electronic systems. The advancement of such design tools also has influenced the optical communication sector such that there has been a continuous progress on the Photonic Design Automation (PDA) tools. Currently, many software for simulating optical communications are available. However, they are very expensive and conceal the information on how components are modeled. To avoid these constraints, we developed our own PDA software for optical communication. This thesis delves into the development of Graphical Unser Interface (GUI) of our software. The studied GUI software conforms to the feature of standard simulation software and assists the users to perform a system-level simulation of fiber optic communication. The developed GUI allows the users to design their layout, run the simulation and view the results in the form of data or plot. The GUI is explained with respect to the graphical layout and the interactive features of the components. The detailed structure is described along with the Java library used to build them. The interactive aspects of GUI are investigated, for adding the hierarchical feature to our GUI software. In addition, a plotting tool is created for the GUI. The thesis provides comprehensive information on working principle of GUI for simulation software and describes the addition of plotting tool and hierarchical design in detail.

TABLE OF CONTENTS

Chapter 1

Introduction

1.1 System models

System models are essential engineering tools to represent real systems and to assist engineers to analyze systems performance and design systems. They remove the necessity of the expensive and time-consuming tests on the hardware to obtain optimal design results. A good model should possess most if not all the salient features of the real system. Thus, a model should be a very close approximation of the real system, such that results of the evaluation performed on the model will be very close to the true reflection of the behavior of the real system. The application of the models not only assist to infer the properties of a system (so system performance can be calculated or simulated) but also aid to predict the change in properties of the system with different configurations (so system can be designed and optimized).

Generally, the system models are a mathematical representation of a physical object, process or system. The mathematical models are further classified in to deterministic and stochastic according to the nature of the system input and output [1]. A deterministic model defines one output for each possible input. If the initial state and input of the model are known, then the model produces one behavior. Models for example with ordinary and partial differential equations are deterministic. Similarly, synchronous digital circuit model and single threaded imperative programs are also deterministic. The definitive analysis that

we can derive from the determinism is a valuable asset for modeling which enables the absolute assertion of system behavior and produces a reliable design of very complex system [2]. Compared to the stochastic models, deterministic models are able to demand significantly less computational resources.

In the real physical world, many engineering problems showcase random behavior with respect to time and/or space [11]. For example, if we are building a web server, a request may come in at random times and may come as burst requests: such a system requires a stochastic modeling to represent their true behavior. In a stochastic simulation, ‘‘random variables’’ are included in the model to represent the influence of factors that are unpredictable or unknown to the end users in the simulation [11]. Each unit, process, events, or their parameters are initiated randomly using random numbers [1]. Also, different runs of system initiated with different random seeds will provide different output. Usually, many runs will be needed to carry out a meaningful statistical study of systems performance, or to probe system optimization [1, 14].

Similarly, models can be differentiated with respect to time. The model that doesn’t consider time is a static model and the model representing time-varying interactions among variable is a dynamic model. A static model often involves drawing random samples to generate statistical outcome [14]. During the simulation of such model, no time axis is used. In contrast, the dynamic model includes the passage of time. The state changes are observed over time. A clock mechanism moves forward in time and state variables are updated as time advances [14]. Here we intend to build system models for optical communication systems which can be used for simulations and design optimization. The

model pertaining to optical communication systems involves both deterministic and random effects and is dynamic.

For example, the model Mach-Zander modulator (MZM) in our software takes two inputs namely electrical driving signal and laser CW signal. The MZM is illustrated in Figure 1.1. When the laser signal is supplied to MZM, the output, amplitude, and phase of the laser, can be controlled by changing the driving electric voltage. The laser output in terms of electric field can be shown by the following equation [26],

$$E_{out}(t) = \frac{1}{2} E_{in}(t)\left(e^{j\emptyset_1 t} + e^{j\emptyset_2 t}\right) \quad (1)$$

, where $\emptyset_1(t) = \frac{u_1(t)}{V_\pi}\pi$ and $\emptyset_2(t) = \frac{u_2(t)}{V_\pi}\pi$. Here, $E_{in}(t)$ represents the input electrical field of laser light, $u_1(t)$ and $u_2(t)$ represents the integral of the applied voltage on the two arms of the MZM to model the electrical driving signal. Since the model follows the mathematical expression we can clearly state for particular inputs $u_1(t)$, $u_2(t)$ and $E_{in}(t)$ there will always be a unique output$E_{out}(t)$. This makes the model a deterministic model. Moreover, because the modulator system is a time-varying dynamic system, equation (1) is a dynamic equation.

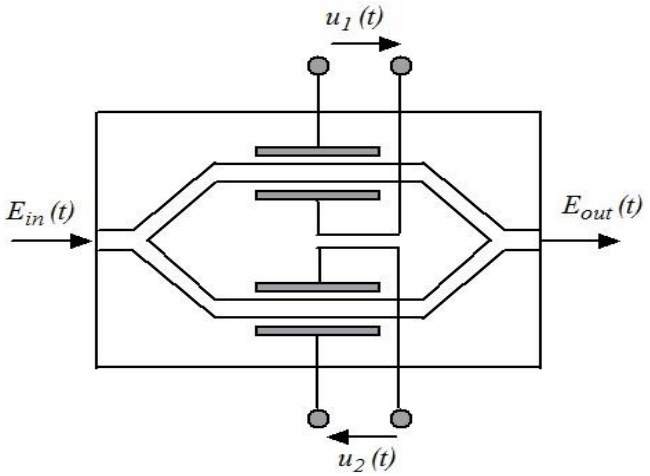


Figure 1.1: Schematic of MZM system

To generate the electrical driving signal, in our optical communication simulation software we have a “PRBS” component. PRBS stands for pseudorandom binary sequence and is used to generate a randomized bit stream. The algorithm used to generate the binary sequence is deterministic as these sequences are generated by implementing linear feedback shift registers [32]. But the generated long bit sequence is used to model “random” bit sequences. The system model with the stochastic source included is thus a stochastic model. Some common sequence generating polynomials are $PRBS31 = x_{31} + x_{28} + 1$, $PRBS23 = x_{23} + x_{18} + 1$ etc. Similarly, our software also uses a model named “BERT” to find the bit error rate. The bit error rate is calculated by comparing the transmitted sequence of bits to the received bits and the counting the number of errors. The ratio of how many bits received in error over the number of total bits received is the BER [33]. Here, we can use the PRBS as our input which is known to us, and then compare it with the output resulting from various processing. Here for specific input and output data, the result of BERT is deterministic. Also, all the above mentioned are dependent on time. In all, our software consists of various optical and electrical components that are modeled in a deterministic and dynamic framework, conditioning on a known source.

Every model is constructed within some modeling framework. The framework defines the syntax and the semantics of the model [6]. Syntax refers to how the model is expressed either in a form of statement, expression, program units or physical form, whereas semantic refers to the meaning represented by the expression, statements, program unit or physical form. For example, “while (<Boolean _expr) <statement> “, refers to a syntax of a programming language like C and Java. Here, the semantic of this statement form is that when the current value of the expression is true, the following statement keeps on

executing. For a system modeling, general syntaxes used are block diagram, bubble and arc diagram, imperative programs and arithmetic expression. Our simulation software mainly works with a syntax of block diagram as shown in Figure 1.2. Each block diagram represents a unique optical, electronic or optoelectronic component, for example, we can use a "PRBS" block diagram to represent random number generator, an "MZM" block diagram to represent the Mach-Zehnder modulator, and so on. The block diagram syntax can have many distinct semantics. The connections between blocks can represent various kind of interaction between components such as asynchronous communication, synchronous dataflow, discrete/continuous communication etc. In our software, the components are connected using an arrow and these connections between the components via arrows represents static data flow/synchronous dataflow between the components. In static dataflow, the components are executed when they receive the required input [5]. Here in our software, when the components to the left of the arrow is executed, it makes the resulting value available to the components to the right of the arrow, which is then used by the following components for its execution. In Figure 1.2 component "A" executes first such that its output then triggers the component "B" and this output is utilized by "B" for its execution.

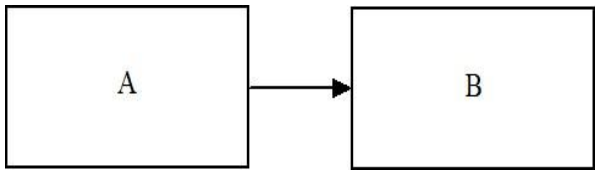


Figure 1.2: Block Diagram Syntax used by our software

1.2 Electronic Design Automation (EDA) and Photonic Design Automation (PDA)

In the electrical and computer engineering (ECE) discipline, abstractions are used to simplify models and design processes [28]. One key useful science in the ECE is electromagnetics. The related first principles are summarized as Maxwell's equations. The following Maxwell equations are the starting points of most electric models.

$$\nabla \cdot \vec{D} = \rho_V \qquad \text{(Gauss's law for electricity)} \tag{2}$$

$$\nabla \cdot \vec{B} = 0 \qquad \text{(Gauss's law for magnetism)} \tag{3}$$

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \qquad \text{(Faraday's law)} \tag{4}$$

$$\nabla \times \vec{H} = \frac{\partial \vec{D}}{\partial t} + \vec{J} \qquad \text{(Ampere's law)} \tag{5}$$

In the above equations, $\vec{H}$ is the magnetic field, $\vec{D}$ is electric flux density, $\vec{B}$ is magnetic flux density, ρ_V is charge density, $\vec{E}$ is electric filed, $\vec{J}$ is current density. These differential equations are complex for any model. However, under certain assumptions for many real systems and devices, we can derive instead simpler equations from the complex Maxwell's equations, to model the systems easily that provides accurate enough approximation of system behaviors. Let us consider an example of a lightbulb connected to a battery via a pair of cables as shown in Figure 1.3. If we are interested in finding the current going through the light bulb, we can follow the difficult path of using Maxwell's equation to derive the amount of current by a careful analysis of physical properties of the bulb, the battery and the cables, which is a cumbersome process. Using Maxwell's equation, we can solve for $\vec{E}$ and $\overrightarrow{H}$ for any time at any location. But we usually do not need this much information and therefore can avoid all of this by restricting our field of interest to a

discipline like investigating only the net current flowing through the bulb. This way, we can ignore the internal properties of the bulb and represent the bulb as a discrete element.

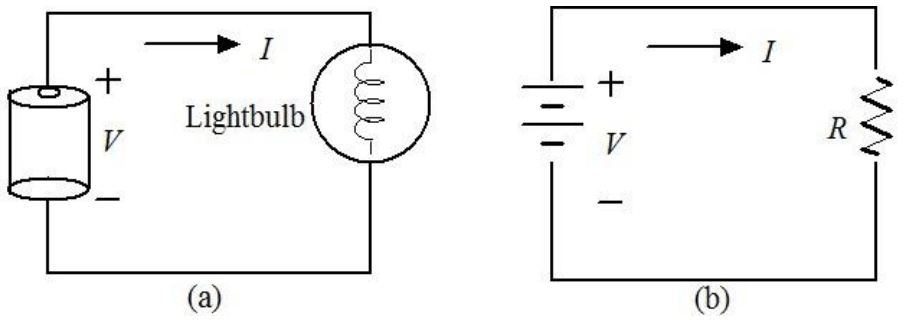


Figure 1.3: Modeling of light bulb system to a lumped element

Furthermore, we can replace the bulb with a discrete element known as the resistor and define the resistance R to be the ratio of the voltage applied and the resulting current through it i.e. $R = \frac{V}{I}$. Likewise, the voltage is the property of the battery which may be of our major concern. Now, ignoring the internal resistance of the battery we can lump the battery into a discrete element by a constant voltage V. We can make this assertion by ignoring the internal properties of battery such as the distribution of the electrical file. This collection of constraints that underlie the lumped circuit abstraction results in a simplification, which allows us to focus on specifically those properties that are relevant to us. For this abstraction to hold, we need to make sure V and I are well defined for the element. The voltage, the current, and the resistance are defined for an element only under certain constraints that we collectively call the lumped matter discipline (LMD) [28].

The lumped matter discipline imposes three constraints on how we choose lumped circuit elements [28]:

1. The rate of change of magnetic flux linked with any closed loop outside an element must be zero for all time for the lumped elements we choose. In other

words, $\frac{\partial \phi_B}{\partial t} = 0$ through any closed path outside the element for the chosen lumped elements boundaries.

2. The lumped element boundaries must adhere to the rule that there is no total time varying charge within the element for all time. In other words, $\frac{\partial Q}{\partial t} = 0$ (Q is the total charge within the element) for the chosen lumped element boundaries.

3. The period of the signal in operation must be much larger than the propagation delay of electromagnetic waves across the lumped elements.

In the above section, we discussed how Maxwell equations can be simplified under some assumptions. Now in this section, we deal with more details on the implementation of Maxwell equations to derive various useful abstractions such as Ohm's law and Kirchhoff's law. Let us take Gauss's law for electricity with some considerations such that we can relate the volume charge density ρ_V and the electric current density $\vec{J}$ of Maxwell's equations. Let us take a tube with cross surface area ΔS, and the charge flow with velocity v inside the tube. Then by the definition of the flux, we obtain

$$\vec{J} = \frac{\Delta Q}{\Delta S \Delta t} = \frac{\Delta I}{\Delta S} \tag{6}$$

, where ΔQ total charge within the enclosed volume is, Δt is the rate of change of time. Furthermore, we know that ΔQ can be defined as follows:

$$\Delta Q = \rho_V \, \Delta V = \rho_V \, \Delta S v \, \Delta t \tag{7}$$

Now, substituting (7) in (6) we get the following written in the vector form,

$$\vec{J} = \rho_V \vec{v} \tag{8}$$

Furthermore, the charge velocity $\vec{v}$ in a conductor is dependent on the charge mobility i.e.

$$\vec{v_e} = -\mu_e \vec{E} \tag{9}$$

$$\overrightarrow{v_h} = \mu_h \vec{E} \tag{10}$$

, where $\overrightarrow{v_e}$ is the velocity of electron of the conductor, μ_e is the charge mobility constant of the electron, $v_h$is velocity of the hole, and μ_h is the charge mobility constant of holes [34].

Now, we can rewrite equation (8) as follows:

$$\vec{J} = \rho_e \overrightarrow{v_e} + \rho_p \overrightarrow{v_h} \tag{11}$$

Further, using equation 9, 10 and 11 the current density $\vec{J}$ can be written as:

$$\vec{J} = (-\rho_e \mu_e + \rho_h \mu_h)\vec{E} \tag{12}$$

We know that the specific conductivity σ depends on the free-charge density and its mobility i.e. [34].

$$\sigma = -\rho_e \mu_e + \rho_h \mu_h \tag{13}$$

Therefore, using equation (13) in (12) we get,

$$\vec{J} = \sigma \vec{E} \tag{14}$$

The equation (14) is the law of electricity called Ohm's law [27]. When the electric field $\overrightarrow{E}$ is uniform and oriented along the length of the conductor (L), we can compute the voltage as

$$V = \overrightarrow{E} L \tag{15}$$

Furthermore, the current density is defined as

$$\vec{J} = \frac{I}{\Delta S} \tag{16}$$

Now, substituting $\vec{J}$ and $\vec{E}$ in equation (9), we obtain the most familiar form of Ohm's law

$$I = \frac{V}{R}, \; R = \frac{L}{\sigma \Delta S} \tag{17}$$

With the above abstraction, we can now use the simple algebraic equation of Ohm's law, instead of cumbersome Gauss's law, to model resistive systems.

Similarly, we can resort to lumped matter discipline to derive Kirchhoff's laws from Maxwell's equations. Here, we are interested in deriving voltages across and the currents through each element in the circuit in Figure 1.4.

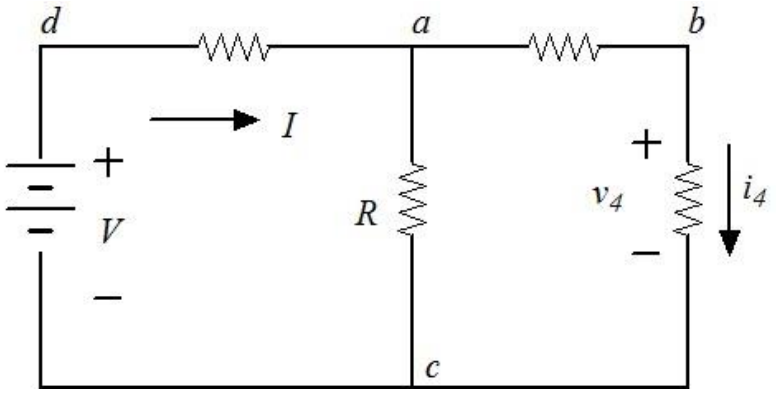


Figure 1.4: A simple resistive network

We can use Maxwell's Equations and the related continuity equation to solve the circuit as shown below:

$$\oint \vec{E} \cdot dl = -\frac{\partial \phi_B}{\partial t} \tag{18}$$

$$\oiint \vec{J} \cdot dS = -\frac{\partial Q}{\partial t} \tag{19}$$

, where ϕ_B is the magnetic flux.

In the previous section, according to the lumped matter discipline, we have constrained ourselves to the circuit domain where $\frac{\partial \phi_B}{\partial t} = 0$ for closed circuit loops and $\frac{\partial Q}{\partial t} = 0$ for circuit nodes. With this assumption equations (18) and (19) can be re-written as:

$$\oint \vec{E} \cdot dl = 0 \tag{20}$$

$$\oiint \vec{J} \cdot dS = 0 \tag{21}$$

Here, under lumped matter discipline, equation (20) states the line integral of the field around any closed path must be equal to 0, where equation (21) states that surface integral of the current over any surface must be 0.

Now, we apply equation (20) to the closed loop defined by the three circuit edges $a \rightarrow b$, $b \rightarrow c$ and $c \rightarrow a$, as depicted in Figure 1.5, we obtain

$$\oint \vec{E} \cdot dl = \int_a^b \vec{E} \cdot dl + \int_b^c \vec{E} \cdot dl + \int_c^a \vec{E} \cdot dl = 0 \tag{22}$$

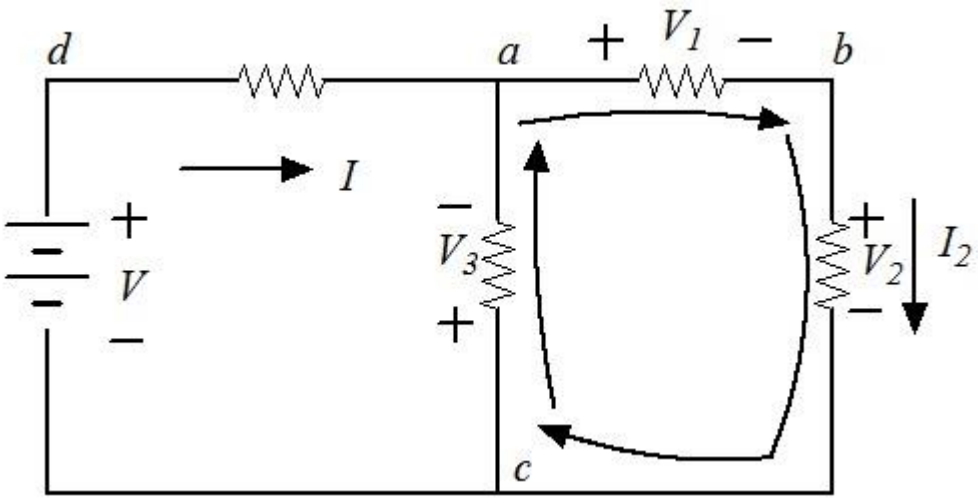


Figure 1.5: The line integral over a closed loop in the network

We know that the potential difference V_{xy} across the xy terminals of elements is given by

$$V_{xy} = \int_x^y \vec{E} \cdot dl \tag{23}$$

Using equation (22) and (23) we can write

$$\int_a^b \vec{E} \cdot dl + \int_b^c \vec{E} \cdot dl + \int_c^a \vec{E} \cdot dl = V_1 + V_2 + V_3 = 0 \tag{24}$$

In other words, the algebraic sum of voltage along any closed path in the circuit must equal 0.

We will now look at the derivation of Kirchhoff's current law. We will apply equation (21) to the closed box-like surface depicted in Figure 1.6. We notice that there are currents flowing only through surface S_a, S_b andS_c. Therefore,

$$\oiint \vec{J} \cdot dS = \oiint_{S_a} \vec{J} \cdot dS + \oiint_{S_b} \vec{J} \cdot dS + \oiint_{S_c} \vec{J} \cdot dS = 0 \tag{25}$$

Since, we have confined our current to the wires entering the three surfaces we obtain

$$\oiint_{S_a} \vec{J} \cdot dS + \oiint_{S_b} \vec{J} \cdot dS + \oiint_{S_c} \vec{J} \cdot dS = -I_a - I_b - I_c = 0 \tag{26}$$

In other words, the algebraic sum of the currents flowing into any closed surface must be zero.

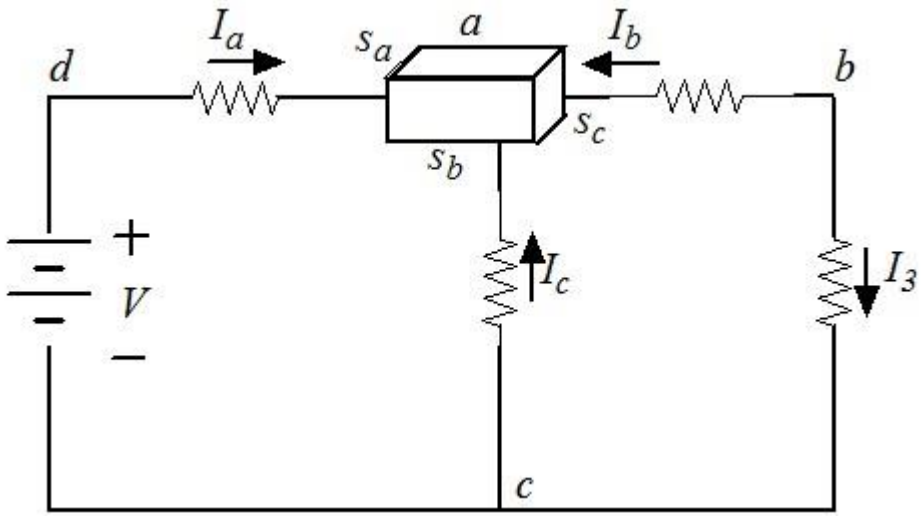


Figure 1.6: The surface integral over a closed surface in the network

Clearly, this abstraction of the complex differential equation of Maxwell to the algebraic sum of voltage and current is easier to model. These simple algebraic equations are what most of the electric circuit simulation tools are based on to simulate complex electric circuit system.

This kind of rendition of complex equations to obtain simpler models has been extensively used from the beginning of the development of simulation tools. Circuit

simulators development started during late 1960's and early 70's. Two groups contributed significantly to the development of the modern current simulator. The ASTAP group at IBM developed many of the numerical methods used. And the SPICE (Simulation Program with Integrated Circuit Emphasis*)* group of the University of California at Berkeley developed and propagated the de facto standard simulator [35]. SPICE was designed to be used to simulate integrated circuits. In the late 70's and early 80's most industries built their own SPICE tools, as from their views the in-house tools provided them the competitive edge in the market. However, the commercial simulators began to surpass the internally developed simulators in terms of capabilities and performance in late 80's and early 90's. In the late 80's Berkeley upgraded SPICE by releasing SPICE3, which had a new architecture that made it considerably easier to add new component models and was written in C [35]. At the same time, Berkeley also released a new type of circuit simulator called Spectre. The computation of the steady-state solution of nonlinear circuits in the frequency domain was included in Spectre, and its major focus was on microwave circuits. Spectre was later bought by Hewlett-Packard, where it was named as Microwave Nonlinear Simulator. The use of harmonic balance algorithm was also replaced by transient analysis algorithms in this circuit simulator [35].

In addition to the simulation tool development, in the late 1970's and early 1980's there was another significant change in electronics design. The shift was towards use and reuse of pre-characterized building blocks or cell as opposed to designing each transistor from scratch for each new chip. This technique later became known as standard cell based design, and it became the standard way used to build application-specific integrated circuits

(ASICs). During same time, methodology to design and verify integrated circuits was also evolving, with the availability of general purpose computing and engineering workstations in conjunction with the advent of computer-aided design (CAD) tools. This later turned into an entire industry now known as electronic design automation (EDA). EDA can generally be referred to as a tool that integrates software and hardware aspect of any electronic system. EDA consists of a collection of methodologies, algorithms, and tools, which assist in the automation of design, verification, and testing of electronic systems. It embodies a general methodology that seeks to successively refine high level to low level detailed physical implementation for designs ranging from integrated circuits to digital communication system and other various electronic systems [25]. It involves modeling, synthesis and verification at every level of abstraction [25]. The usage of EDA is universal as its implementation occurs in all kinds of different industries such as IC fabrication, Digital Communication, control system, photonics, automotive etc. Here we are mostly concerned about the use of EDA tools for photonic design, and the newly emerged photonic design automation (PDA) tools which are developed specifically with a focus on the photonics area, under very similar concept as of EDA.

1.2.1 Photonic Design Automation (PDA)

The integrated photonics research started in the late 1980s and in the early 1990s [36]. Materials like polymers, doped glass, and dielectric thin film materials like silicon oxide and nitride were key areas in photonics at that time. A new emerging field was integrated optics, an area of studying light wave devices or planar light wave circuits (PLC). Because of these research activities, a need for proper design software emerged focusing on the simulation of mostly passive optical structures at the micrometer scale and the mask layout

for the actual fabrication of the structures and devices. This was reflected in the start of an annual conference of Optical Waveguide Theory and Numerical modeling (OWTNM) in 1992, and the first commercial activities in design and simulation tools, e.g., BBV (today phoenix software) in the Netherlands in 1991 and Photon Design in the United Kingdom in 1992.

Since the early 1990s, photonic IC design software has gradually developed to current status. Due to unavailability of the advanced design kit, recently European companies and institution have worked together to set up a Photonic Design Automation (PDA) tool-set [30]. The PDA includes circuit simulators, mask layout tools, measurement databases, and design rule checkers. In addition, they include physical modeling tools such as mode solvers and propagation simulators.

The design of integrated photonics mostly has been based on bottom-up approach, starting with the fabrication technology and materials and taking these as starting points to develop integrated photonic devices [36]. However, with the introduction of more standardized and generic fabrication process since 2005 and the resulting creation of process design kit (PDKs, also called physical design kits), a mixed design approach has evolved in which a group of designers develop the contents of the PDKs and a second group of designers uses these PDKs in a top-down manner starting from the system or circuit level. Our developed software in the thesis work is a part of the PDK that deals with the system level design of optoelectronics components such as the laser, active phase shifters, modulators, detectors etc.

The modeling of these optoelectronics components is more challenging than passive electrical or optical components due to the material involved and associated additional

process steps. The manufacturing of these components is complicated. These optoelectronics components have different operation principles that are highly dependent upon the process, technology, and physical geometry. Unlike electrical circuit simulators, photonic circuit simulators must consider the complex nature of light that includes the optical signal's polarization, phase, and wavelength. The optical simulation and analysis of PICs are particularly challenging because it involves bidirectional, multichannel and even multimode propagation of light and waveguide connections between consecutive components require specialized treatment unlike what is done for electrical traces. Therefore, a basic component level photonic circuit simulators must rely on proper compact models, calibrated for a foundry process which accurately represents the optical and electrical response of these components in time and frequency domain. A key challenge is how to develop and calibrate compact models for these components [36].

Several tools are available currently to design silicon photonic circuits that consists of many components. All these components are desired to be built with simple models. The simulations thus focus on the functionality and performance of the entire circuit. Many methods are available for implementing compact models. However, some simulating devices and sub-systems the models will have the analytic solutions (e.g., 1-dimensional structures for slab waveguide), but some systems are physically too large such that they are not effectively handled by numerical methods. In the latter, phenomenological models, such as parameterized waveguides, can be used to simplify the simulation of larger circuits. Desired simulations include the frequency domain response of the system (optical filter characterization) and time-domain simulations (transient, eye diagram, bit error rate) [29].

The optical circuit modeling can be performed on various platforms [29]. The design can be done in MATLAB by building simple circuit models or using open-source solutions such as the Caphe from University of Ghent which is based on Python. We can also use EDA environment to design optical models like the method followed by Luxtera of using VerilogA. There are also various commercial tools one can use, for example, using Advanced Simulator for Photonic Integrated Circuits (ASPIC) to simulate the steady-state response; using Photon Design PICWave and Optiwave OptiSystem for time domain modeling, and using Synopsys RSoft OptiSim and ModeSYS, VPIsystems VPItransmissionMaker, Lumerical Interconnect and Caphe for both the time-domain and frequency-domain circuit response [29].

These commercial PDA tools are used in practical system design, and they could be very helpful for classroom teaching because they reveal correct design characteristics and correct trends of design results. Moreover, unlike the industrial in-house tools that usually have text interfaces, the commercial tools feature a rich GUI so that they are easier to use for interactive teaching and easier for students to learn. However, these tools are very expensive and are not open source. For example, for each of the VPI software packages such as VPIphotonics' TransmissionMaker, the one year license fee costs USD 7000 even for academic use. Moreover, the commercial tools are all black box models, and users cannot gain any insight of how various components are modeled. Also, some new system models and design techniques are not included in these commercial tools. To overcome these constraints of commercial tools, we developed our own GUI based software using free integrated development environments (IDEs). Our tool conforms to the current industrial design practice that includes state of the art design flows, models, and design

techniques. Currently, our software tool has an emphasis on system level simulation of fiber optic communication. The software tools can be used for research and teaching purpose of fiber optical communication. Students or researcher can perform optical system design by using various components available in the software library and evaluate the performance of the system to obtain the optimized solution. The software is based on client/server model [37] and has been developed exclusively using free software development environments. The GUI of the software is developed using Java which provides the users platform to drag and drop components from the library, connect them with arrows, build their systems, run the simulation and observe the results. In addition, the users can change the component parameters as per their design requirements and input the simulation control flags. The software's server is based on C/C++ which is the backend engine of the software performing complex mathematical calculations. Each component's mathematical model is designed in C/C++ and the interaction between components are also dealt with it. These mathematical component models such as information sources, optical modulators, optical fiber, optical receiver, and signal processing blocks have been developed with high accuracy. Furthermore, once the system is designed it can be saved as Extensible Markup Language (XML) [37] which can be easily reloaded to the design window by the users.

We use Java 1.7.0_71 to develop our GUI component on Eclipse IDE to develop our client models, and use Microsoft Visual Studio 2005 Express to develop our server component. The framework of our developed software tool consists of three abstracted layers as shown in Figure 1.7. The operation platform layers represent the front-end operation of the software. This layer consists of GUI interface where system layout can be

developed, component parameters can be varied and the simulation results can be viewed as a plot. The management and simulation layer represents functions of the server codes. In this layer, all the parameters of the component are verified with their type and range. Furthermore, the connection of the system is checked and the error message is shown if the execution order of components is incorrect. After error checking of the system layout and the parameters value the server performs the mathematical calculation, yielding results as an output data file.

The interaction between the top two layers is shown in Figure 1.7 by the double directed arrow between these layers. The bottom layer is the data maintenance layer and it deals with the file management and recording aspect of the software. All the storing and handling of parameter values, system layout configuration is dealt in this layer such that system can be saved and reloaded with ease. All these layers work in tandem to provide a complete platform for the simulation of an optical communications system. Although currently our software is dedicated towards simulation of optical communication system, this can be extended to model more general cyber physical systems.

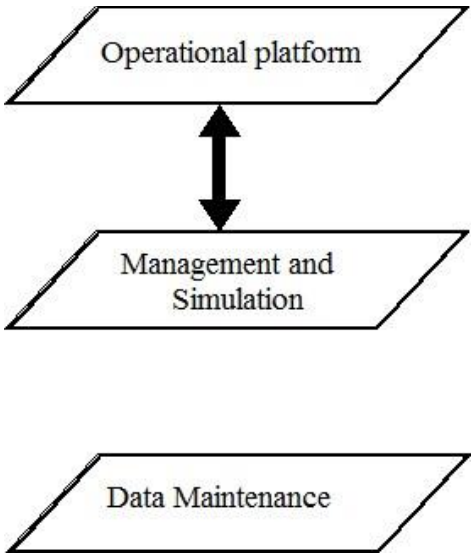


Figure 1.7: A simple schematic to illustrate our software framework

1.3 Cyber-Physical System

Cyber-Physical Systems (CPS) are integrations of computation, networking, and physical processes [6]. It combines cyber systems (computational systems such as microprocessors and digital communication networks) with other physical systems (electromechanical, chemical, structural, and biological systems) [6]. Embedded computers and networks monitor and control the physical processes, with feedback loops where physical processes affect computations and vice versa. The economic and societal potential of such systems is vastly greater than what has been realized, and major investments are being made worldwide to develop the technology. CPS integrates the dynamics of the physical processes with those of the software and networking, providing abstractions and modeling, design, and analysis techniques for the integrated system [38].

Currently, hot fields such as Internet of Things (IoT), Industry 4.0, the Industrial Internet, Machine-to-Machine (M2M), the Internet of Everything, TSensors (trillion sensors), and the Fog (like the Cloud, but closer to the ground) are all kinds of CPS. CPS is a broad area since it focuses on the comprehensive vision of incorporating cyber and physical worlds and involves many types of specific technologies, like in Internet of things or Industry 4.0. Application areas of CPS include medical devices and systems, aerospace systems, transportation vehicles and intelligent highways, defense systems, robotic systems, process control, factory automation, building and environmental control and smart spaces. CPS interact with the physical world and must operate dependably, safely, securely, and efficiently and in real-time [31].

Figure 1.8 assist a conceptual understanding of CPS and their applications along with the attributes and the needs for successful CPS. The figure shows one major feature of the

CPS being feedback systems, where the output is tracked and evaluated with input to obtain the optimized output. The effect of human, environment and economy could also be added into the feedback loop to make the model closer to real system. The system could consist of network/distributed, adaptive/predictive, intelligent and real-time feedback loop. Furthermore, CPS have various physical devices interacting with each other through a network either locally or via Internet. Due to the widespread use of cloud computing, it is possible that many CPS have a presence in the cloud. So, they require efficient cybersecurity platforms to prevent the system being vulnerable to malicious web attack. Cybersecurity becomes critical especially to systems for which malfunction could lead to disastrous effect. These safety critical systems include smart grid, power plant, etc. The design procedure of these system starts with the modeling, and usually, the model is heterogeneous so that it can be used to evaluate the performance of CPS across a variety of computational domains. In addition, the modeling tool should have the functionality to model the interoperability among various systems along with the network structure. Time synchronization across heterogeneous systems should also be included in the model. Finally, the modeling tool should be capable of integrating heterogeneous systems. Note that with further enhancement, our developed software can also be used to model the CPS with required functionality added to the existing software. The long-term goal is to make the software suitable not only for modeling communication systems but also to model complex CPS.

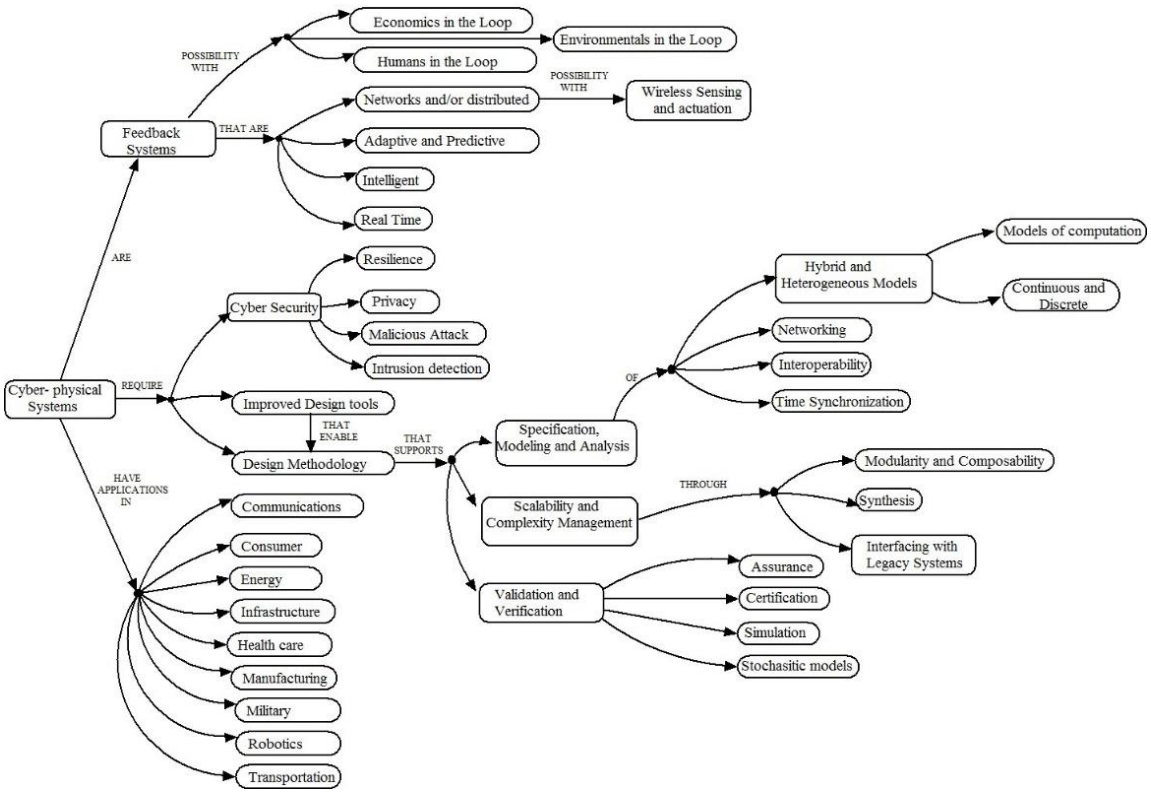


Figure 1.8: Concept Map of CPS

1.4 Tasks and thesis organization

In above sections, we have briefly introduced the background of thesis work. The software currently has a fully functional GUI incorporated with plotting tools. Furthermore, the backend code for all the components is also fully implemented in C/C++. The current version of the software can simulate a complex optical communication system. However, there are several tasks required to update the software. One task is to re-develop the plotting tools since the portion of the plotting tools is out-of-date, due to the fact that the library used in old software expires. Another task is to add more classes and redesign the software so that it has the capability of modeling hierarchical systems. This is another main task of the thesis. The third task is adding the capability of explicitly modeling feedback systems, which will be left for future work.

The thesis will be focused on two issues. The first issue is pertaining to the development of GUI and the other is the hierarchical model design. The details about the functionality of the GUI and its development procedure will be explained in chapter 2. The chapter 2 will provide details about each component of GUI interface, including the theory behind the construction of the components. We will also discuss the usage of GUI interface and the features of our software in detail. The chapter 3 will consist of the work we performed to update the software on two fronts. The first half of chapter 3 we will explain how we added the plotting tools in our software and the second half will be dedicated to the result of work on the hierarchical design. The functionality of hierarchical design its importance and the benefits of the usage of it in our software will be dealt in detail in chapter 3. A brief overview of the algorithm will also be discussed in this section. The chapter 4 will summarize all the preceding chapter and will conclude the thesis. In addition, the need of feedback features for simulation of an optical communication system and possible algorithm to achieve it will be discussed as a future work in chapter 4.

Chapter 2

Graphical User Interface of the Software

In this chapter, we will explain about the graphical user interface (GUI) of our software. First, we will briefly provide an overview of the software and then deal with many key components of GUI in details. Here, the explanation of GUI is performed with respect to its layout and the interactive features. Our GUI development is done in Java using its swing components, therefore we will explain some swing components of Java that were used in the development of GUI before presenting the detailed structure of our software. The main contribution of the thesis work includes the migration of the GUI software to the new computer platform, and detailed documentation of the software. In this Chapter, we summarize the developed documentation to clearly explain the developed GUI in detail.

2.1 Introduction to Software Graphical User Interface (GUI)

GUI is an essential feature of any software tools. The goal of GUI is to enhance the users experience by making the interaction between users and software easy. Our objective is also to provide the users-friendly working environment to our software users. The GUI of our software is based on drag and drop framework. The four major components of our GUI are the main window where system layout is created, component library, component lists, and toolbar (containing arrow, delete, global variable, and loop variable buttons). Figure 2.1 illustrates the GUI of our software with a simple design layout.

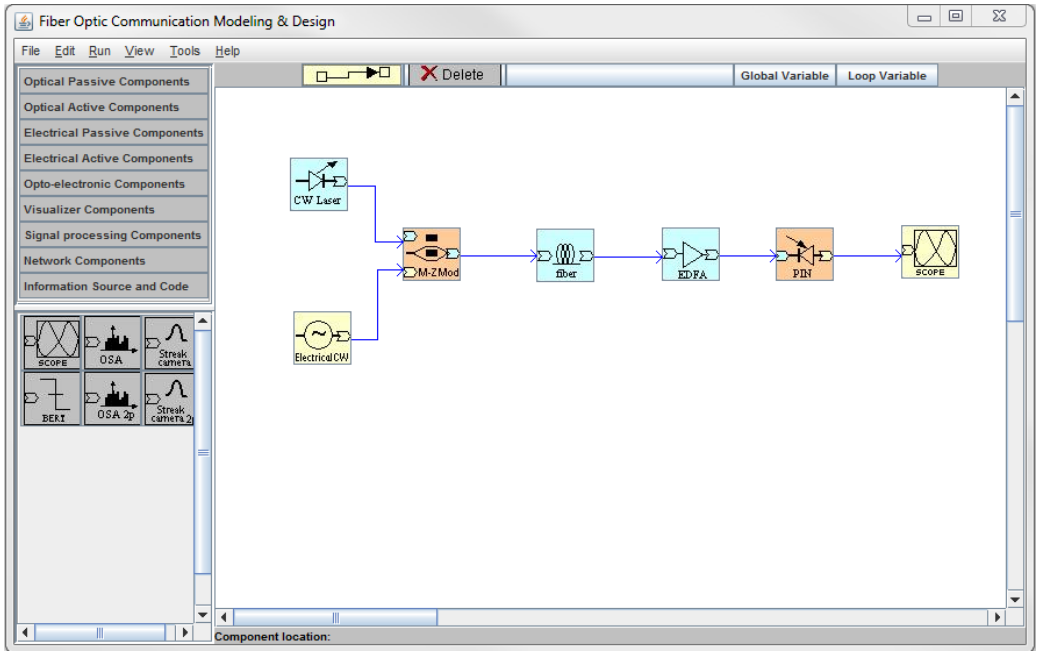


Figure 2.1: The GUI main window with a system layout

As shown in Figure 2.1, the component library is in the top left panel. The library is grouped into nine categories based on the functionality of the components. Each category when clicked will showcase all the components related to the group: these components are displayed in the bottom left panel residing under the top left panel as seen in Figure 2.1. To view the components under any component group, the users can simply click on the group library button, and all the components of the group will be displayed in the bottom left panel. Users can drag the components from the bottom left panel and move and drop it in the main window.

As mentioned previously, our GUI works on drag and drop approach. This approach is widely used by various commercial simulation software such as Simulink, Cadence, and P-SPICE etc. Furthermore, for our developed GUI, when the users select the component, the color of the component changes to green, blue or orange depending on the type of components. So, color coding is used in our software. The blue color represents the optical

component, the yellow represents the electrical component, and the orange represents the optoelectronic component. This color code is followed when component are placed in the Draw panel of main window. This color coding helps users to visualize and debug the optical and electrical components easily when large number of components are used to build the system. After placing more than one components in the design window, the users can use the arrow connection provided in Tool panel in the upper right section of the main window. The panel consisting of arrow connection, global variables, and loop variables will be referred to as a Tool panel in this paper. The users can click on the arrow button, then click on the starting component followed by finally clicking the ending component, to set up the connection between two components.

Currently, components are grouped into 9 component libraries in our software, however, our goal is to add more components in the future in each component library, and we also will add more component libraries. In the later section, we will discuss some new library added to the software. Following is the list of the component libraries including the component present in each library:

- Optical Passive Components: This library contains PRBS generator.
- Optical Active Components: This library contains component such as EDFA and Lasers
- Electrical Passive Components: This library contains Electrical CW, Electrical delay, Electric Coupler and Electric splitter
- Electrical Active Components: This library contains Bessel filter, Time-division multiplexing (TDM), optical delay, phaser etc.

- Optoelectronic Components: This library contains Mach Zender Modulator, photo detector
- Visualizer Components: This library contains component that analyzes signal such as oscilloscope, optical signal analyzer and so on.
- Signal Processing Components: The components under this library are NRZ generator and Bessel filter.
- Network Components: This library consists of PRBS generator, photo detector and Mach Zender modulator.
- Information Source and Code: This library currently only contains PRBS generator as its component.

All the components mentioned above have default parameters specifying their default properties, but the users can configure the components by entering users-defined parameter values using the parameter editor. Parameter editor of each component can be viewed by double-clicking the component. Figure 2.2 parameter shows the parameter editor of "PRBS" component. Parameter editor when initially opened consists of a default value. However, the users can enter their own values in the editor. In Figure 2.3, the parameter values such as "Number Bit Sequence", "Number Bits", and "Number Samples per bit" are changed to 16, 8 and 8 respectively from their default values. For different components, different parameter lists are displayed to describe different properties of the components. For some components, the users can also insert the datasheet file name as the component parameters. Some components might have a similar parameter in such case the GUI provides a functionality of declaring a global variable to the complete system.

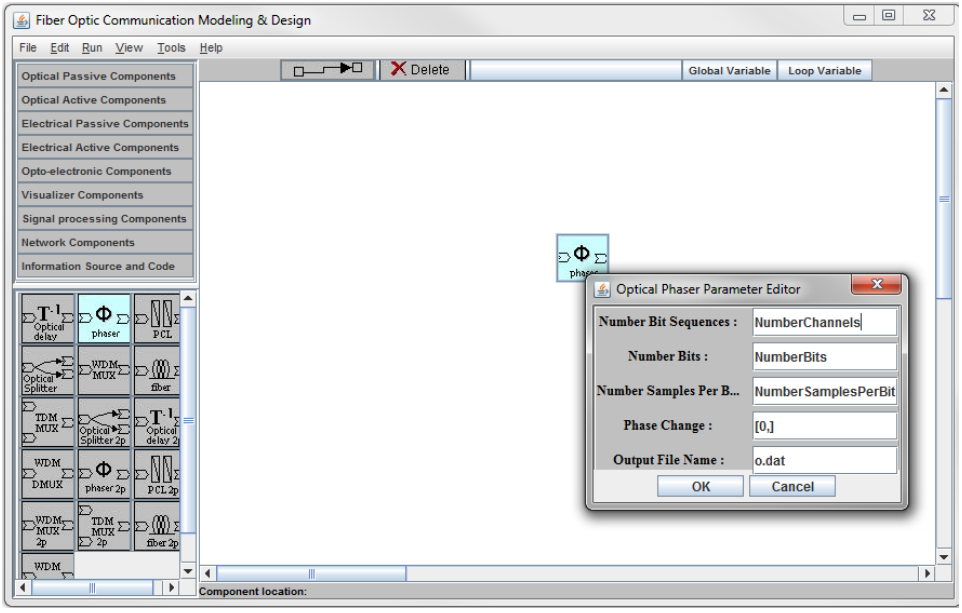


Figure 2.2: Parameter editor with default setup

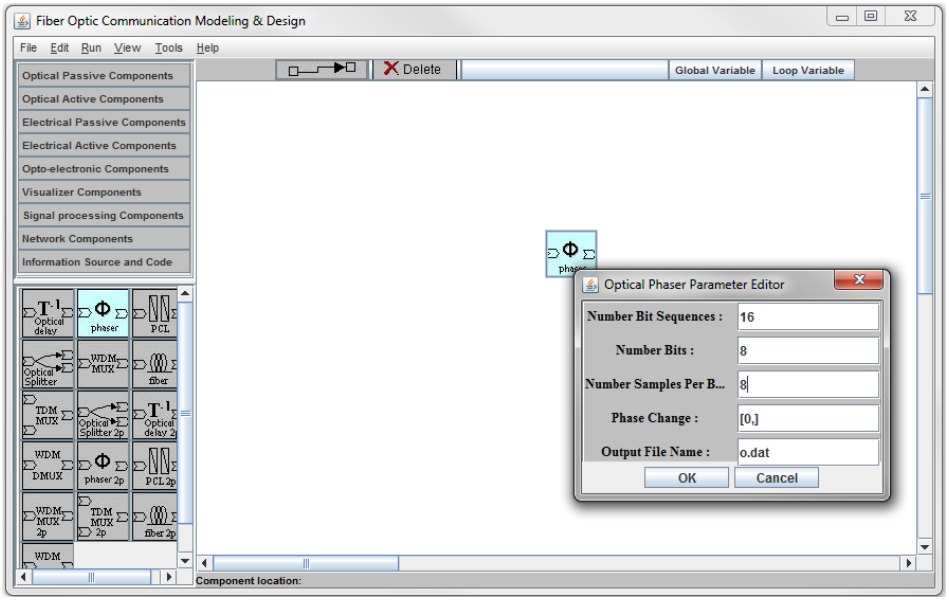


Figure 2.3: Parameter editor with user's configuration

The Tool panel consists of a button named as global variable and loop variable. When the users click the global variable button, global variable parameters editor is displayed. Here the users can enter the values for these global variables as per their system specification. These global variables are accessed by all the components. This makes it

easier for the users to define the variables that are used by various components in a single instance. In the component parameter editor panel, the users can enter the component parameter field with the global parameter name or mathematical formula which contains the global parameter. The loop variables button is adjacent to the global variable button. When double-clicked the loop variable editor will be displayed. Here the users can define loop variables and define the simulation loops. Our software currently supports up to 3 levels of simulation loops. Besides global and loop variable button, Tool panel also consists of a delete button. When the users need to redesign their system by removing some components and arrow connections, the users can click on the "Delete" button from Tool panel, move the cursor to select the undesired component, and with a right click, the component will be removed.

Once the system design layout and configuration is complete, the users can start to perform simulations. Now they first need to verify their system design layout by clicking on "Run Verify" item from menu "Run". This checks the connection of the system design, and if no error is found in the connections, the users can proceed with "Run Simulation". The "Run simulation" activates the server side of the software to perform simulation and produce the simulation results. The results are obtained as '.dat' file and these files are used in plotting tool to view the output in the form of a graph. The plot tool is flexible as users can plot the '.dat' file of on any component and observe the results of any section of the simulated system.

Now that we discussed GUI that involves with the system design aspect, in this section we will discuss how to save those design and load it for future reference. As we know, all the available simulation software in the market has this universal functionality of saving

and loading the design. Our software also has this feature of saving and loading the file. The users can save their design as .xml file by clicking “Save” button on the “File” menu. These save file can be opened using “open” option from the “File” menu.

In all, GUI of our software consists most of the essential functionality that any commercial simulation software must have. In the following chapters, we will look into more details about each component of our GUI. Before that, we will briefly explain some swing components involved in the development of our GUI.

2.2 Java Swing Components

We have earlier mentioned that we used Java swing components for the development of our GUI. The swing components extensively used were JFrame, JPanel, JButton, dialog box, menu bar etc. Here, we will discuss JFrame and JPanel to understand the concept of swing components and its implementation in Java.

2.2.1 JFrame

JFrame is a swing version class of a Frame class present in AWT library. Frame is a top-level window that cannot be contained by any other window. It is contained in a javax.swing package and is declared as “public class JFrame extends Frame implements WindowConstants, Accessible, RootPaneContainer”. We know that in Java the class comprises of fields, constructors, and methods. We have listed the constructors with their descriptions in following Table 2. 1.

Table 2.1: Constructor of JFrame class

Constructor	Description
JFrame()	It constructs a new frame that is initially invisible
JFrame(GraphicsConfiguration gc)	It creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
JFrame(String title)	It creates a new, initially invisible Frame with the specified title.

From the above table, we used JFrame (String title) constructor for our GUI. Now, we will discuss the various methods available in JFrame class in following Table 2.2.

Table 2.2: Methods of JFrame class

Method	Description
protected void addImpl(Component comp, Object constraints, int index)	It adds the specified child Component.
protected JRootPane createRootPane()	It is called by the constructor methods to create the default rootpane
protected void frameInit()	It is called by the constructors to initialize the JFrame properly
Container getContentPane()	It returns the contenPane object for this frame
int getDefaultCloseOperation()	It returns the operation that occurs when the users initiates a "close" on this frame.
Graphics getGraphics()	It creates a graphics context for this component
JMenuBar getJMenuBar()	It returns the menubar set on this frame
protected void processWindowEvent(WindowEvent e)	It processes window events occurring on this component
void remove(Component comp)	It removes the specified component from the container.
void repaint(long time, int x, int y, int width, int height)	It repaints the specified rectangle of this component within time milliseconds.
void setIconImage(Image image)	It sets the image to be displayed as the icon for this window.
void setJMenuBar(JMenuBar menubar)	It sets the menubar for this frame,
void setLayout(LayoutManager manager)	It sets the LayoutManager.
void update(Graphics g)	It calls paint(g)

From the above-mentioned list of methods, we used methods such as setLayout (LayoutManager manager), processWindowEvent(WindowEvent e), setIconImage(Image image), getJMenuBar(), setJMenuBar(JMenuBar menubar), void update(Graphics g), and update(Graphics g) etc. for the development of our GUI.

2.2.2 JPanel

JPanel is a generic lightweight container that can contain other components. It inherits JComponent class and provides space where an application can attach any other component. This class is declared as "public class JPanel extends JComponent implements Accessible" in javax.swing.Jpanel class file. The constructors of this class are listed in Table 2.3 as follows:

Table 2.3: Constructors of JPanel class

Constructor	Description
JPanel()	It creates a new JPanel with a double buffer and a flow layout.
JPanel(boolean isDoubleBuffered)	It creates a new JPanel with FlowLayout and the specified buffering strategy.
JPanel(LayoutManager layout)	It creates a new buffered JPanel with the specified layout manager.
JPanel(LayoutManager layout, boolean isDoubleBuffered)	It creates a new JPanel with the specified layout manager and buffering strategy.

From the above list of constructors, we utilized JPanel() for our GUI design. Some of the methods of this class are listed in the following Table 2.4:

Table 2.4: Methods of JPanel class

Method	Description
PanelUI getUI()	It returns the look and feel object that renders this component.
String getUIClassID()	It returns a string that specifies the name of the look and feel class which renders this component.
protected String paramString()	It returns a string representation of this JPanel.
void setUI(PanelUI ui)	It sets the look and feel object that renders this component.
void updateUI()	It resets the UI property with a value from the current look and feel.

2.3 Structure of Graphical User Interface (GUI)

In the above section, we explained some key swing components with their constructors and methods. These swing components, along with other components not included here, are implemented to develop various parts of our GUI. Here, we will discuss these various parts of GUI individually. A single mainframe is developed using JFrame. Inside the mainframe, several containers are added each containing various JPanels. The mainframe consists of 3 major components: Left panel (JPanel), Mid panel (JPanel) and Menu bar (JMenubar) as illustrated in Figure 2.4.

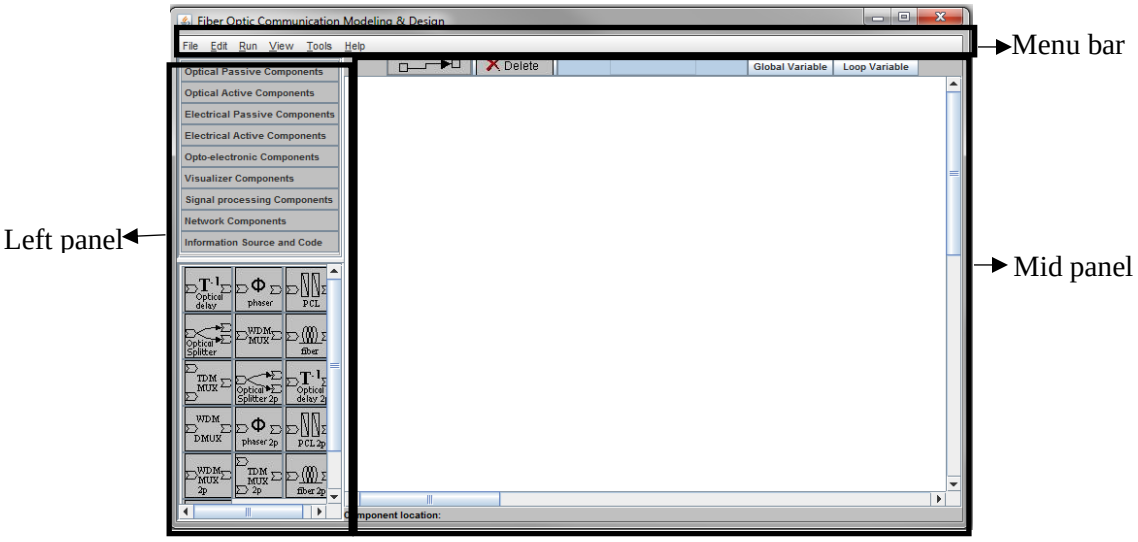


Figure 2.4: Main Windows highlighted: Left panel, Mid panel and Menu bar

2.3.1 Left panel

We will begin our explanation of individual components of GUI with the Left panel. Figure 2.5 illustrates that the Left panel is composed of two sections. Here we have distinguished two sections as upper panel and lower panel as shown in Figure 2.5. The upper panel consists of the component libraries, and the lower panel consists of components inside the library. These panels are constructed using JscrollPane. The upper panel consists of 9

buttons representing 9 component libraries, whereas lower panel displays the components present in the library. When the users select any component library, the components are displayed in the lower panel. For each group of 9 buttons, 9 different JPanels are assigned, and each 9 buttons is provided with action listener. The action carried by each button is to display their corresponding panel with the components in the lower panel. Furthermore, components in lower Panels are buttons with image icon. The images help the users to easily identify the components.

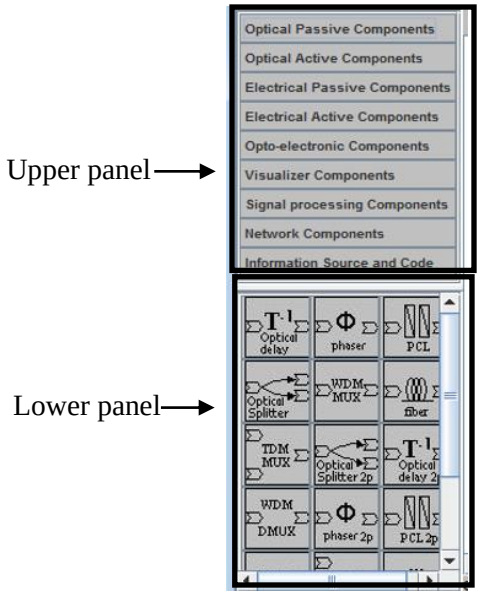


Figure 2.5: Upper panel Lower panel of Left Pane

2.3.2 Mid panel

Here, we will explain the composition of Mid panel. Mid panel is a JPanel with the default width and height of 550 and 500 respectively. It can be grouped into three main sections: Tool panel (JPanel), Draw panel (JPanel), and Label (JLabel) as shown in Figure 2.6. For a better understanding of Mid panel, we will explain each section in detail.

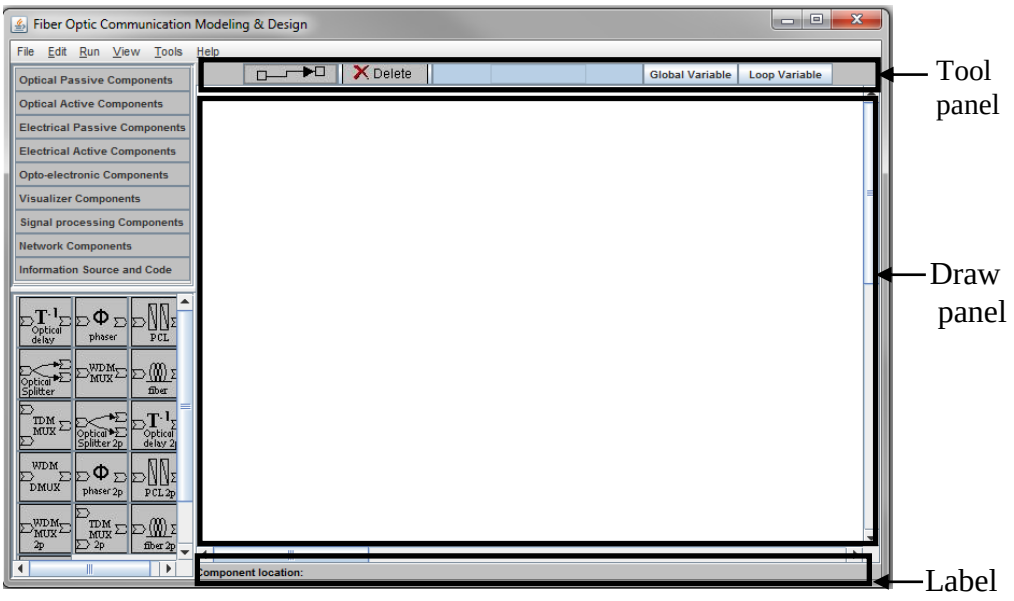


Figure 2.6: Main Window highlighting Tool panel, Draw panel and Label

(a) Tool panel

Figure 2.6 illustrates that the Tool panel is located on the top of the Mid panel. The Tool panel consists of 4 major buttons, “Arrow”, “Delete”, “Global Variables”, and “Loop Variables”. Using arrow button, users can connect two components, whereas with delete button the users can delete the component. The arrow button is color-coded to represent the different state of the button. Figure 2.7 shows the different state of arrow button. Furthermore, when users click a “Global Variable” button a dialog box appears as shown in Figure 2.8, with a number of global variables linked with all the components of all the libraries. User can edit the parameters and set up the values for the variables as per their need. This makes it easy for the users to change the variable at only one location rather than changing the variables at each component. The working methodology and functionality of “Loop Variable” button is same as the “Global Variable” button. When this button is clicked, a dialog box appears as shown in Figure 2.9 where users can edit the

loop variables. All these actions are controlled by actionlistener associated with respective buttons.

 (1st State: Default state),

 (2nd State: Button pressed),

 (3rd State: Button pressed and released).

Figure 2.7: Different state of Arrow button

(b) Draw panel

The system layout is designed in Draw panel using components and arrows. This is a platform where components form low panel is dragged, moved in, and dropped. Furthermore, components need to be connected using the arrow in this panel. This panel is placed in the center of Mid panel. The Draw panel is a JScrollPane and consists major interactive features as this panel handles the system design layout and parameter editor of each component. First, there is no restriction on where users can drop the components, and they also have the flexibility of moving the components around any corner of Draw panel. Second, these components after placed in Draw panel can be double-clicked, which displays a dialog box containing parameters editor of the component.

(c) Label

This is in the bottom of the Mid panel as shown in Figure 2.6. This is developed using JLabel class of Java swing library. The location of this section is also managed by Borderlayout manager. The Label displays the coordinates of the components when clicked. When the users click on the components, the x and y coordinate of the component location is retrieved and displayed inside the Label. It eases users to identify the location of the component when a substantial number of components are placed in the window.

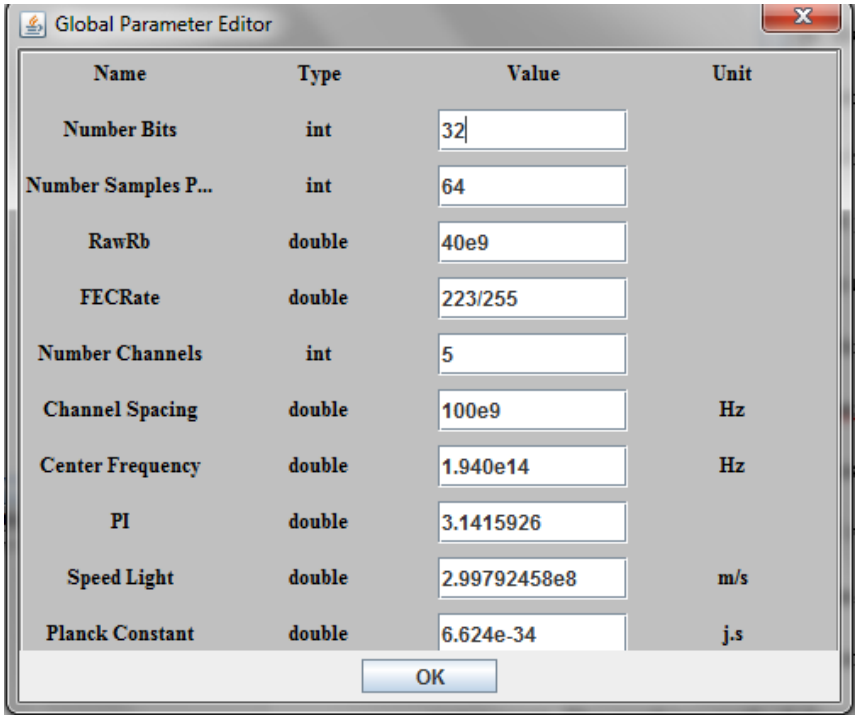


Figure 2.8: Global Variables Parameter Editor

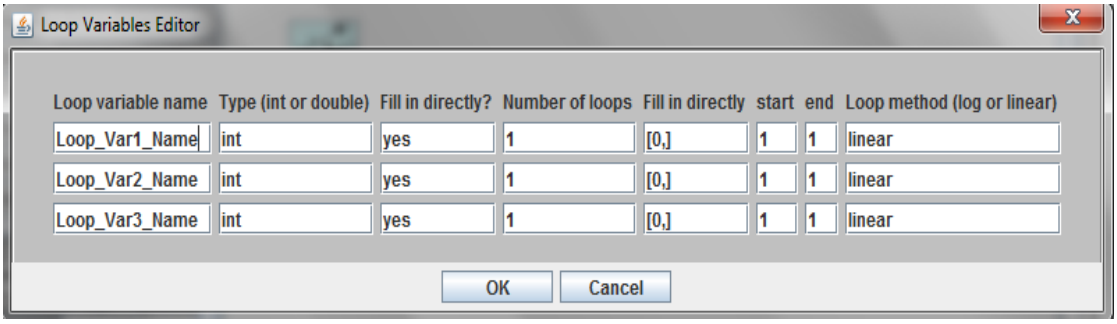


Figure 2.9: Loop variable editor

2.3.3 Menu Bar

The third major component inside the mainframe of the GUI is Menu bar. As shown in Figure 2.4 the software Menubar consists of 5 menu options: File, Edit, Run, View, Tools, and Help. These menu options are created as a menu item in Java and then added to the menu bar. The menu items are constructed using JMenu class, whereas menu bar is constructed using JMenuBar class. Both JMenuBar and JMenu class belongs to Java swing

package. Furthermore, each Menu consists of sub-items, for example, “File” menu consists of “New”, “Open”, “Close”, “Save”, “Save As”, “Print”. These menu items are constructed using the JMenuItem class of Java swing package. For each menu items, action listener is assigned along with the actions. For instance, when “New” button is clicked, a new window is created clearing all the existing design in the window. Similarly, using JMenuItem other menu items for “Edit”, “Run”, “View”, “Tools” and “Help” menu is constructed and each item is provided with specific actions implementing actionlistener.

2.4 Interactive features of Graphical User Interface (GUI)

In the previous section, we focused on the layout of GUI. Here we will discuss the usage of the GUI. First, we will briefly explain about the interface of Java used to create an interactive environment in our software. Then, we will delve into the interactive environment of our software. Most of the interaction with GUI is performed with the mouse, as the users can drag, move, and click the components. The keyboard is also used on many occasions to enter the parameters of the component, to save, and to open files. The interface such as actionlistener, mouselistener, mousemotionlistener provided by Java are extensively used to develop these interactive features of the GUI.

2.4.1 Java interface used for designing our software

In this section, we will discuss the interfaces provided by Java. Java provides the “ActionListener” interface for adding actions to its swing components. The use of “ActionListener” is usual in Java applications. We use this “ActionListener” interface to define what action is taken when the users perform certain operation [39]. The actions can be the following: the users clicking a button, choosing a menu item, entering a text in a text field and so on. When any of the above-mentioned action is performed, an actionPerformed

message is sent to all the action listeners that will be registered on the relevant component. We can implement action listener in Java by following steps:

1. We declare an event handler class and specify that the class either implements an ActionListener interface or extends a class that implements an ActionListener interface. For example:

public class MyClass implements ActionListener {…}

2. We then register an instance of the event handler class as a listener on one or more components. For example:

someComponent.addActionListener(instanceOfMyClass);

3. Finally, we include code that implements the methods in listener interface. For example:

public void actionPerformed(ActionEvent e) {

...//code that reacts to the action...}

Similarly, to provide actions especially to various mouse events, Java consists of interface called “MouseListener”. The Java MouseListener is notified whenever you change the state of the mouse. In other words, MouseListener is notified when any MouseEvent occurs. The mousevent on a component can be press, release, click, enter, and exit. The MouseListener interface is found in java.awt.event package. The Mouse Listener interface consists of five methods and they are: [39-40]

1. public abstract void mouseClicked(MouseEvent e)
2. public abstract void mouseEntered(MouseEvent e)
3. public abstract void mouseExited(MouseEvent e)
4. public abstract void mousePressed(MouseEvent e)

5. public abstract void mouseReleased(MouseEvent e)

Furthermore, to track mouse moves and mouse drags, we can use the MouseMotionListener interface. The class that deals with the mouse event either implements this interface (and all the methods it contains) or extends the abstract MouseAdapter class (overriding only the methods of interest). The 2 methods found in MouseMotionListener interface are [39, 40]:

1. public abstract void mouseDragged(MouseEvent e)
2. public abstract void mouseMoved(MouseEvent e)

2.4.2 Action performed by the components of our GUI

The interactive feature of our GUI is based on the above-mentioned interface of Java. Using above interfaces, interactive features are provided to the Left panel, Mid panel, and Menu bar. The Left panel consists of 9 buttons representing the component library in its upper panel. When any button among 9 buttons is clicked, the button gets highlighted and the components are displayed in the lower panel. This linkage between the upper panel and lower panel is achieved by using action listener interface of Java. Furthermore, the buttons are highlighted when clicked to make the users easily identify the button when pressed. The components inside the lower panel are toggle buttons. These buttons are provided with icon image as per the component's properties. When the components are selected in the lower panel it gets highlighted with different color. The components in the upper panel and lower panel are the toggle button. If the users decide to choose another component instead then the users can simply choose the next component, and when that happens, the program control will simply switch over to the new, currently selected component.

After the component form lower panel is selected, the users can drop the component selected in Draw panel. The Draw panel consists major interactive features based on the mouse event handler. First, there is no restriction on where the users can drop the component, and they also have the flexibility of moving the component around any corner of Draw panel. Second, when the component, after being placed in Draw panel, is double-clicked, a dialog box will be displayed containing parameters editor of the component. This interactive feature of component movement and double-click is obtained by using mousemotionlistener and mouselistener. Furthermore, inside parameter editor, opened by double-clicking of the component, the users can edit the parameters as per their system requirement.

Once more than two components are placed in the Draw panel, the users can use the "arrow" from the Tool panel to connect components in Draw panel. To connect the components, the users need to select the starting component and ending component. When the users click the start component the action performed is to obtain the coordinate of that component, similarly when the users click the second ending component the coordinate of the second component is obtained. Using these coordinates and arrow is drawn by the software using the graphics function of Java.

The Tool panel also consists of delete button which is used to delete components and connection in Draw panel. In addition, the Tool panel consists of two more interactive components "Global Variable" and "Loop Variable". When the users click "Global Variable" button a dialog box appears as shown in Figure 2.8. This global variable editor contains a number of global variables which are linked with various components. The users can edit the global parameter names and set up the values as per the users' need. This makes

it easy for the users to change the variable at only one location rather than changing the variables for each component. The working methodology and functionality of “Loop Variable” button is same as the “Global Variable” button. When this button is clicked a dialog box appears as shown in Figure 2.9, where the users can configure the loop variables.

After laying out the system and entering all the parameter values, the users can verify and run their simulation. To verify the simulation, the users need to select “Run” menu and then click “Run verify”. The action handler event provided to this button is such that it checks if the connection and parameters provided by the users comply with our software restriction. If software doesn’t yield any error pertaining to system layout then, users can proceed with simulation by clicking “Run simulation” option under “Run” menu. When “Run simulation” is clicked it triggers the server side to perform its operation and produce the output. The output refers to the data file containing the results. For each component, there exists a parameter field, where the users can enter a file name to save the optical or electrical signal field propagating through that specific component [37]. After simulation and calculation, the users can go to “View” menu and then click on “Result View”, this opens a plotting editor as shown in Figure 2.10. In this plotting editor, the users can enter the data file name and control parameters and view the plot of their simulation results. The users can also save their system design to an XML file and load it back again. The next section deals how the saving and loading is handled in our software.

2.4.3 Saving and Loading the design

Saving and loading of the system design are performed using Extensible Markup Language (XML) file. We first briefly discuss how the XML file is used to store the data. This will be followed by the explanation of the implementation of the XML file in our software for

data handling. As we mentioned before, the users can save their work by clicking the "Save" button in the File menu. The files are saved in .xml format. All the information of the component physical properties, component parameters, global variables and loop variables are stored in the XML file. When users hit the "Save" button from the Menu a XML file is created. User can specify the name of the XML file to be saved.

The saved XML file can be opened using a menu option "Open" from the File Menu. The users can select any XML file that was previously saved. When the XML file is loaded, the saved design will appear in the window. Also, all the parameters previously saved will also be available and present for the newly launched system.

2.5 Sorting Algorithm

In this section, we will briefly discuss how the software is working internally to generate the execution order of the components placed in the 2-dimensional GUI display. In our software, the users can place the components in Draw panel in any order. For instance, in Figure 2.11 component B might be the first component placed in the Draw panel, C might be the second, and A might be the last. Furthermore, the users can connect arrow in any order. The users can choose to connect an arrow from A to B first. Then, the users can make arrow connection from B to C. The other possibility is that users connect arrow from B to C first, followed by the connection from A to B. This suggests that the placement of component and arrow connection can be in an arbitrary order. However, for simulation purpose, the order of execution must be A, B and then C. Our algorithm, arranges the arbitrarily placed and connected arrows to a correct order of execution for simulation. When the users hit "Run Verify" button this sorting process is executed. After that process, the users can perform the simulation. Therefore, it is a must to perform "Run Verify" before

performing the simulation or clicking “Run Simulation” button in our software. If the simulation is performed without performing “Run Verify” there will be a simulation error.

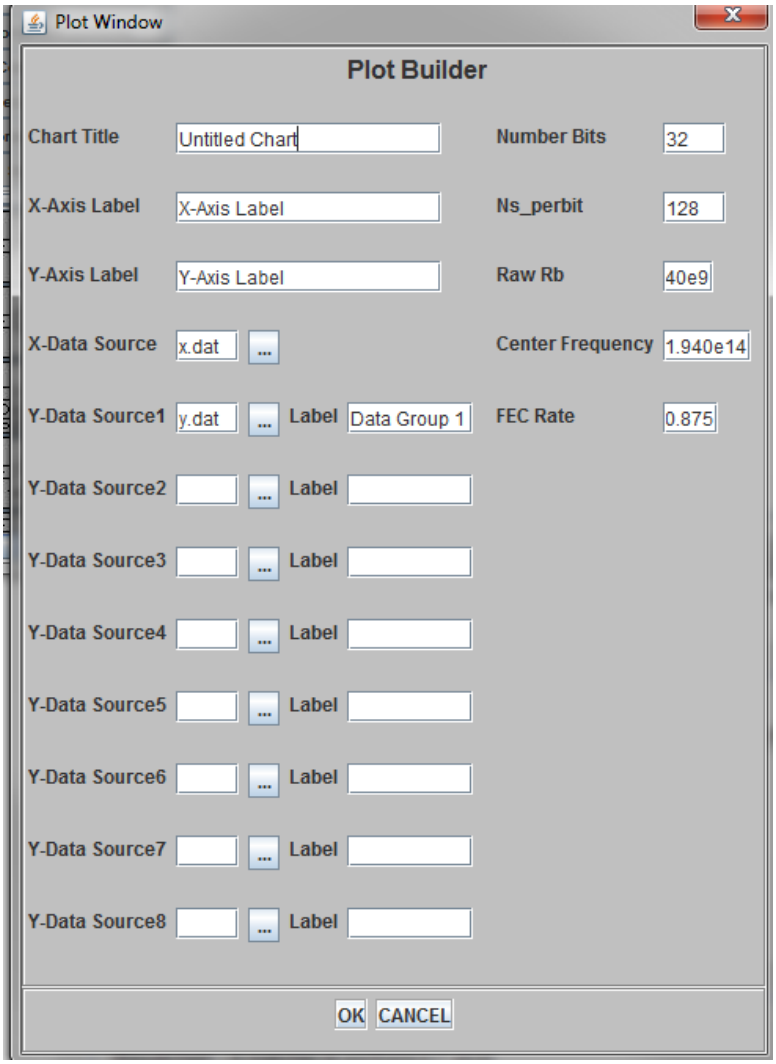


Figure 2.10: Plot Builder of our Software

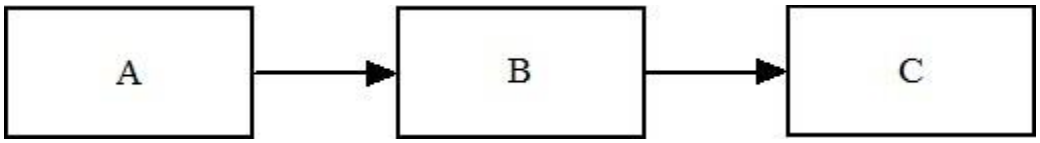


Figure 2.11: Block representation of component and its connection

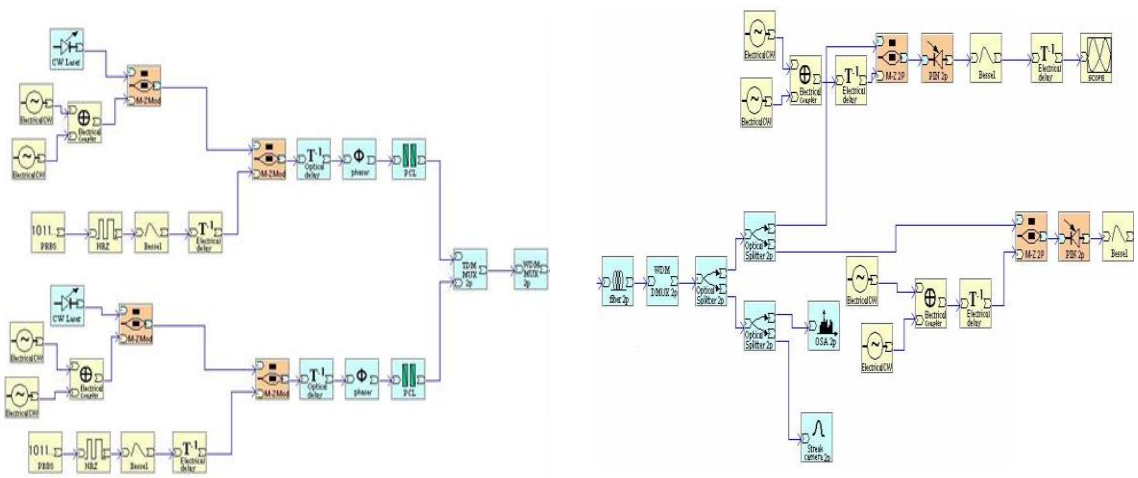
2.2 Usage of GUI with an example

In this section, we explain the usage of the GUI to simulate an optical communication system. The complete simulation not only involves the GUI side of work e.g. laying out and configuration of the system to be simulated but also involves the server side work, which is out of the scope of this thesis. We have taken this example from Dr. Zhang's paper [37] to make the thesis self-contained. The following example is not part of this thesis work, however, we found it essential to add to this thesis to provide overall usage of our software and its implementation in designing complex optical system.

We use the developed software to model and design an 80 Gb/s optical time division multiplexing (OTDM) system which employs intensity modulation direct detection (IM-DD) transceivers. System layout in the design window is shown in Figure 2.12 (A) and (B). The design project was named MiniCase.

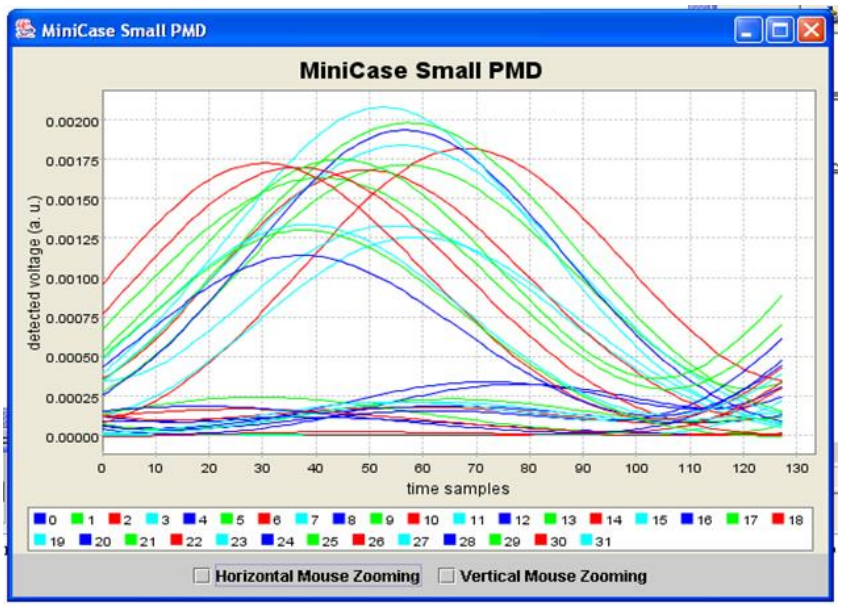
We assume that the two 40 Gb/s optical signals are generated by electrical time division multiplexing (ETDM). Then they are multiplexed in the optical domain using OTDM technology into an 80 Gb/s signal. Seven such wavelength channels are combined through the WDM MUX. We use an orthogonal launch meaning that neighboring wavelength channels have orthogonal state of polarizations (SOPs) with respect to each other. After the transmitter EDFA, the 7×80 Gb/s optical signal is launched into a fiber plant that is composed of 5 spans of 80 km fibers with distributed Raman amplifiers. Please note that although not shown here, the fiber2p component has more than 40 parameters/parameter arrays for the setup of fiber plant and for describing the property of the fibers. At the end of the link, the center wavelength channel carrying 80 Gb/s optical signal is selected by the WDM DEMUX. The 80 Gb/s optical signal then passes the time domain DEMUXs

resulting in 2×40 Gb/s optical signals. In Figure 2.13 (A) and 21 (B), we show the eye diagrams of the detected voltage with two different fiber PMD values, 0.06 $ps/\sqrt{\text{km}}$ for the new type of fiber and 0.3 $ps/\sqrt{\text{km}}$ for the old type, respectively. The eye closing in Figure 2.13 (A) results from fiber nonlinearity and WDM linear crosstalk. With an increase of the PMD value, the eye-opening further decreases, as shown in Fig. 8. We also did an optimization study on the launch signal power. We found that when the 0.5 nm link OSNR = 17 dB, best system performance of Q = 12.9 dB (BER = 5e-6) can be achieved if the new type of fiber is used. Please note we assume that all the OTDM channels have same optical launch phase thus there is no coherent crosstalk.

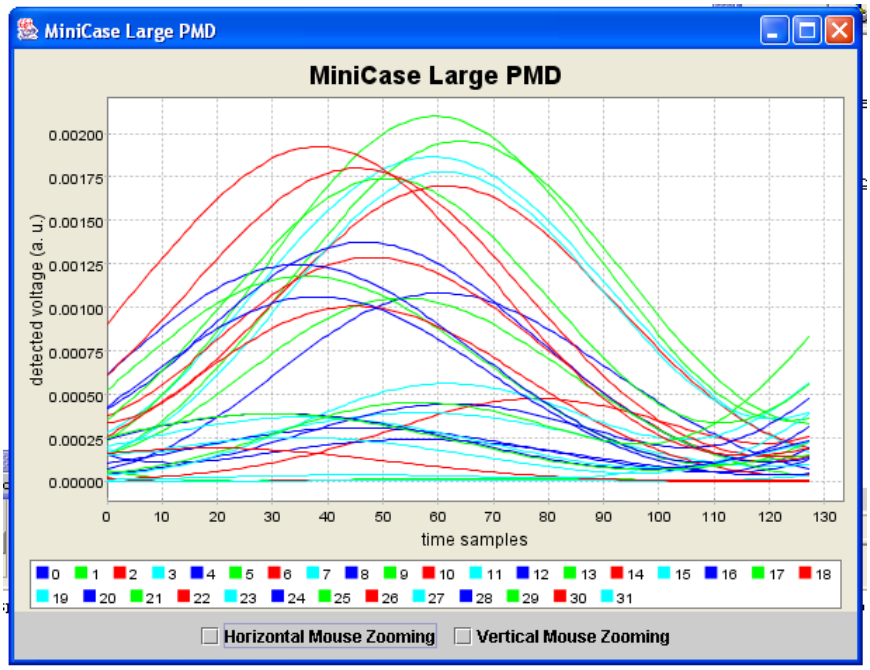


(a) (b)

Figure 2.12: System layout (A) The transmitter side; (B) the fiber channel and the receiver side



(a)



(b)

Figure 2.13: Electric eye diagram: (a) for PMD = 0.06 $ps\ /\ \sqrt{\text{km}}$,

(b) for PMD = $ps\ /\ \sqrt{\text{km}}$

Chapter 3

Development of GUI plotting tool and study of hierarchical design

In this chapter, we will summarize the work we performed related to the GUI software. We performed the two major tasks on the software. The first task performed was the development of the simulation result visualization plotting tool for the GUI and the second task performed was to modify the GUI to achieve the hierarchical design. In the first section of this chapter, we will explain a library we used to develop the plotting tool along with the functionality of the developed tool itself. Then in the second section, we will explain the concept of hierarchical design with examples from the Ptolemy software. Finally, we will present our hierarchical design concept and explain its features, and our implementation examples.

3.1 Development of the Plot tool

We used JFreechart library to develop our plot tools. To utilize this library, we add two libraries namely JFreechart and JCommons in our Java package file, where JCommons is a Java class library that is used by JFreechart. This allows us to use the class files and method of JFreechart library in our Java application. We used various classes and methods of this library to build our plotting tools as this library provides comprehensive features dedicated to the development of charts. We first explain the JFreechart.

3.1.1 JFreechart

JFreeChart is an open source library developed in Java [41] that supports a wide range of charts for the Java based application. The major type of 2D and 3D charts such as pie chart, bar chart, line chart, XY chart, area charts, scatter charts, pie charts, 3D charts, and various specialized charts such as wind chart or bubble chart, can be built with JFreechart. Furthermore, JFreeChart is extensively customizable. We can modify the chart as per our need. For instance, we can change colors of chart items, legends, styles of the lines or makers. Also, it provides the axis scale and legends automatically. Without any additional code, we can use the mouse to zoom in and out the chart. The existing charts can be easily updated through the listeners in the library's data collections. In addition, we can save the chart in multiple formats such as PNG, JPEG, PDF, and SVG which makes plot tool flexible and users-friendly [41]. Apart from the advantages mentioned above, we used JFreechart library because it has well-documented APIs which made it easy for us to understand and implement it in our application. Some of the APIs and the JFreechart architecture is explained in next section.

Figure 3.1 illustrates the class level architecture of JFreechart [42]. It shows how classes from library interact with each other to create various type of charts. In Figure 3.1, File represents the source file that contains the dataset e.g., our simulation result data whereas, Database represents the database containing dataset. We only use the File as our main source of the plot data. After developing the dataset, Create Dataset accepts these datasets and store them in a dataset object. There are three kinds of Dataset: General Dataset, Category Dataset, and Series Dataset. General Dataset is used to build pie charts, Category Datasheet is used for bar chart and line chart etc., and Series Dataset is used to construct

line charts using series of data. Various series data set can be added to series collection dataset which is used for XYLine Charts. After the data collection, create chart method is executed to create a final chart. The final chart is displayed as an image on a frame. The users can also create an image of a different format as per their need.

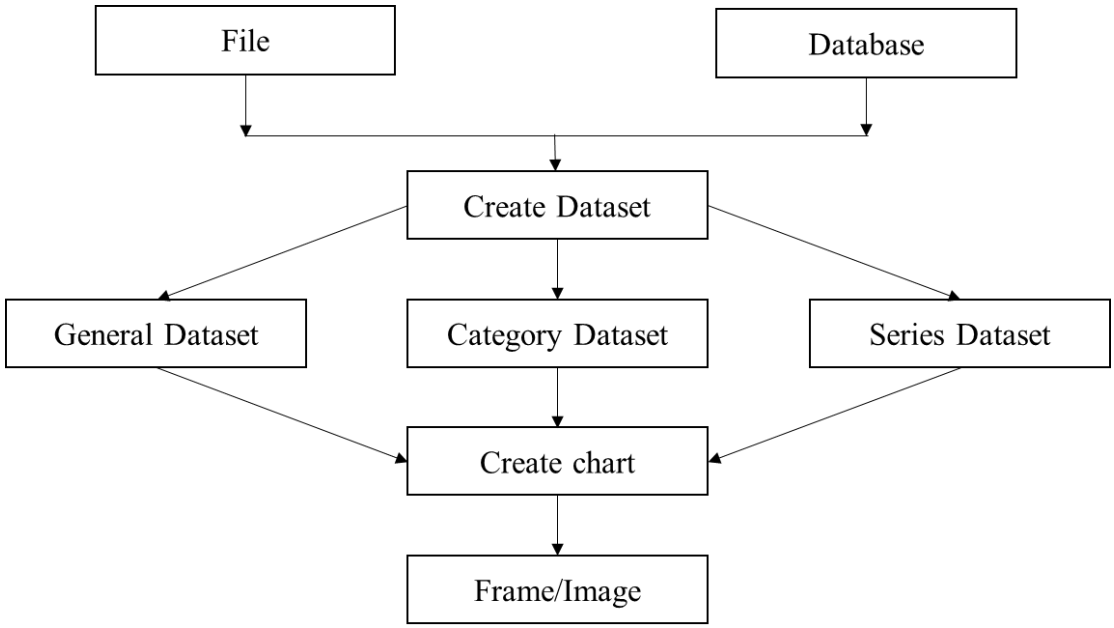


Figure 3.1: Class level Architecture of JFreechart

Above we discussed the class level architecture of JFreechart. Here we discuss how JFreechart library is placed inside a Java Application. As shown in Figure 3.2, the input data is first fed to our client program. For instance, the input file could be the '.dat' file and with the use Java API we can input the data from the file to our program. Then using JFreechart APIs, we can process these data in any format as per our need. For example, we can use JFreechart APIs to process our data in (x, y) format such that we can get the output as XY line chart. We can then view the output in an application frame or as PNG/JPEG image file.

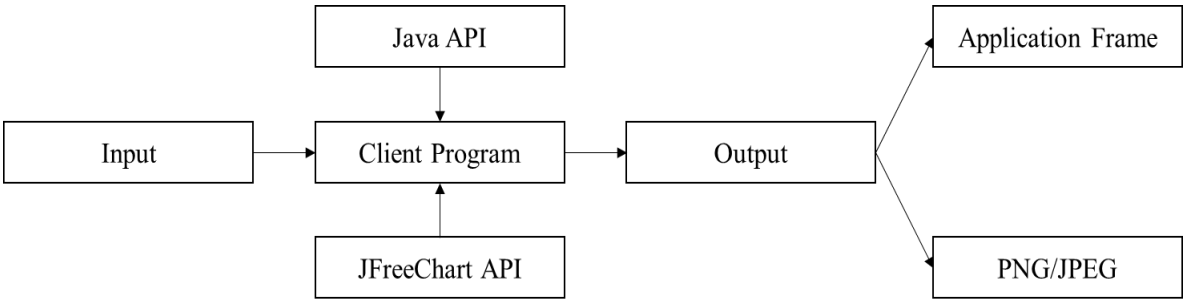


Figure 3.2: Application level architecture

So far, we have only focused on the architecture of the JFreechart without consideration of its implementation in Java. Now, we will focus on the JFreechart library with its various API. The API contains numerous packages, classes, and methods, and using these API we built our plotting tools. In this section, we will limit the explanation of JFreechart API to some of the major contents of the JFreechart library that are frequently used. We will discuss some major class and for each class, we will briefly explain their constructors and methods. We also discuss how they are used in our developed software. Following are some of the major class files of JFreechart library.

i. JFreeChart Class

It is the core class in the org.jfree.chart package. It consists of the major method to create bar charts, line charts, pie charts, and XY plots including time series data. We used this class to develop a chart for our results. The method getXYPlot() was used in development for our plotting tool. Following Table 3.1 and Table 3.2 describe about the constructors and method of this class respectively.

Table 3.1: Constructors of JPanel class

Class Constructor	Description
JfreeChart(Plot plot)	It constructs news chart based on the supplied plot.
JfreeChart(java.lang.String title, java.awt.Font titleFont, Plot plot, boolean createLegend)	This constructor creates a new chart with the given title and plot.

JfreeChart(java.lang.String title, Plot plot)	This constructor creates a new chart with the given title and plot.

Table 3.2: Method of JPanel class

Methods	Description
getXYPlot()	It returns the plot chart as XYPlot. It allows utility operations on XY charts.

ii. PlotOrientation Class

This is a serialized class in org.jfree.chart.plot package. The orientation of plot either horizontal or vertical is set through this method. We used this to set the orientation of Y-axis. Information about various orientations that were allowed by the class is listed in the following Table 3.3 and Table 3.4.

Table 3.3: Orientation available in PlotOrientation Class

Type	Description
PlotOrientation	Plot orientation can be set up as horizontal Here, the range axis (Y-axis) is horizontal.
PlotOrientation	Plot orientation can be set up as horizontal Here, the range axis (Y-axis) is vertical. This is the default orientation.

Table 3.4: Methods of PlotOrientation Class

Methods	Description
isHorizontal()	If this orientation is horizontal it returns true, and false otherwise.
isVertical()	If this orientation is vertical it returns true, and false otherwise

iii. XYPlot Class

This is a general class under the package org.jfree.chart.plot. We utilized this class to plot data pairs in the form of (x, y) where x represents the x –coordinates and y represents y coordinates. Since this class implements XYDataset interface we arranged the data points of our results as the XYDataset. Table 3.5 and Table 3.6 shows the constructor and methods of this class respectively:

Table 3.5: Constructors of XYPlot class

Class Constructor	Description
JfreeChart(Plot plot)	It constructs a new XYPlot instance with no dataset, no axes, and no renderer.
XYPlot(XYDataset dataset, ValueAxis domainAxis, ValueAxis rangeAxis, XYItemRenderer renderer)	It creates a new plot using the specified dataset, axis, and renderer.

Table 3.6: Method of XYPlot class

Methods	Description
setRenderer(XYItemRenderer renderer)	It sets the renderer for the primary dataset and sends a change event to all registered listeners.

iv. NumberAxis Class

This class is under org.jfree.chart.axis package. It is used to access numerical data of any axis. It has a mechanism for automatically selecting a tick unit that is appropriate for the current axis range [42].

The default axis range fits the axis according to the range of the data. However, this class allows the users to set the lower margin and upper margin of the domain and range axes. We list constructors and methods of this class in following Table 3.7 and Table 3.8 respectively.

Table 3.7: Constructors of NumberAxis class

Class Constructor	Description
NumberAxis( )	This is a default Constructor.
NumberAxis(java.lang.String label)	It constructs a number axis, with default values where necessary.

Table 3.8: Methods of NumberAxis class

Methods	Description
setLowerMargin(double margin)	It is the method of ValueAxis class. Lower margin for the axis is set executing the method. Further, it triggers an AxisChangeEvent to all registered listeners.
setUpperMargin(double margin)	It is the method of ValueAxis class. The upper margin for the axis is set executing

	the method. Further, it triggers an AxisChangeEvent to all registered listeners.

v. XY Datasets

It is an interface through which data in the form of (x, y) items can be accessed. We can use this dataset to create an XY chart. This interface allows plotting the values in terms of x vs y such that we can get the XY plot.

vi. XYSeries

The class represent represents data items in the form (x, y) or a sequence of zero. By this it means, in the XYSeries object, we can add several x-coordinate values and y- coordinate values. For example, if we want to add values such as (1, 5) and (5, 6), we can proceed in the following way:

```
XYSeries series = new XYSeries("Random Data");
    series.add(1,5);
    series.add(5,6);
```

The above code will store the two (x, y) items. The default setup is such that the items in the series are in ascending order by x value. The duplicate x-values are also allowed by the class. The default setup of sorting and duplicate can be varied in the constructor. The missing values of Y can be represented as null. Table 3.9 and Table 3.10 illustrates some of the constructors and method of this class respectively.

Table 3.9: Constructors of XYseries class

Class Constructor	Description
XYSeries(java.lang.Comparable key)	It creates a new empty series.
XYSeries(java.lang.Comparable key, 54oolean autoSort)	It creates a new empty series with the auto-sort flag set as requested. The duplicate values are allowed by the constructor.

XYSeries(java.lang.Comparable key, boolean autoSort, boolean allowDuplicateXValues)	It constructs a new XY-series that contains no data.

Table 3.10: Methods of NumberAxis class

Method	Description
add(double x, double y)	It adds the data item to the series.

vii. XYSeriesCollection

When XY series is populated with values, we create XYSeriesCollection which is then passed on to develop a final chart. This class represents a collection of XYSeries objects that can be used as a dataset. It inherits the properties of AbstractXYDatset, AbstractSeriesDataset, and AbstractDataset class. Referring to the code in above section (vi), we can construct the XYSeriesCollection of the series in following way:

Final XYSeriesCollection data = new XYSeriesCollection(series);

After the construction of the XYSeriesCollection, it can be passed as the dataset to JFreechart constructor which will produce a chart with the data points. Some of the constructors and methods of this class are as follows:

Table 3.11: Constructors of XYSeriesCollection class

Class Constructor	Description
XYSeriesCollection()	It constructs an empty dataset.
XYSeriesCollection(XYSeries xyseries)	It constructs a dataset and populates it with a single series.

Table 3.12: Methods of XYSeriesCollection class

Method	Description
addSeries(XYSeries series)	It adds a series to the collection and triggers a DatasetChangeEvent to all registered listeners.

3.2 Plot tool

We have discussed the library we used to develop our plot tool. Here we discuss the function specifications of our plot tool. Using the plot tool, the users can plot their data

result obtained from simulation. The users can open the plot tool, by selecting “Results view” option from “View” menu. When the users click on the “Results view” button, under “view” menu bar, following dialog box appears as shown in Figure 2.10. Here in the plot builder, the users can enter chart title, x-axis label etc. Furthermore, the users can enter the x-data file and y-data file names. These files contain simulation results. The users can plot the data at any instance of the simulation. Since output data file is generated for each component after simulation, the users can use these data files to view the results using the plot tool. The users can also enter multiple y-data files to overlap multiple curves in one figure.

The functionality of the Plot builder are as follows:

1. The text field corresponding to “Chart Title”, “X-Axis label”, and “Y-Axis Label” is used to enter the title, axis label as per user’s choice.
2. The box corresponding to “X-Data Source” is a location where we enter the “.dat” file containing data point for the X axis. The users can browse the file by clicking on a small button right to the box. Currently, the code is structured such that the file must be present in “usersDataresults” folder. Therefore, the users can only enter the file contained in “usersDataresults” folder.
3. The box corresponding to “Y-Data Source” is a location where we put the “.dat” file containing data point for Y-axis. Similar steps mentioned in 2 should be taken to successfully upload the “.dat” files.
4. The users can press “OK” button once both boxes pertaining to X-Data Source and Y- Data Source are populated with the desired files. After “OK” button is pressed, a new window containing the plot will pop up.

5. The users can also use multiple Y-axis data to plot an overlapping graph. For this, the users can input ".dat" files in the box pertaining to Y-Data Source1, Y-Data Source2 and so on.

Using the above functionality, we entered the .dat files to X-Data Source and Y-Data Source corresponding to pulse train and Figure 3.3 shows the result provided by our plot tools.

Figure 3.3: Pulse Train from the plot tool

3.3 Hierarchical Design

In this section, we will focus on the second task pertaining to hierarchical design. First, we will briefly explain the concepts and implementation of hierarchical with the aid of available software "Ptolemy" and explain our way of implementing it in our software.

The Optical Fiber Communication System (OFCS) can be a complex design involving more than 50 different components. It is difficult to manage large number of components in one window. So, organizing these components at various levels of granularity will make

the GUI more users-friendly. Therefore, we will develop hierarchical system models that contain components/subsystem that are themselves system models. Such components/subsystems can be called composite components. To obtain this hierarchical design we will develop a new component in the component in library. This component will be named as a composite actor. The functionality of the component is such that, when the users double-clicks this component, a new window is opened. This will be like the main window with all the functionality. The users can design the new subsystem as a composite component. The stored single composite module can have the users-defined name. User can save the component with the unique name and the new component will be saved in a user's library. When the components are saved, it can be used as a single module. The component will be like any other component with the input and output ports representing its interface to other components.

The concept of hierarchical design is present in most software tools such as Simulink, VPI, and Ptolemy etc. Here, we will first discuss the hierarchical feature of the Ptolemy to show how the existing simulation tools are using this feature. Figure 3.4 illustrates the hierarchical actor model of Ptolemy [5]. Here, A and C are the composite actor and consists of sub models. B is an atomic actor like our regular component. Note that composite actor is a composition of other actors including composite actors and/or atomic actors. The sub-component of A is linked to B and C via port p. Here the port p is an output port which links the output of D to the input of B and C. Similarly, A is linked with the subcomponent of C via the input port q. The port q connects A with C's subcomponent E. The Ptolemy uses this hierarchical approach to the model heterogeneous system. We will discuss how Ptolemy uses hierarchical approach for heterogeneous modeling.

Figure 3.4 shows we can connect various sub models to form a group of interacting actors. Ptolemy II constraints each level of the hierarchy to be locally homogeneous, using a common model of computation. A heterogeneous model thus can be created combining all these homogeneous models. By developing homogeneous system, we can select the best

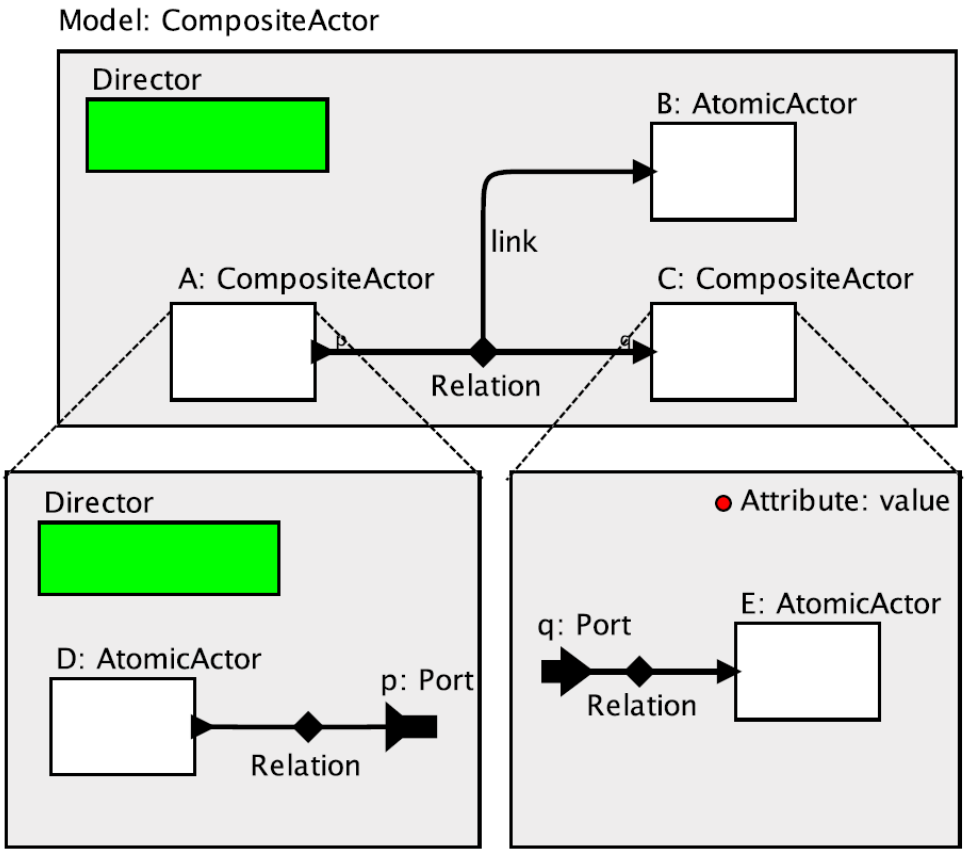


Figure 3.4: Hierarchical Design in Ptolemy

model of computation that matches the system's processing requirements. Furthermore, in Ptolemy II, a director determines how models are represented i.e. determines the semantics of a model. Figure 3.4 shows that two directors are present, one in the top level i.e. system containing A, B and C and the other is the lower level system containing block D. However, the subsystem of C doesn't contain a director. Therefore, the subcomponent of C, i.e. E as shown in Figure 3.4 is controlled by the top-level director which means the model inside

C is visible to the director of the top level. However, there is another director present inside A and this director governs the interaction of actors within the sub model. In Figure 3.4 actor D is controlled by the director present in the sub model. Ptolemy defines actors like A as an opaque composite actor and its content are not visible to top-level director. The top-level director treats actor A like any other model, although A contains model internally. These directors at different level can implement any model of computation and thus this facilitates for realizing hierarchical multi modeling and cosimulation in Ptolemy.

3.3.1 Usage of hierarchical model in Ptolemy

As explained in the above section, Ptolemy II consist the feature of hierarchical modeling. The hierarchical modeling is achieved by using models called composite actors [5]. We will discuss how the GUI of Ptolemy works in implementing the hierarchical models with an example of signal processing problem of recovering a signal from a noisy channel. First, we will use a composite actor to model a communication channel. Then this model will be used in larger models with other components. Figure 3.5 illustrates the working environment of Ptolemy with the composite actor in the main window.

The Ptolemy consists of CompositeActor in Utilities library. We can drag this component to main window. Then we can open a new window from the composite actor by selecting “Open Actor” as shown in Figure 3.5. The new window must consist of interface of input and output port to link with top-level layers. The port can be added by clicking on the white and black arrow head “port” buttons located on the toolbar as shown in Figure 3.6. Ptolemy provides option of varying the features of port such as renaming the port name. Furthermore, right clicking on background as shown in Figure 3.7 and selecting

"Ports" we can add ports, remove ports, or change whether a port is an input, output or a multiport.

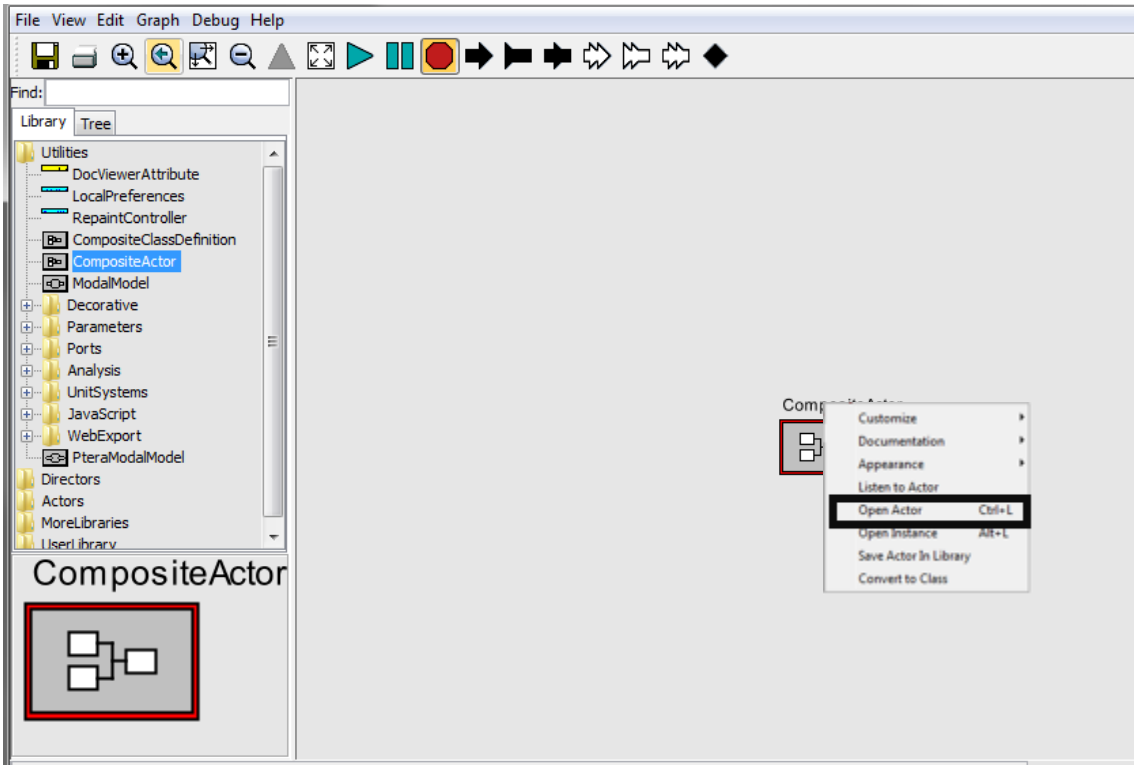


Figure 3.5: Developing composite actor in Ptolemy

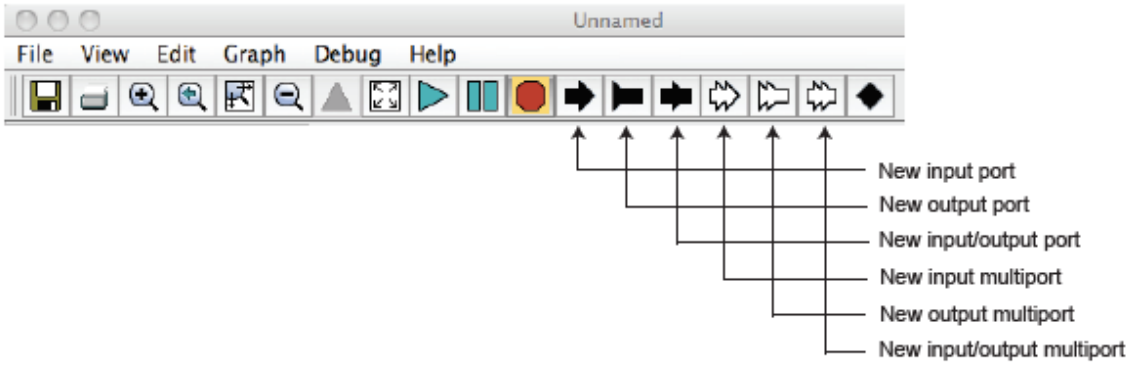


Figure 3.6: Tool panel of Ptolemy

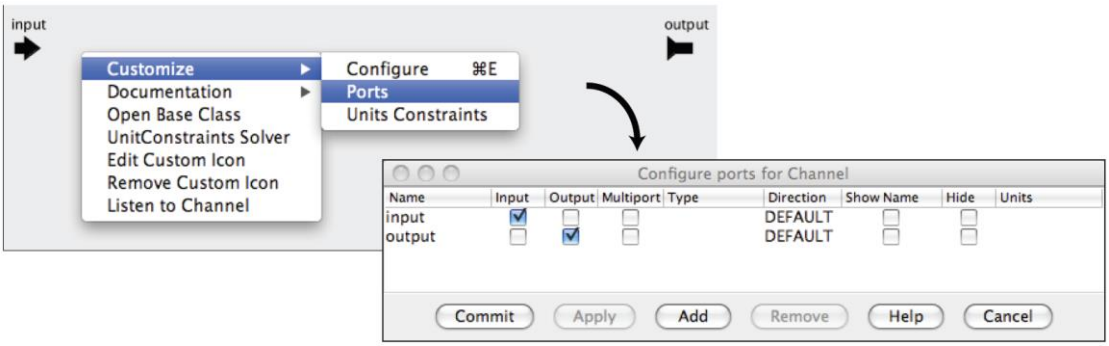


Figure 3.7: Configuration of ports in Ptolemy

With the addition of ports and components from the library a system we can create a sub model inside a composite mode. Figure 3.8 is an example where we create a simple channel with components such as AddSubtractor and Gaussian, with input and output ports connected to AddSubtractor. This is the model inside the composite actor named as the channel. Ptolemy also has a feature of setting the type of Port, and the type includes complex, double, fixed point, float, general, int, long, matrix, object, scalar, short, string, unknown, unsignedByte, xmlToken, arrayType(int), arrayType(int, 5), [double], and {x= double, y=double}. Now using the composite actor in our top-level design, a simple signal processing system can be created as shown in Figure 3.9.

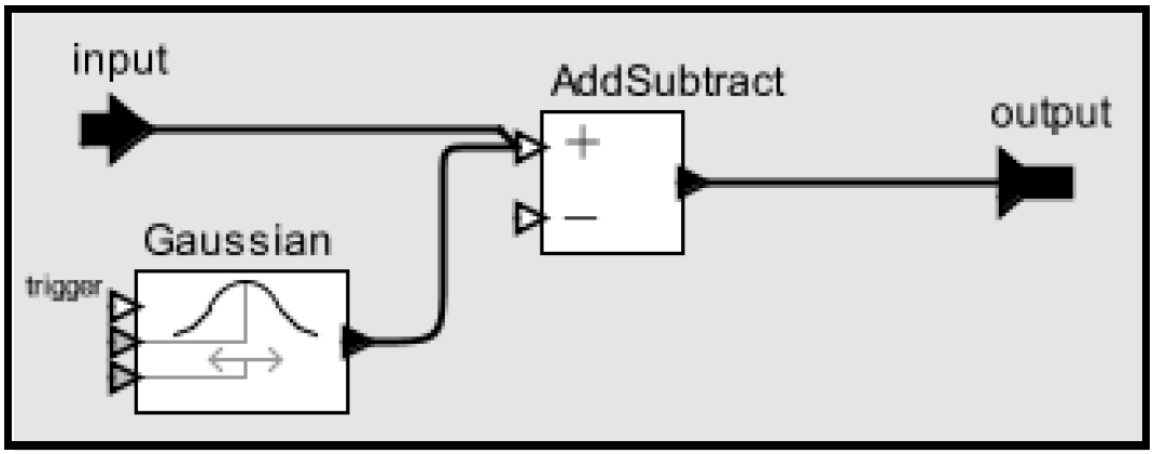


Figure 3.8: A simple channel model defined as a composite actor

In some cases, the composite actor might contain multiple input/output ports and this is also included in Ptolemy. Figure 3.10 shows the composite actor with multiple input ports along with information regarding how components and port are organized inside the composite actor. The capability of adding parameters to hierarchical models is another important feature that Ptolemy provides.

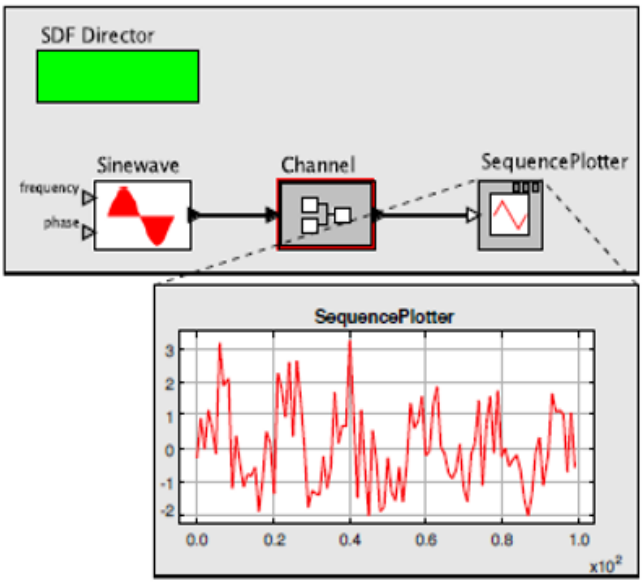


Figure 3.9: An example of simple signal processing

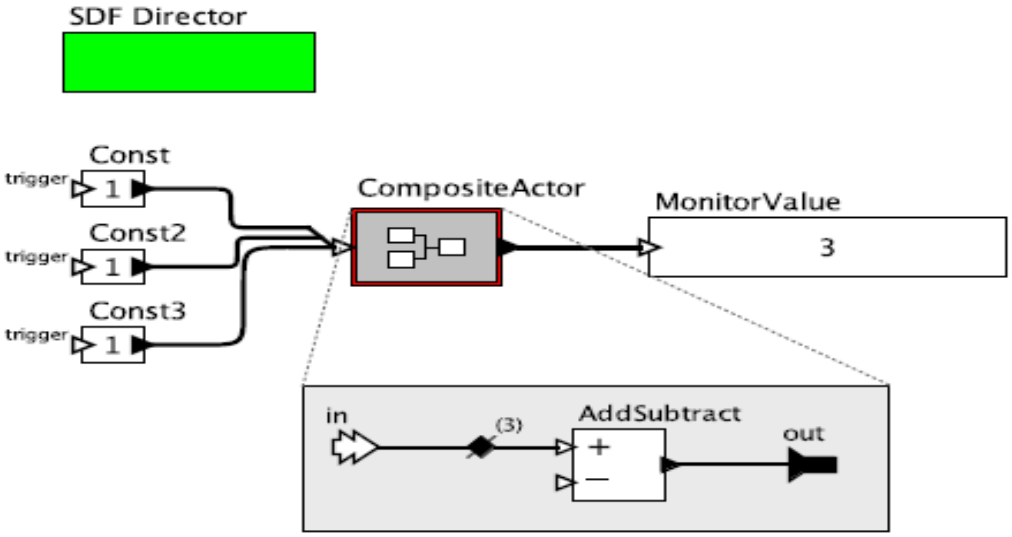


Figure 3.10: Usage of Multiport in Ptolemy

To add a parameter that sets the level of the noise in composite actor “channel”, we open the model channel and add a parameter component form Utilities library as shown in Figure 3.11. We can rename the parameter name to noisePower. The default value can be changed by double clicking on the parameter itself. Now we can use this parameter inside the parameter editor of Gaussian component as shown in Figure 3.11.

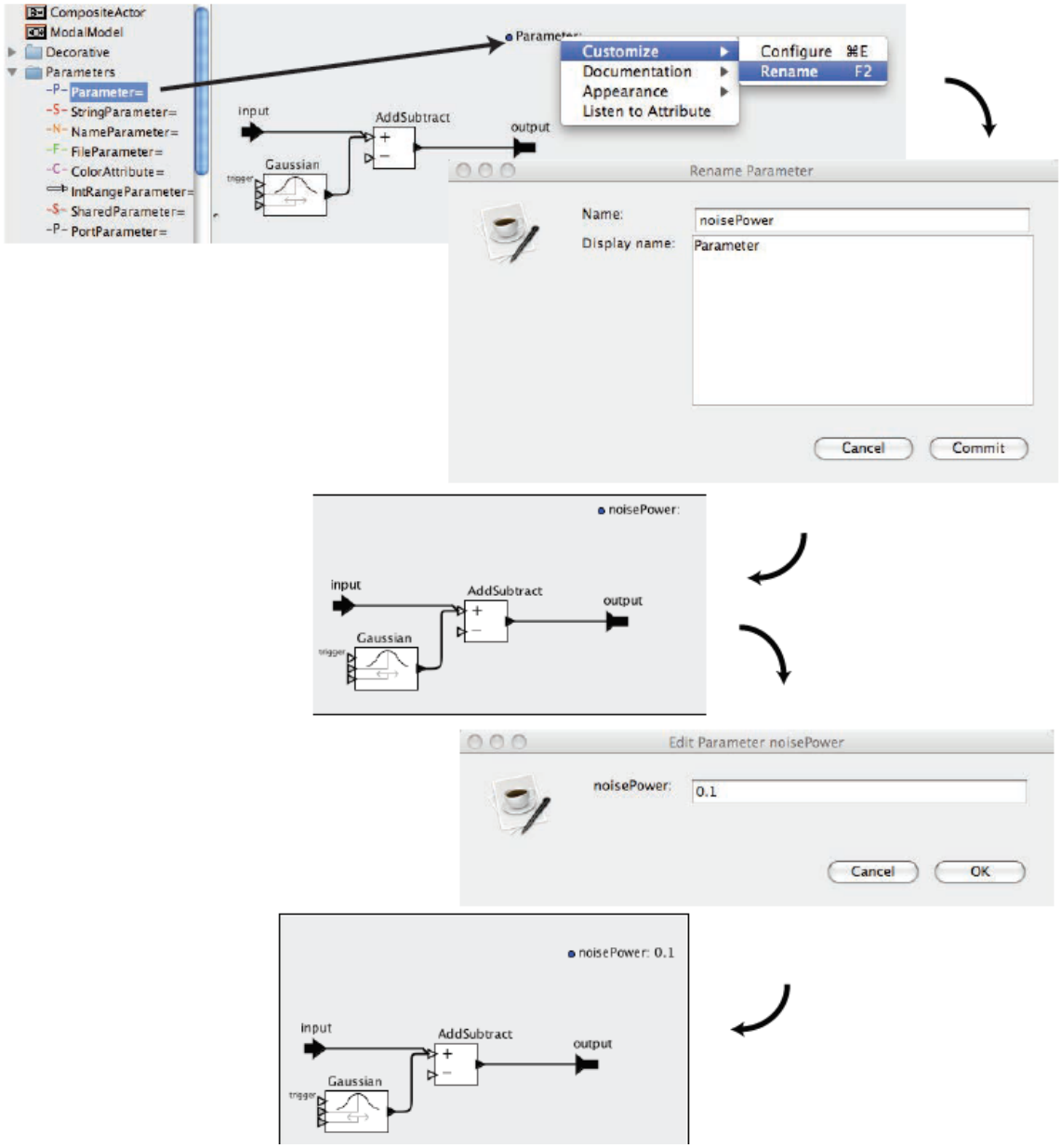


Figure 3.11: Adding a parameter to the Channel mode

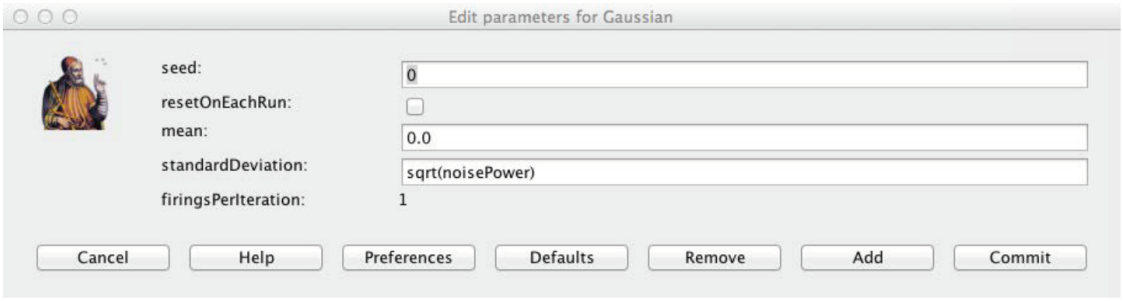


Figure 3.12: Parameter editor for Gaussians in Ptolemy

3.3.2 Hierarchical Design of Our Software

We have discussed the importance of hierarchical design and the implementation of it using a Ptolemy software as an example in the previous section. We also implement this feature in our software to make our GUI more users-friendly. In our software, an additional library component named as "Composite" is added in the upper panel as shown in Figure 3.13. This is similar to the other buttons in the upper panel. When the users click this button a list of the component is displayed in the lower panel. The components inside this library are "composite actor", "input port 1", "input port 2", "input port 3", "input port 4", "output port 1" and "output port 2" as shown in Figure 3.14.

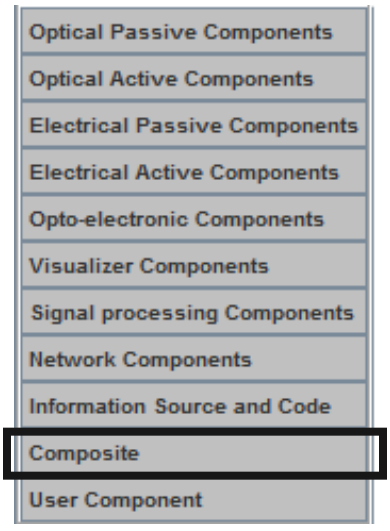


Figure 3.13: Addition of Library in our software

Here we have limited the number of the input port and output port to a fixed number but a flexible input and the output port which can be extended to any number can be developed in future. The composite component can be dragged to the main window as any other components. When the users double-click this component after placing it in the window a new window is popped up as shown in Figure 3.14. Here the users can design the system using all the components from the library as shown in Figure 3.15. Once the design is completed the users can save their system with any name as shown in Figure 3.16. The saved component will be saved in a library named under "User Component" as shown in Figure 3.17. The users can then go to the "User component", find the component with saved name and use the component in their system design. The global parameter must be common to both the second-level components and top-level components. However, we can make these parameters independent, which can be performed as a further update of the software.

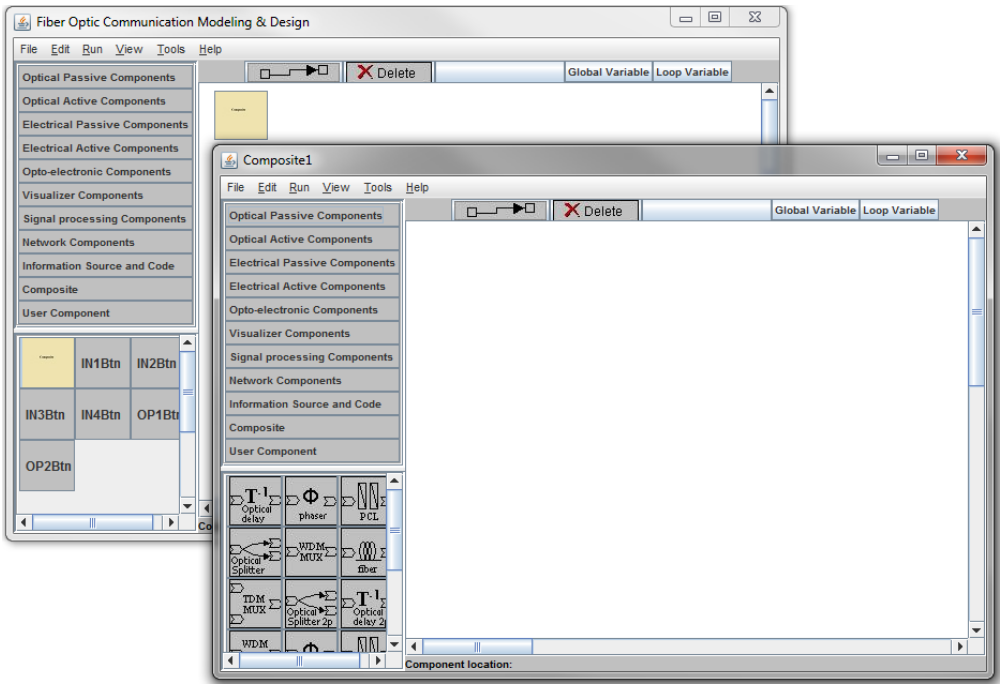


Figure 3.14: New window appears when composite actor is double clicked

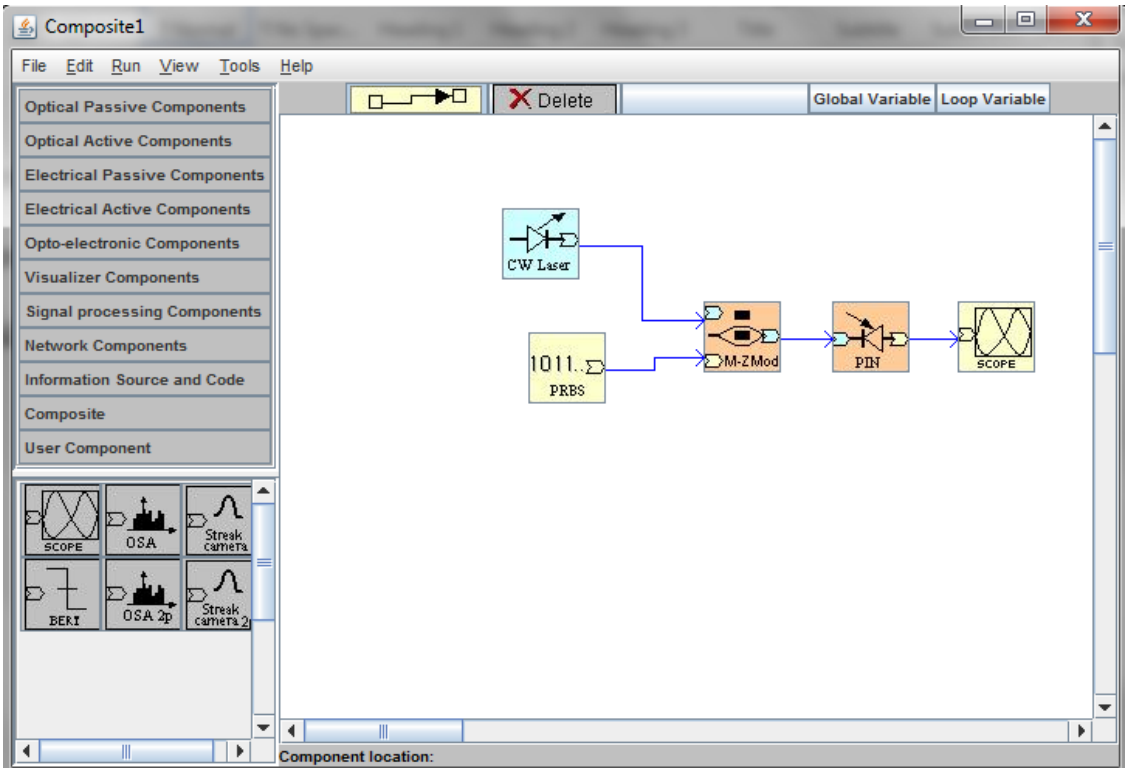


Figure 3.15: Sub system in composite actor

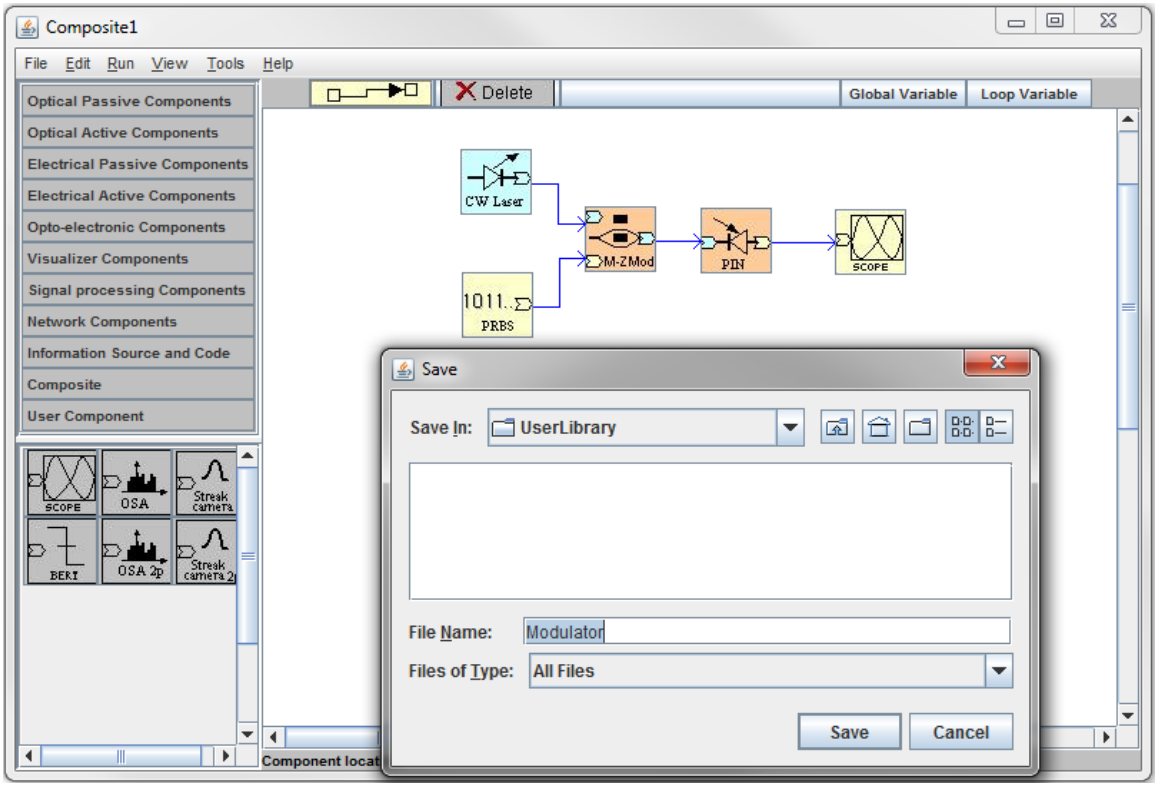


Figure 3.16: Saving the composite actor

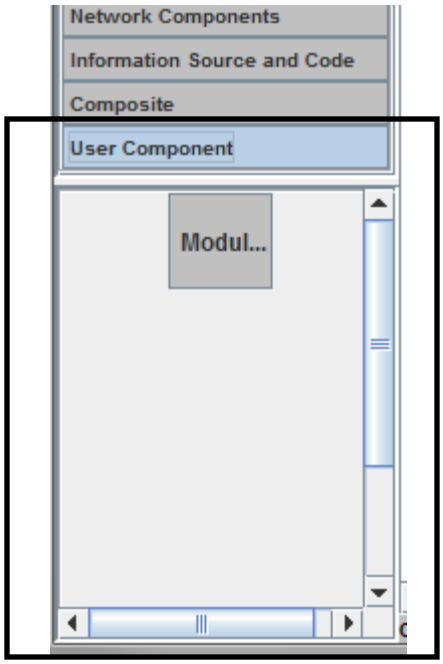


Figure 3.17: User defined components in User component library

3.3.3 Consideration for Development of Hierarchical

The development of Hierarchical design consists of various tasks. In the following we explain the four major tasks that we worked to obtain hierarchical design.

1) Modification of GUI

The GUI is modified to accommodate multiple windows at once. Without major changes to the previous code of the software, a module of code is added to update the library with the new component named “Composite”. Similarly, the “Composite” button is provided with the actionlistener and the action performed is such that the list of the components in this library are displayed in low panel. The components in low panel are like other components in low panel i.e. they all consists of actionlistener and mouselistener such that they can be handled by mouse and dropped in the window. When the component is dropped in the window, the action execution of the composite actor is different than other normal components. When we double-click other components a parameter editor is opened but for

this double-click will open the new window where we can design a sub model. This modification of action is handled by one of the method of mouselistener. The action defined in this method is such that a new window is opened every time the users double-click this button. However, a restriction is applied such that only one window can be opened by double-clicking the composite component. The users can double-click and open a new window if and only if there is no other window opened corresponding to that composite component. Also, the closing of new window will not affect the top-level window. The design in the hierarchical model consists of input and output port as discussed in previous section, to make an interface between multiple levels. These ports are important because they link various level of design together. Once the system layout is designed, the model is saved as an .xml file in a folder named as "UserLibrary". When the model is saved, the new created model automatically appears in the lower panel. Further, the component from the lower panel can be dragged to the window and can be used as any other components.

2) Rearranging the Sorting Algorithm

Previously, the arrangement of the components was only performed for one level. The algorithm to arrange the component only dealt with the component in one window, as they were arranged as per the execution order of the component. The algorithm for arranging component as per the execution order still holds but with the addition of the arrangement of new components when composite components are placed in the design. For example, for the system shown below in Figure 3.18 (a): the execution order of the system should be A, B and C. However, if B is a composite component, then the execution order should also include the internal components of the B. Figure 3.18 (b) shows the internal components

of B. Including the internal components of B, the final execution order for the simulation shall be A, X, Y, Z, and C.

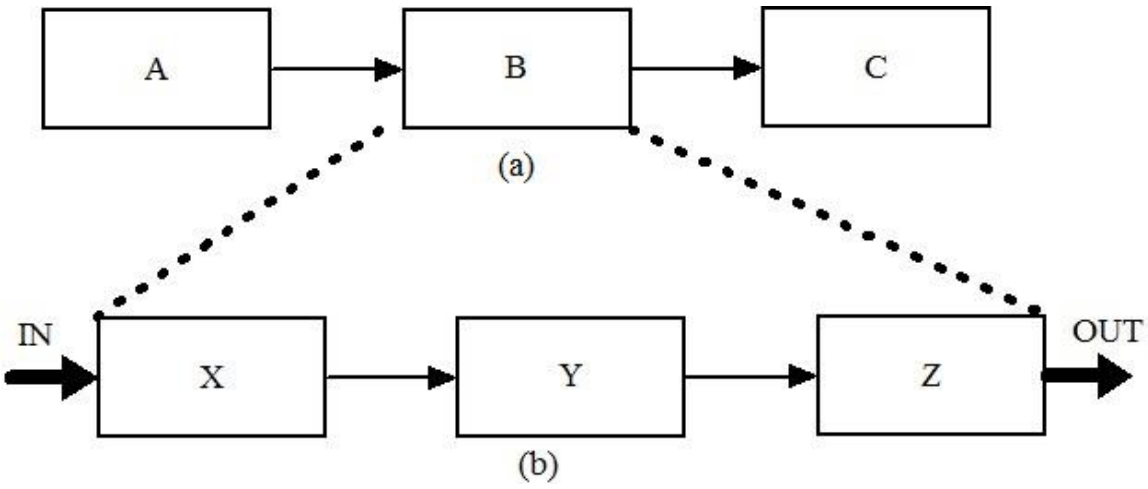


Figure 3.18: (a) Block representation of component connection with B as a composite component (b) Sub-model inside composite component B

3) Connection between two levels

When composite components are placed a new level is added to the system. To run the simulation, a relation between these levels should be established. All the components from the top level (1st level) to the 2nd level and so on should be set in a correct executing order. Here, Figure 3.18(a) represents the top-level system, where Figure 3.18 (b) represents the second level system. As mentioned in the previous section that the execution order for simulation should include components of both level in correct order. The key to the arrangement is the port as shown in Figure 3.18 (b) named as IN and OUT. The sorting algorithm identifies these input and output port and builds the connection between two levels. First, it checks the level of the window, and if it finds the level is greater than 1, it looks for the input port and output port to establish the connection of the second level with first level components or in general it establishes the connection of n+1 level with n level. Here, B is the composite component and consists of the second level window, therefore,

the input ports and output ports of this second level will be linked to the top-level components A and C respectively. This results in linkage of component A of the top level with component X of the second level. Similarly, the output port will link with component Z of the second level with the component C of the top level. If any of the components in the second level is composite, for instance, say X is a composite then similar process is performed between 2nd level and 3rd level using the input and output port of the 3rd level system. Then, the result of the combination of 2nd and 3rd level is synchronized with the top-level system.

4) Storing the system design as a model

The users need to save the internal system design of composite component to be able to use their composite component. These models are saved as the XML file. The users need to save them in the specific folder named as “UserLibrary” such that the model appears under the “User Component” library. All the information pertaining to the system component, arrow connection, parameters are saved in the XML file. When these components from the “UserLibrary” are double-clicked after placing on the window, the XML file is uploaded. This functionality is like opening our saved XML file in our previous version of the software.

Chapter 4

Summary and Future Work

In this chapter, we will summarize thesis and explain possible future work for the Graphical User Interface (GUI) software. First, we will briefly summarize all the chapter of this thesis starting from chapter 1.

4.1 Summary

In chapter 1, we introduced the system model and the importance of this tool in the evaluation of the performance of systems. We then discussed several types of models as per different criterion of the system such as determinism or system dependency on time. Furthermore, with Maxwell's equation, we showed how the abstraction helps to develop a simplified model. We derived some simpler algebraic equations representing Ohm's law and Kirchhoff's voltage and current law from the complex Maxwell's equation. After the description of the model, we explained Electronic Design Automation (EDA) tools that implement various kind of electrical system model for simulation purpose. The evolution of the circuit simulation tool from SPICE models to currently EDA tools was also described in the chapter. In addition, PDA tools evolution and its content were briefly explained. This was followed by the introduction of our software and the objectives behind the development of our software. Finally, with consideration of upgrading our software to model Cyber Physical System (CPS) in future, we explained the concepts of CPS.

Chapter 2 consists detail description of the GUI of our software. The GUI was explained in two phases, first, the component and graphical layout aspect of the GUI was explained which was then followed by the interactive features of GUI. For the graphical layout description, we divided our GUI into numerous section and explained each in detail. The detail illustration of graphics of GUI was preceded by the explanation of some major classes of java that were used in developing the graphics of our GUI. The interactive features were then explained, initially with the illustration of the interface of Java that we used to make our components in GUI interactive. Each action pertaining to click, drag, double-click etc. was described in detail for each component. Once the interactive features were explained with respect to a system layout design, we described the saving and loading features of our software. Furthermore, we briefly provided an insight of the sorting algorithm used in our software. Finally, we concluded the chapter 2 with an example of the simulation of the optical communication system performed by our software.

After the description of our software in chapter 2, we proceeded with the illustration of the task we performed to update the software in chapter 3. In chapter 3, we described two major works done in our software. First, we explained about the integration of plot tools with a brief description of the JFreechart library. Some class of JFreechart library was discussed to provide an overview of the class that we used in the development of the plot tool. Second, we explained the concept of hierarchical design. For this, we used the software Ptolemy to describe how the hierarchical design is implemented and used in a simulation software. Once we established the concept of hierarchical design, we explained how our hierarchical design works and how we updated our GUI to accommodate this feature in our software.

4.2 Future Work

Our software currently can perform the simulation of complex optical communication system this feature. However, there are many updates we can perform to make our software more robust and comprehensive. In this section, we will discuss numerous task that we can perform to expand our software functionality.

In our current GUI, when components are placed and connected by the arrows, we can move the component but the arrow connection does not move with the component. The future work will require an update in the code so that the component and the arrow connections are hooked together in every instance. This will make arrow connection follow the movement of the component. This will make the use of software easy as the users can readjust their system layout by moving the components. Furthermore, in our "Edit" menu although we have provided various edit options, currently no action is provided to this menu item such as "Undo", "Redo", "Cut", "Copy", "Paste" and "Delete". We need to update the code of GUI to provide each menu item an action event handler such that they can perform the action as per their name suggest.

In addition to the update of graphics, we also need to expand the component library significantly, as the board field of the Optical Fiber communication system (OFCS) covers both Photonics and Communications areas and comprise a large number of components. This represents a major task for the server code along with few updates in GUI. In GUI, the upper panel and lower panel needs to be updated to accommodate the new library and new components. In the server side, simulation models and mathematical solutions need to be investigated and server codes need to be developed for new components. The server program will include a complete set of design libraries listed below:

- Information sources and coding: PRBS sequence generation; Debrjun sequence generation, differential encoder/decoder
- Noise sources: Gaussian noise generator, noise generator with arbitrary probability design functions
- Electrical/Photonic components: lasers, driving circuits, TIAs, modulators, detectors, splitters, couplers, polarization devices, filters, WEDM multiplexers and de-multiplexers
- Subsystem: transmitter, amplifier (EDFA and Raman amplifier), receiver, PMD compensator, OADMs
- Link budget: OSNR, gain equalization, pre-emphasis
- System: signal modulation format, fiber channel simulation, BER and Q evaluation suing large deviation methods as well as quasi-analytical methods, dispersion map, PMD and PDL, amplifier transient behavior and hole burning, channel equalization by electronics and optics
- Visualizes: digital oscilloscopes, optical spectrum analyzers, streak cameras
- Signal processing module/blocks: pulse generators, FFT and inverse FFT methods, filtering methods, synchronization functions
- Channel coding and decoding blocks

Another, significant feature that we need to implement in our software is the feedback system. The advanced signal processing involves feedback and is a vital component in modern communications. We can use these advanced signal processing to improve the OFCS performance, and thus to simulate such system our software must incorporate the

functionality of feedback. Figure 4.1 is an example of the feedback loop topology that we intend to add in our software.

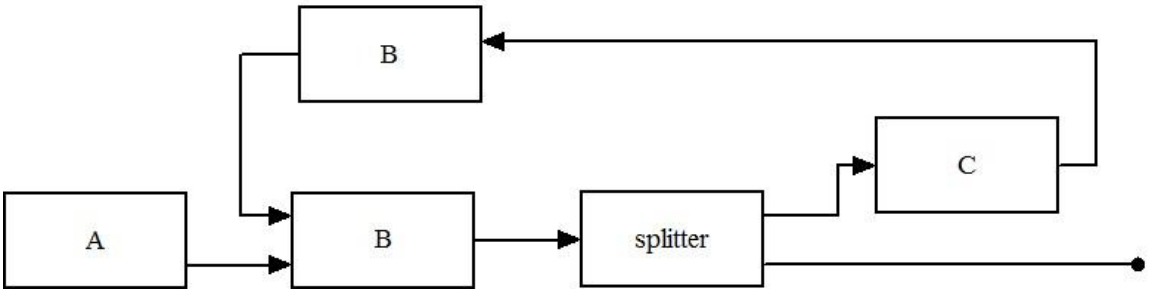


Figure 4.1: Illustration of detailed feedback loop layout of proposed composite feedback component

Finally, with the addition of various components and extension of the functionality of our software with feedback systems and other required features we can make our software capable of dealing with a heterogeneous model to simulate cyber-physical systems.

References

[1] A. Maria, “Introduction to Modeling and Simulation,” in *Proceedings of the 29th Conference of Winter Simulation*, pp 7-13, 1997.

[2] E. A. Lee, “Fundamental Limits of Cyber-Physical Systems Modeling,” *Journal of ACM Trans. Cyber-Phys. Syst.*, vol.1, issue 1, article 3, November 2016.

[3] J. Liu, E.A. Lee, “Component-based hierarchical modeling of systems with continuous and discrete dynamics,” in *Proceedings of IEEE symposium on Computer-Aided Control System Design*, pp 95-100, 2000.

[4] M.I. Jordan, “Hierarchical Models, Nested Models and Completely Random Measures,” University of California, Berkeley, 2013.

[5] Claudius Ptolemaeus, Editor, System Design, Modeling, and Simulation Using Ptolemy II, Ptolemy.org, 2014.

[6] Herbert Parehofer, “Object Oriented, Modular Hierarchical Simulation Modeling: Towards Reuse of Simulation Code,” in *System Practice and Theory*, vol. 4, issue 4, pp 5-10, 1996.

[7] S.S Bhattaacharyya, E. Cheong, J. Davis II, M. goel, B. Keinhuis, C. Hylands, E.A. Lee, J. Liu, et al, “Heterogeneous Concurrent Modeling and Design in Java,” vol. 2

[8] J. Banks, “Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice,” John Wiley & Sons, Inc., 1998.

[9] S. A. Edwards and O. Tardieu, “SHIM: a deterministic model for heterogeneous embedded systems,” in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 8, pp. 854-867, Aug. 2006.

[10] P. Lachor, K. Puszyński, A. Polański, “Deterministic models and stochastic simulations in multiple reaction models in systems biology Journal of Biotechnology, Computational Biology and Bionanotechnology, vol. 92, no. 3, pp. 265-280, 2011

[11] R. V. Field, “Stochastic models: Theory and simulation,” *Technical Report SAND*, Sandia National Laboratories, 2008.

[12] A. M. Law, W.D. Kelton, "Simulation modeling and analysis," 4th edition, MCGraw-Hill, 2008.

[13] M. Olifer, "Architectural model synthesis from source code using Simulink and Hierarchical Function Call-Graphs," Dissertation, 2016.

[14] C.R Harrell, "Simulation Using ProModel," 3rd edition, McGraw-Hill Education, 2011

[15] A. B. Sharma F. Ivancic A. Niculescu-Mizil H. Chen G. Jiang "Modeling and analytics for cyber-physical systems in the age of big data," *ACM Sigmetrics*, 2013.

[16] G. Karsai, "Modeling Cyber-Physical Systems: Challenges and Recent Advances," Seminar at U Conn., 2015

[17] P. Derler, E.A. Lee, A. Sangiovanni-Vincentelli, "Modeling Cyber-Physical Systems," in *Proceedings of the IEEE*, vol. 100, pp. 13-28, 2012

[18] E. A. Lee, S. A. Seshia, "Introduction to Embedded Systems - A Cyber-Physical Systems Approach," 2nd edition, MIT Press, 2017.

[19] B. P. Zeigler, "Object-Oriented Simulation with Hierarchical, Modular Models-Intelligent Agents and Endomorphic systems," Academic Press, 1990

[20] S. Narayanan, D.A. Bodner, U. Sreekanth, T. Govindaraj, L.F. McGinnis, C.M. Mittchell, "Research in object-oriented manufacturing simulations: an assessment of the state of the art," in *IIE Transactions*, vol. 30, no. 9 pp. 795-810, 1998

[21] H. Kocher, M. Lang, "An object-oriented library for simulation of complex hierarchical systems," in *Proceedings of the Object-Oriented Simulation Conference*, pp. 145-152, Jan. 1994

[22] L. Weng, K. C. Tan, "Modern Industrial Automation Software Design – Principles and Real-World Applications," IEEE Press, A John Wiley & Sons, Inc.

[23] E.A. Lee, "Cyber Physical Systems: Design Challenges," *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, Orlando, FL, 2008, pp. 363-369.

[24] R.K Mishra, B. Lohani, "An Object-Oriented Software Development Approach to Design Simulator for Airborne Altmetric Lidar," 2007

[25] R. Brayton, J. Cong, "NSF Workshop on EDA: Past, Present, and Future (Part 2)," in *IEEE Design & Test of Computers*, vol. 27, no. 3, pp. 62-74, May-June 2010.

[26] J. Franz, V. Jain, Optical Communications – Components and Systems, CRC Press

[27] S. J. Orfanidis, Electromagnetic Waves and Antennas, Online, 2011, retrieved from http://www.ece.rutgers.edu/~orfanidi/ewa/ewa-1up.pdf

[28] A. Agarwal, J.H.Lang, Foundations of Analog and Digital Electronic Circuits, Elsevier, Morgan Kaufmann, 2005

[29] L. Chrostwoski, J. Flueckiger, C. Lin, M. Hochberg, J. Pond, J. Klien, J. Ferguson, C. Cone, "Design methodologies for silicon photonic integrated circuits," in Proceedings Smart Photonic and Optoelectronic Integrated Circuits XVI, vol. 8989, 2014.

[30] T. Korthorst, R. Stoffer, "Integrated Photonic Design Flow Automation," Phoenix Software, 2012.

[31] R. Rajkumar, I. Lee, L. Sha and J. Stankovic, "Cyber-physical systems: The next computing revolution," Design Automation Conference, Anaheim, CA, 2010, pp. 731-736.

[32] D. Riccardi, P. Novellini, "An Attribute-Programmable PRBS Generator and Checker," Application Note: Xilinx FPGAs

[33] G. Breed, "Bit Error Rate: Fundamental Concepts and Measurement Issues," High Frequency Electronics tutorial, Jan 2003.

[34] W.H. Hayt. J A. Buck, "Engineering Electromagnetics," 8th edition, Raghothaman Srinivasan

[35] K. S. Kundert, "The Designer's Guide to SPICE and SPECTRE," Kluwear Academic

[36] L. Pavesi, "Silicon Photonics III: Systems and Applications, " Springer

[37] Q. Zhang, "GUI based Computer Modeling and Design Platform to Promote Interactive Learning in Fiber Optic Communications," *2007 37th Annual Frontiers In Education Conference - Global Engineering: Knowledge Without Borders, Opportunities Without Passports*, Milwaukee, WI, 2007, pp. S3H-14-S3H-19

[38] http://cyberphysicalsystems.org/

[39] Java documentation, Java™ Platform, Standard Edition 7 API Specification retrieved from https://docs.oracle.com/javase/7/docs/api/

[40] C.S Horstmann, G. Cornell, "Core Java: Volume I Fundamentals," 9th edition, Perentice Hall, 2012

[41] D. Gilbert, "The JFreechart Class Version 0.9.1: Reference Document," Simba Management Limited, 2002

[42] JFreechart: chart java chart library, retrieved from https://www.tutorialspoint.com/jfreechart/jfreechart_tutorial.pdf