System Level Simulator
High Level Design

Title	System Level Simulator - High Level Design
Document Number	
Author	Minjoong Rim
Creation Date	September 28, 2016
Last Modified	Feburary 11, 2019
Version	1.0
Status	Working
Path To File	

Modified By	Date	Version	Reason For Change
Minjoong Rim	9/28/16	0.01	Created
Minjoong Rim	10/2/16	0.1	Initial version
Minjoong Rim	10/4/16	0.11	Minor modification to initialization process
Minjoong Rim	10/10/16	0.12	Add link level simulator
Minjoong Rim	10/14/16	0.13	Modification to link level simulator
Minjoong Rim	11/17/16	0.2	Split into Requirement Spec and High Level Design
Sangjin Cho	1/30/17	0.21	Modification to source code part in system level simulator
Minjoong Rim	3/1/17	0.22	Modification to simulation main
Minjoong Rim	5/23/17	0.3	Split into High Level Design and Skeleton Design
Minjoong Rim	8/2/17	0.4	Minor modification
Sangjin Cho	9/25/17	0.5	Minor modification
Minjoong Rim	10/11/17	0.6	Minor modification
Minjoong Rim	10/30/17	0.7	Add calibration process
Sangjin Cho	1/11/18	0.8	Add UML diagram
Minjoong Rim	9/8/18	0.9	Minor modification
Jaewon Lee	2/11/19	1.0	Integration with module design documents

Table of Contents

List of Figures

List of Tables

1. Introduction

This document describes the high level designs for 5G System Level Simulator, including the module design.

1.1 Purpose

This document is intended for use by software engineers working directly on the system level simulator.

1.2 Scope

The scope of this document includes the following:

- Design process
- Architecture philosophy
- Simulator structure
- Module design

1.3 Definitions

The following definitions are used for the simulator and this document.

Table 1-1 Definitions

Word	Description
BS	Related to cell Refers to sector (NOT base station site), TRP(transmission reception point), RSU(road side unit), RRH(remote radio head), RU(RF unit), or RRU(remote radio unit) We will not use the terms TRP, RSU, RRH, RU, or RRU in this simulator.
MS	Refers to mobile station, user, device, UE(user equipment), UT(user terminal) We will not use the terms UE or UT in this simulator.
Site	Related to base station position Base station, site, eNode-B, eNB, gNB We will not use the terms eNode-B or eNB in this simulator.
BBU	Baseband unit, DU(digital unit) We will not use the term DU in this simulator.

1.4 Acronyms

The following acronyms are used for the simulator and this document.

Table 1-2 Acronyms

Acronym	Description
MS	Mobile Station
BS	Base Station
RS	Relay Station
BBU	Baseband Unit
RX	Receiver
TX	Transmitter
DL	Downlink
UL	Uplink
FDD	Frequency Division Duplexing
TDD	Time Division Duplexing
TTI	Transmission Time Interval
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
RB	Resource Block
ARQ	Automatic Repeat Request
HARQ	Hybrid ARQ
ACK	Acknowledgment
NACK	Negative Acknowledgment
LOS	Line of Sight
NLOS	Non Line of Sight
SNR	Signal to Noise Ratio
SINR	Signal to Interference plus Noise Ratio
ESM	Effective SINR Mapping
FER	Frame Error Rate
CDF	Cumulative Distribution Function
V2X	Vehicular to Anything
D2D	Device to Device Communication
IoT	Internet of Things
CoMP	Coordinate Multi Point
MIMO	Multiple Input Multiple Output
FD-MIMO	Full Dimensional MIMO
MU-MIMO	Multi-User MIMO
ULA	Uniform Linear Arrays
CDL	Clustered Delay Line
TDL	Tapped Delay Line
AoD	Azimuth angle of departure
AoA	Azimuth angle of arrival
ZoD	Zenith angle of departure
ZoA	Zenith angle of arrival
MAC	Media Access Control
RRM	Radio Resource Management
GUI	Graphical User Interface
SLS	System Level Simulation, System Level Simulator
LLS	Link Level Simulation, Link Level Simulator
NS	Network Simulation, Network Simulator

sim	Simulation, Simulator
config	Configuration
param	Parameter
pos	Position
p	Pointer

2. Design Process

2.1 Software Design Process

2.1.1 Typical Software Design Process

A typical software top-down design process is shown in the following figure.

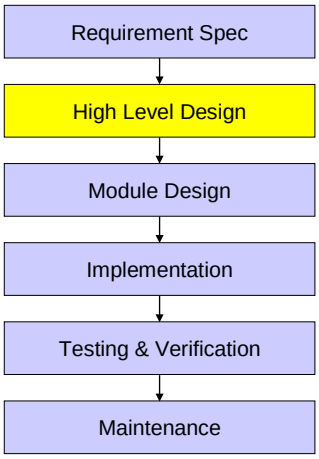

Figure 2-1 Typical software design process

After high level design is complete, low level designs or module designs can be started.

2.1.2 Software Design Process for This Project

A top-down design process can be implemented with incremental modification to requirements and high level designs.

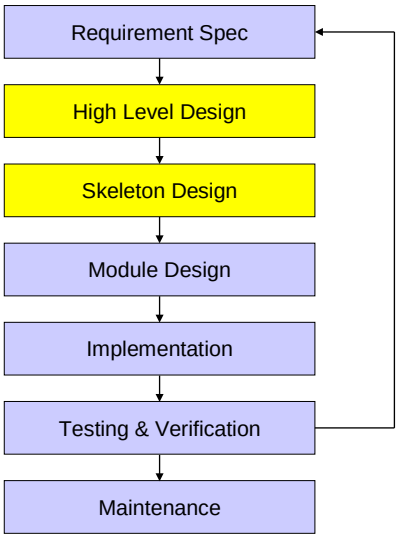

Figure 2-2 Software design process for this project

While a high-level design is followed by a module design in a typical design process, a skeleton design step is included in this project between the high-level design and the module design steps. The purposes of the skeleton design include the following:

- Verify design concepts
- Help software engineers to easily understand the design philosophy
- Help software engineers to easily understand the module concepts more clearly
- Help software engineers to easily follow the coding style

2.2 Coding Style

2.2.1 Programming Language

C++

Compiler: Visual Studio

However, the simulator will not be dependent on the compiler except some special features such as parallel processing parts.

2.2.2 Naming Convention

The following naming convention will be used for the simulator. The naming convention used in this simulator is based on CamelCase with some modification. Note that '_' is used for special purpose and should not be used in the middle of a variable without special meaning.

Table 2-1 Naming Convention

Type	Naming Convention	Example
Local Variable	Start with a lowercase letter Words are joined without spaces and capitalized	noiseFigure
Global Variable	Start with a capital letter The number of Global variables should be limited	Sim
Class	Start with a capital letter	Scheduling
Acronym in the middle	Do not use all capital letters	numMsPerMacro
Acronym at the end	Capital letters are allowed	numMS
Plural form	Plural form is not used	numMS
Local Type Definition	Start with a capital letter	DataType
Global Type Definition	File name + _ +	SystemSim_Scenario
Local Constant	All capital letters Words are joined with under bar	BPSK
Global Constant	File name + _ +	SLS_MAX_BS
Local Function	Start with a capital letter	Initialize()
Global Function	Class name + . + Start with a capital letter If there is no class in the file, File name + _ + Start with a capital letter	Scheduling.Initialize() Scheduling_Initialize()
Special Function	File name + _ + All capital letters	DEBUG_PRINT()

2.2.3 File Layout

An example of file layout can be found in "Directory Structure".

2.2.4 Code Layout

An example of code layout can be found in the documentation called "SLS Skeleton Design".

2.3 Simulator Validation

2.3.1 Calibration

In a system-level simulator, the simulation results can be affected by technology details of module implementation and algorithms, which cannot be adequately defined as simulation parameters. In order to ensure the validity of the results obtained from a system-level simulator, it is necessary to conduct a calibration process after the development phase. For this process, it is important to define a calibration process for the key building blocks of system level

simulation, and outcomes of different simulators are cross-checked according to the calibration process to detect incoherencies. If the calibration process is well defined and the results of a simulator are aligned with those of other simulators, the simulator is considered properly calibrated. Otherwise, corresponding modules need to be corrected and implementation details need to be redesigned so that all results are aligned.

A stepwise calibration methodology was followed in the framework defined in 3GPP, where the calibration processes are well defined, and the simulation results of channel model and baseline simulation verification are provided by a large number of companies and institutions. The calibration process consists of channel calibration and baseline calibration. The channel calibration verifies the implementation of the channel model defined in 3GPP for LTE-Pro and 5G systems [TR 36.783, TR 36.900, TR 36.901], and is composed of two steps: the calibration of the large scale parameters of the channel model and the calibration of the small-scale parameters of the channel model. The baseline calibration process performs calibration of baseline system configuration with focusing on the calibration of MAC layer and link adaptation model that are technology dependent. For more details, refer to [3-5].

2.3.2 Channel Calibration

There are two different channel models in 3GPP: below-6GHz [TR 36.873] and above-6GHz channel models [TR 38.900]. Commonly, channel calibration is performed on two steps: calibration of the large scale parameters of the channel model and the calibration of the small-scale parameters of the channel model in a given network structure. After dropping base stations and terminals, reference signal received power (RSRP) is calculated and the serving base station is determined for each terminal through cell association. Then, cumulative distribution function (CDF) is calculated for coupling loss, signal to interference plus nose ratio (SINR), and Eigen-value. This allows a simulator to perform validation of channel model implementation according to key parameters that affect the system performance. In addition, for above-6GHz channel model, further calibration process necessary to reflect mmWave characteristics including additional features such as oxygen absorption, blockage, and so on.

2.3.3 Baseline Calibration

The baseline calibration is a system-level simulation verification process that performs performance evaluation with radio resource management (RRM). Assuming that the verification of the PHY model is completed with the channel calibration process mentioned above, the baseline calibration verifies MAC layer implementation and link adaptation model. For this process, a simulator needs to implement RRM functions based on common simulation assumptions. For example, HARQ management, link adaptation, scheduling, traffic load can be implemented based on the most basic PHY model. The performance verification is performed typically through spectral efficiency, user throughput, and SINR distributions.

3. Architecture Philosophy

3.1 Modular and Flexible Structure

3.1.1 Modularity

The system level simulator supports high levels of readability and consistency, and provides modularity and flexibility for users to easily modify and extend the simulator. In order to satisfy this, the object concept in C++ is used, and modules can be activated or deactivated by a file called the simulation configuration file. The operation of the simulator can be changed with the followings:

(a) Simulation parameters

A different simulation parameter set can be applied by modifying or replacing simulation parameter files.

(b) Main function

There can be multiple main functions and different procedures of simulations can be supported by selecting different main functions.

The system level simulator includes several simulator mains as well as many modules. Each simulator main selects and uses some combination of modules according to its purpose and scenarios.

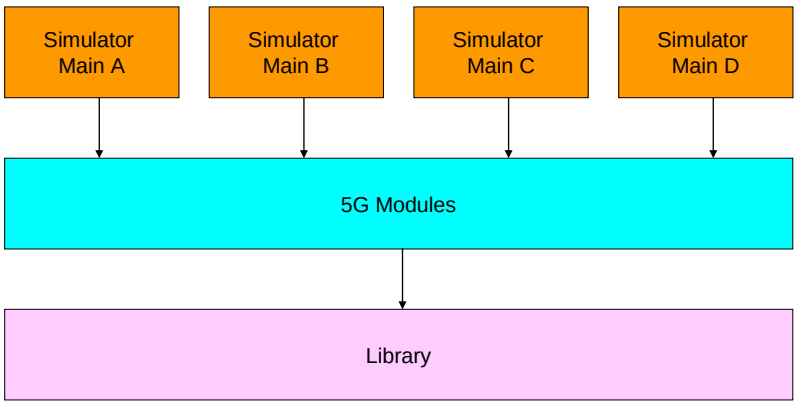

Figure 3-1 Multiple main functions

(c) Simulation configuration file

A different type or version of a simulator can be constructed by replacing some modules with others using the simulation configuration file. There can be multiple files including classes with the same name and an appropriate module can be selected according to simulation scenarios.

The following figure shows an example to illustrate how the simulator can change its operation. In the example, the simulator includes four choices of simulation main, three versions for each module, and four different sets of simulation parameters. In the figure, Simulation main 2, Module A1, Module B1, Module C2, Module D3, and Simulation parameter set 3 are selected to simulate a certain scenario.

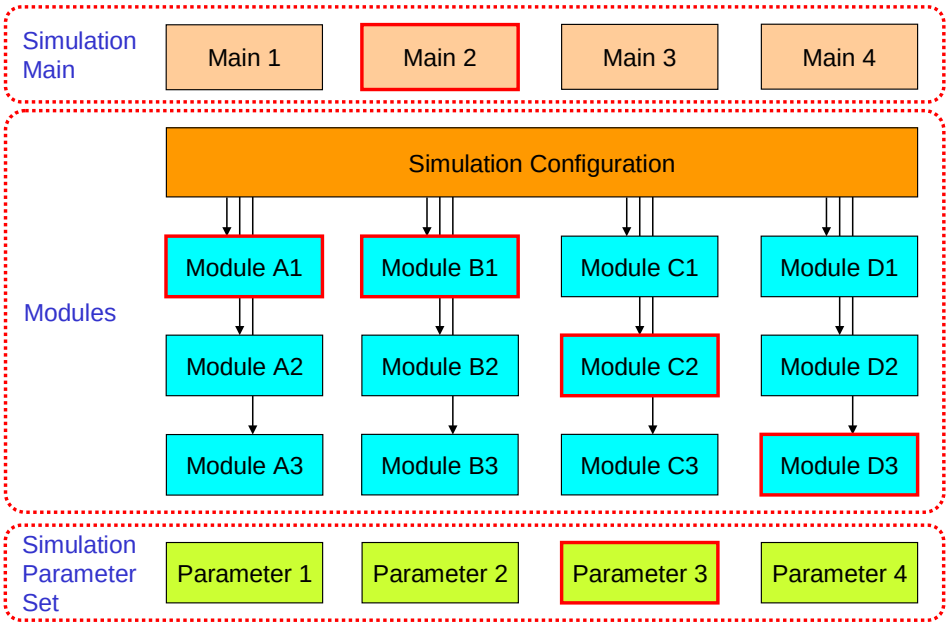

Figure 3-2 Simulation configuration

3.1.2 Multiple Files for a Single Class

There can be multiple files for a single class. For example, both "Performance.h" and "Performance_simple.h" include class "Performance" and the simulation main can select an appropriate version of the class according to the simulation purpose.

If a simple version is preferred, one can use

#include "../../LinkPerformance/Performance/Performance_simple.h"

Else if a full version is preferred, one can use

#include "../../LinkPerformance/Performance/Performance.h"

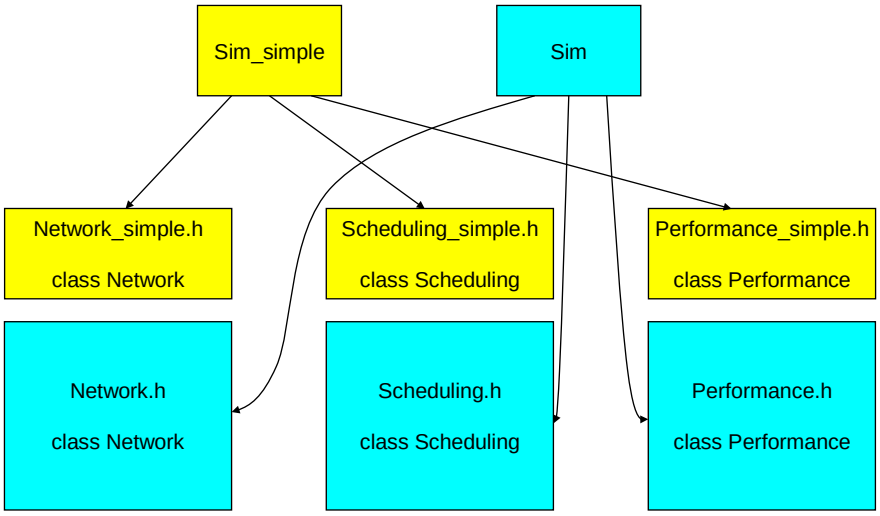

Figure 3-3 Implementation of multi-purpose multi-level simulator

"SystemSimConfiguration.h" includes the simulation type and "include" files.

<SystemSimConfiguration.h>

#define SYSTEM_SIM_CONFIGURATION_SIMPLE

#ifdef SYSTEM_SIM_CONFIGURATION_SIMPLE
#include "../../LinkPerformance/Performance/Performance_simple.h"
#else
#include "../../LinkPerformance/Performance/Performance.h"
#endif

3.1.3 Inheritance

If two versions of a class are similar or a new one is simply a superset of the old one, inheritance can be used. In this case, only the incremental part is programmed in the child.

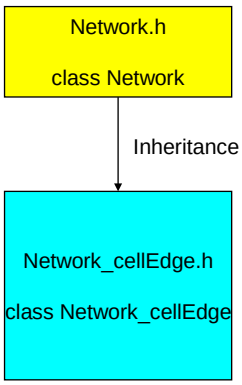

Figure 3-4 Example of inheritance

3.1.4 New Module Design

There will be multiple versions of the same module according to completeness, abstraction levels, used technologies, desired simulation time, and so on. If one wants to implement a new feature, he or she can select an appropriate version for the starting point. Note that, sometimes, a full version is not the best choice to modify since it may be too complicated and include many unnecessary features for the new simulator.

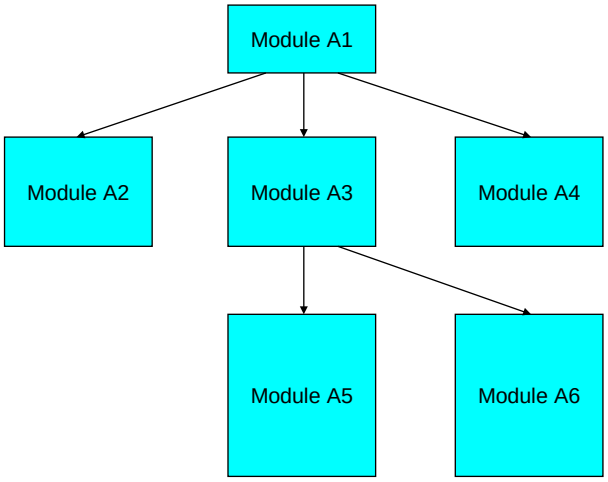

Figure 3-5 Example of module version hierarchy

3.2 Multiple Levels of Abstraction

The module selection mechanism using the simulation configuration file also allows for various forms of compromise between the simulation modes as well as the tradeoffs between the simulation complexity and accuracy.

A conceptual diagram of multiple abstraction levels is shown in Fig. 3. In the initial phase of algorithm development, a light-weight simulator can be constructed by combining simplified and fast-running modules for rapid prototyping. After coarse tuning of the algorithm is complete, a full-version simulator can be handled for accurate results. Various levels of abstraction allow diverse tradeoffs between the simulation time and the accuracy of results. Also, providing a simplified version of the simulator can help users to easily understand the software structure so that they can modify or extend the simulator.

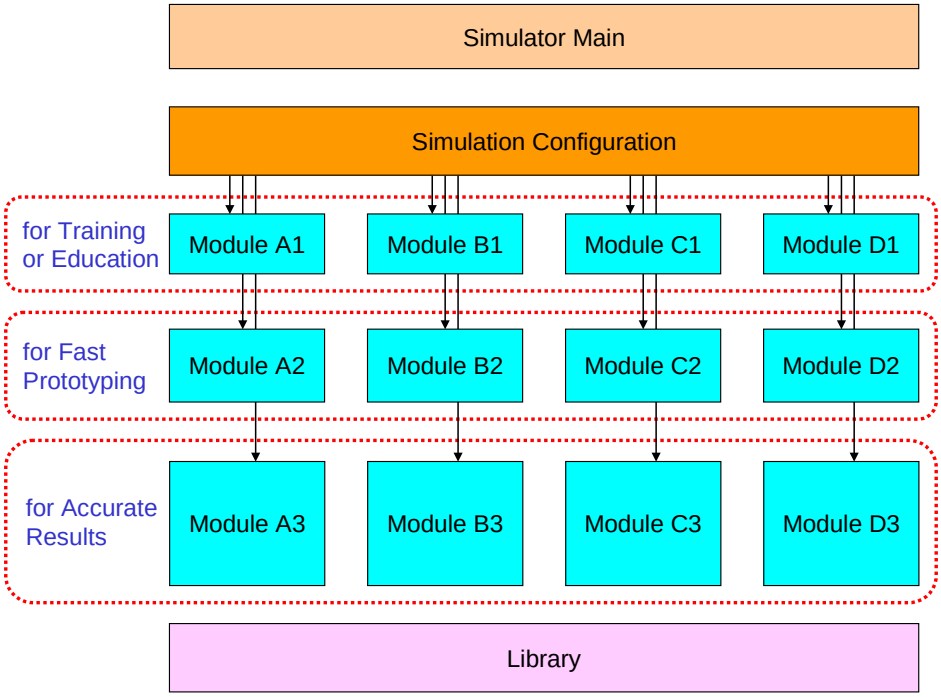

Figure 3-6 Multiple levels of abstraction

There are multiple modules which perform the same function. A simple version is used for education or training. A more detailed but not too complicated version is used for fast prototyping. A full version is also provided to produce accurate results.

Multi-purpose multi-level simulators can be implemented in two ways:

- Multiple files for a single class
- Inheritance

3.3 Integrated Simulation

Link level, system level, and network level simulators may need to be developed in consideration of integration since integration or tight interoperation of 5G simulators enables various types of simulations in accordance with user's requirements in a unified framework.

Although the simulator discussed in this document is mainly a system level simulator, it has a form of an integrated system and link level simulator. It performs link level and multi-cell link level simulations as well as system level simulation. Of course, when operating as a multi-cell link level simulator, it is not feasible to perform full functions of link level and system level simulations due to the simulation time constraints, and some reduced operations need to be performed.

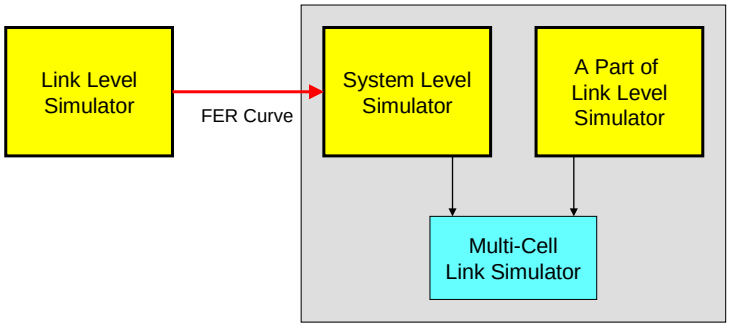

Figure 3-7 Integrated Simulation

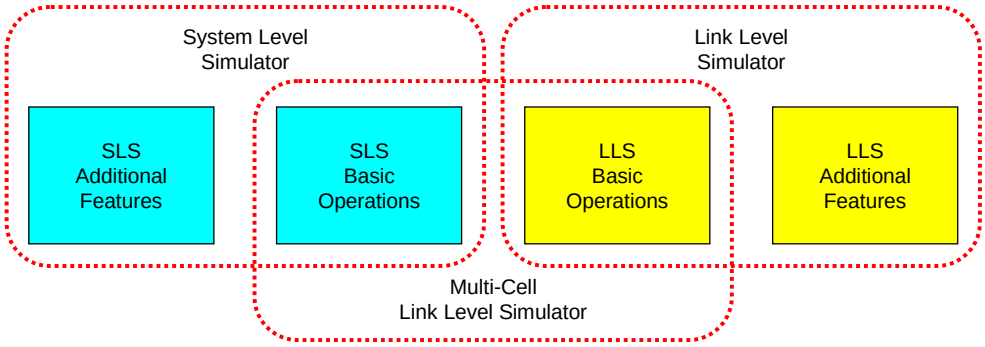

Figure 3-8 Integrated system and link level simulator

3.4 Miscellany Issues

3.4.1 Event Driven vs. Time Driven

Although event driven approaches are more general to handle various simulation cases, a time driven scheme is employed to reduce the simulation time.

3.4.2 Array vs. Linked List

Although linked list structures are more general to handle various simulation cases, an array structure is employed to use parallel loop.

3.5 Optimization Policy

3.5.1 Tradeoffs among Accuracy, Speed, and Readability

In order to reduce the simulation time, some calculation may be omitted in the simulation. However, this may affect the accuracy of the results.

For example, the followings may be considered:

- Limited number of interferers
- Limited number of channel clusters for interferers

4. Simulator Structure

4.1 Directory Structure

4.1.1 Overview

The simulator has the following directories.

- Top layer: Simulation top

  Simulation main, Global definitions and variables (including Network Element)

- Main layer: Main modules
  - Network Configuration

    Network, BS placement, MS placement, mobility, association, admission control
  - Radio Resource Management

    Scheduling, MS feedback, power control, rate control, hybrid ARQ
  - Link Performance

    Performance, Throughput, FER calculation, transmission and reception, effective SINR mapping
  - Channel Model

    Channel, antenna, large scale channel, small scale channel, shot term channel
- Utility layer: Library and Data

  Library, utility, input data, output data, logging
- Architecture-independent layer: GUI

  Graphical user interface

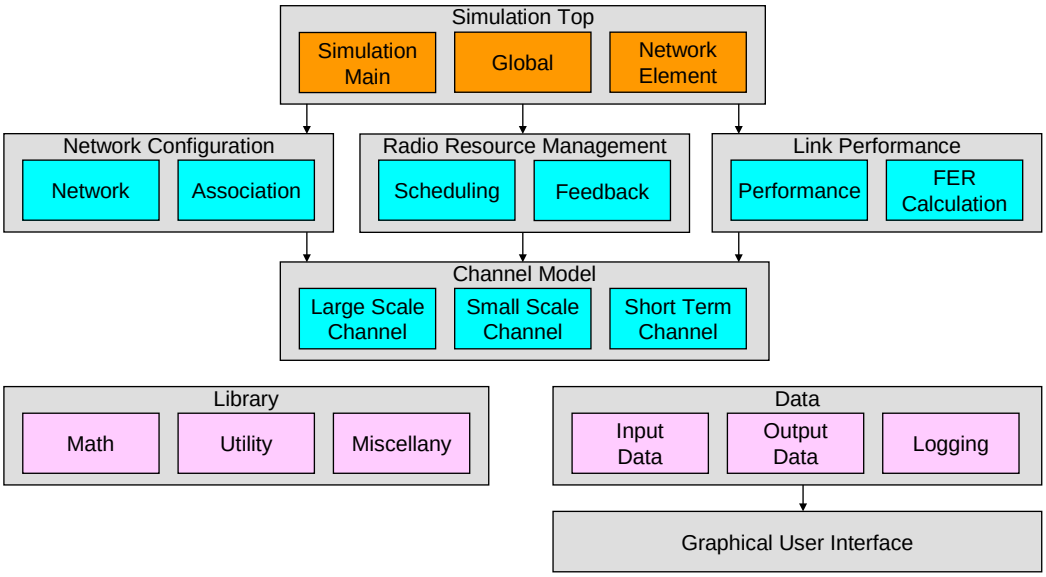

Figure 4-1 Example of directory structure

In addition, there is another directory named "LinkLevelSimulation", which will be explained later. "LinkLevelSimulation" is not used for system level simulation.

4.1.2 Directory and Files

The detailed direction structure is shown in the following table.

Table 4-1 Directory Structure

Directory	Sub-Directory	Sub-Sub-Directory	File Name
SimulationTop	SimulationMain		BasicMain
			IndoorHotspotMain
			DenseUrbanSingleLayerMain
			DenseUrbanTwoLayerMain
			RuralMain
			UrbanMacroMain
	Global		SystemSimConfiguration
			SLS
			SystemSim
	NetworkElement		SystemBS
			SystemMS
NetworkConfiguration	Network		Network
	NetworkBS		NetworkBS
	NetworkMS		NetworkMS
RadioResourceManagement	Scheduling		Scheduling
	SchedulingBS		SchedulingBS
	SchedulingMS		SchedulingMS

LinkPerformance	Performance		Performance
	PerformanceBS		PerformanceBS
	PerformanceMS		PerformanceMS
ChannelModel	Channel		Channel
	ChannelBS		ChannelBS
	ChannelMS		ChannelMS
	Antenna		Antenna
Library	Math		
	Utility		
	Debug		Debug
	Miscellany		
Data	InputData	FER	FER.txt
		SystemSimParameter	SimParamTop.txt
			SimParamNetwork.txt
			SimParamScheduling.txt
			SimParamPerformance.txt
			SimParamChannel.txt
		LinkSimParameter	SimParamLink.txt
	OutputData	NetworkConfiguration	PositionBS.txt
			PositionMS.txt
		SystemSimResult	Throughput.txt
		LinkSimResult	FER.txt
			UncodedBER.txt
			CodedBER.txt
	Logging		
	Cache		
GraphicalUserInterface			

There is another directory called "LinkLevelSimulation", which will be described later.

4.2 Global Variables

4.2.1 Overview

There are a few global variables used in the simulator since they are accessed by many modules. The number of global variables should be limited.

4.2.2 Global Variables

The global variables include

- System level simulation top, which includes simulation parameters and top level methods
- Base station arrays

- Mobile station arrays

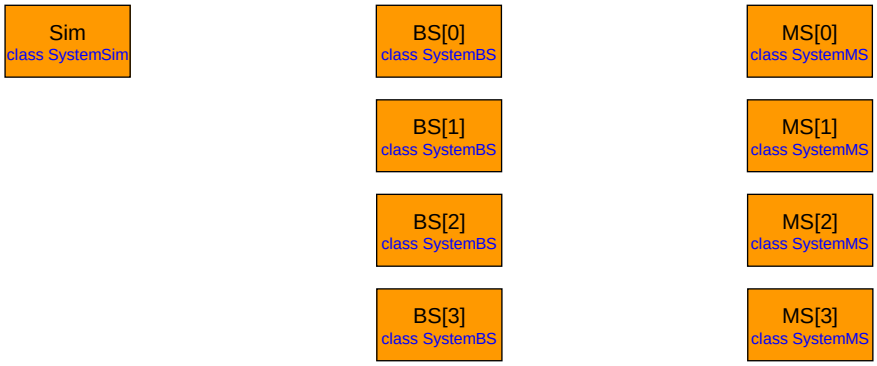

Figure 4-2 Global variable

Other network elements (such as "SystemRS", "SystemSite", or "SystemBBU") may be defined later.

4.3 Class Structure

4.3.1 File Name

A header file may include only one class which can be used globally. (Some header files may not include any class.) A file name is the same as the class name possibly with some extension. For example, "Performance_simple.h" has class "Performance".

4.3.2 Modular Design

The main part of the simulator (besides the simulation top) is decomposed into the following four main modules:

- Network Configuration

  Network, BS placement, MS placement, mobility, association, admission control

- Radio Resource Management

  Scheduling, MS feedback, power control, rate control, hybrid ARQ

- Link Performance

  Performance, FER calculation, transmission and reception, effective SINR mapping

- Channel Model

  Channel, antenna, large scale channel, small scale channel, shot term channel

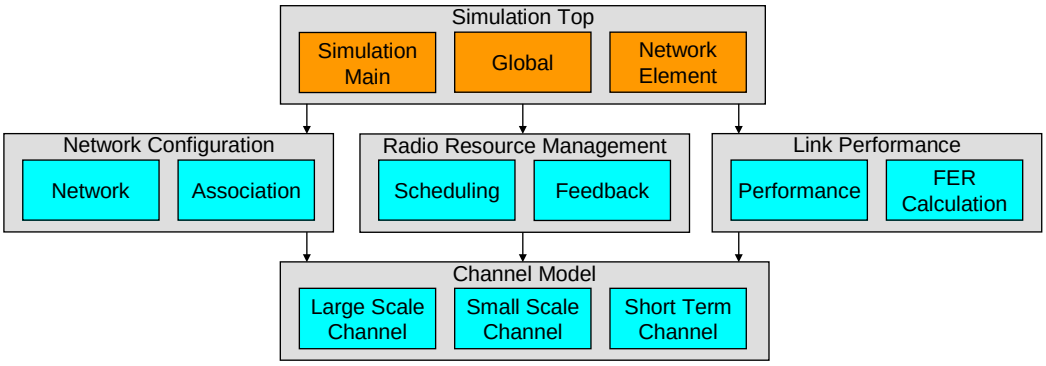

Figure 4-3 Main modules

For example, simulation parameters are placed in the four main modules.

```
class SystemSim
{
   ...
   Network network; // Parameters related to network configuration
   Scheduling scheduling; // Parameters related to radio resource management
   Performance performance; // Parameters related to link performance
   Channel channel; // Parameters related to channel
}
```

Of course, the initialization processes are performed where they are defined.

```
SystemSim::Initialize()
{
   ...
   Network.Initialize(file); // Parameter initialization for network configuration
   Scheduling.Initialize(file); // Parameter initialization for radio resource management
   Performance.Initialize(file); // Parameter initialization for link performance
   Channel.Intialize(file); // Parameter initialization for channel
}
```

Similarly, the variables and methods related to classes "SystemBS" and "SytemMS" are placed in the four modules.

```
class SystemBS
{
```

```
    NetworkBS network; // Network configuration info at BS
    SchedulingBS scheduling; // Scheduling info at BS
    PerformanceBS performance // Link performance info at BS
    ChannelBS channel; // Channel info at BS
}

class SystemMS
{
    NetworkMS network; // Network configuration info at MS
    SchedulingMS scheduling; // Scheduling info at MS
    PerformanceMS performance // Link performance info at MS
    ChannelMS channel; // Channel info at MS
}
```

Table 4-2 Class structure

Directory	Name	Simulation	Base Station	Mobile Station
Simulation Top	Class	SystemSim	SystemBS	SystemMS
	Variable	Sim	BS	MS
Network Configuration	Class	Network	NetworkBS	NetworkMS
	Variable	network	network	network
Radio Resource Management	Class	Scheduling	SchedulingBS	SchedulingMS
	Variable	scheduling	scheduling	scheduling
Link Performance	Class	Performance	PerformanceBS	PerformanceMS
	Variable	performance	performance	performance
Channel Model	Class	Channel	ChannelBS	ChannelMS
	Variable	channel	channel	channel

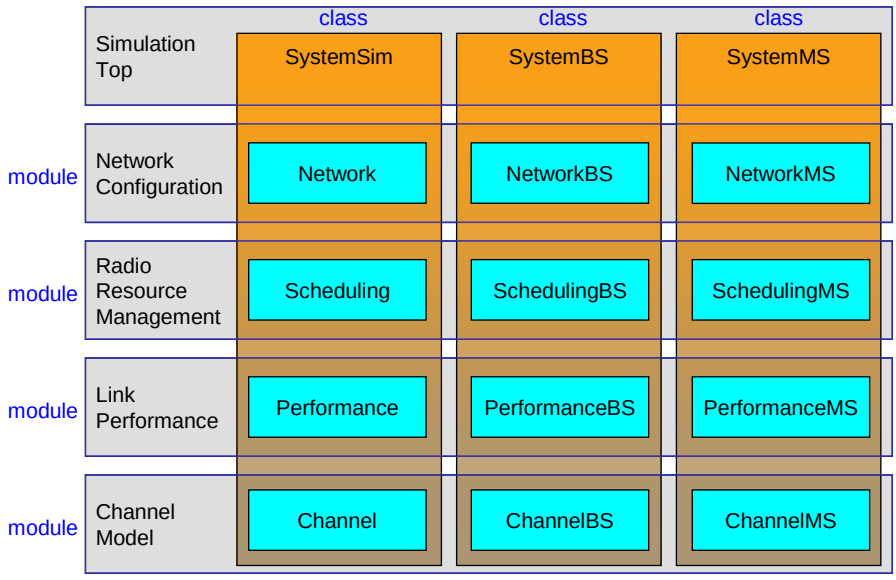

Figure 4-4 Class structure

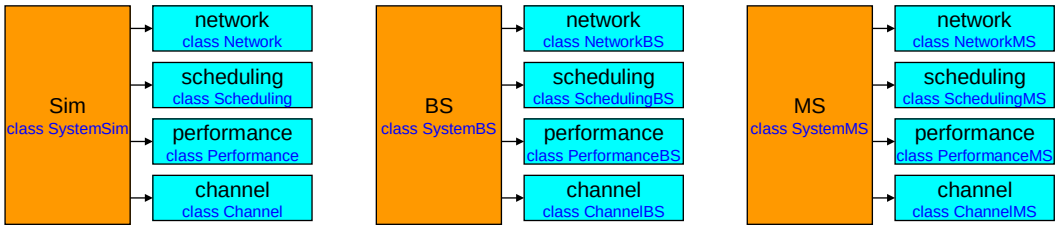

Figure 4-5 Global variables

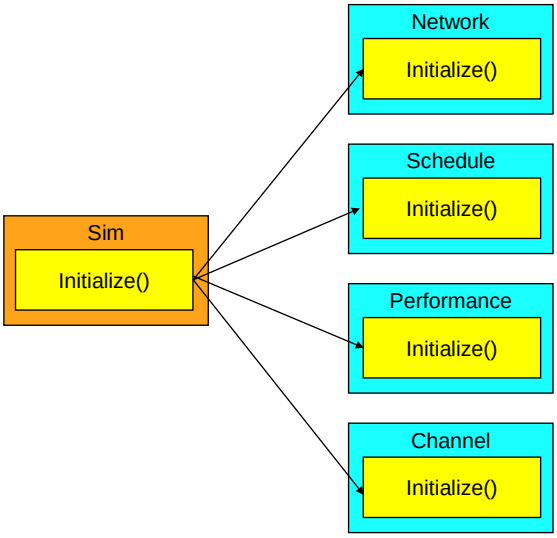

Figure 4-6 Method structure

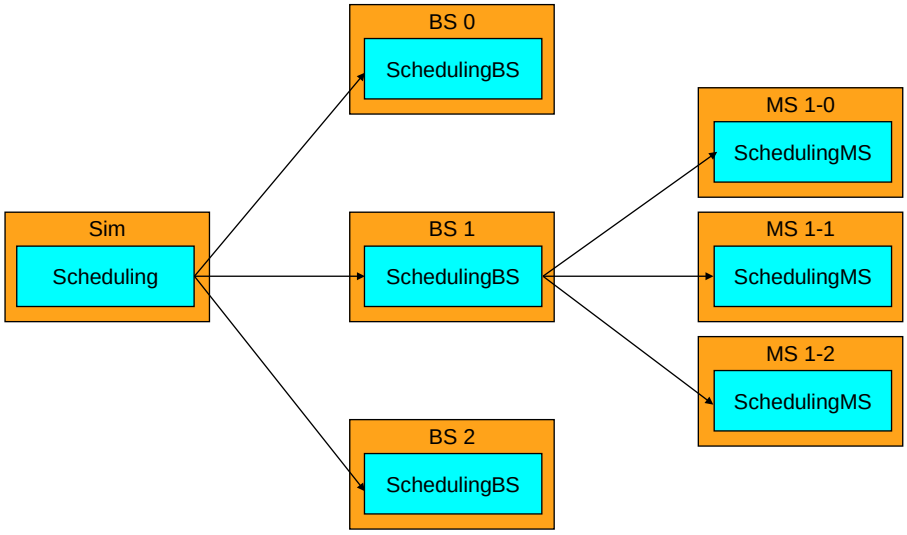

Figure 4-7 Object structure

If necessary, dynamic object construction can be used. Dynamic object construction will be avoided if possible.

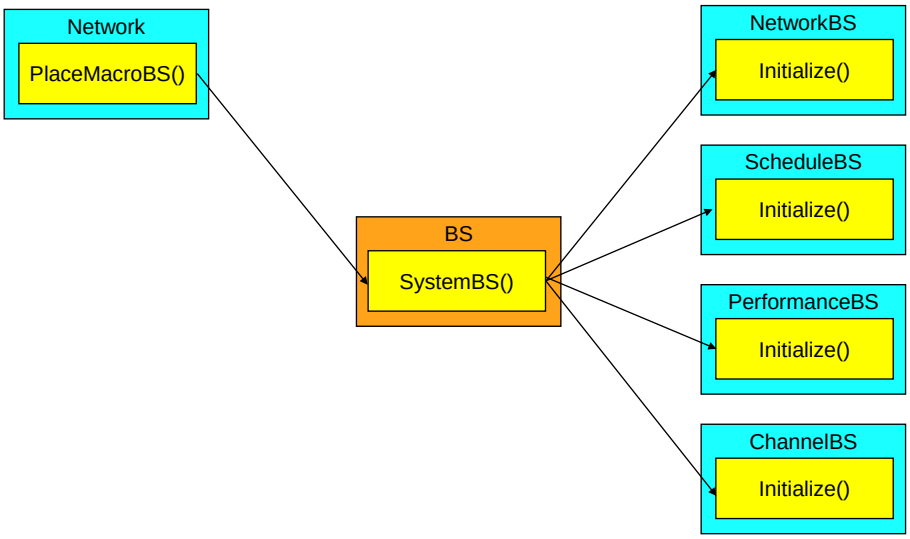

Figure 4-8 Dynamic object construction

4.3.3 Modular Design Diagram

The following diagram shows the structure of classes described in the previous section.

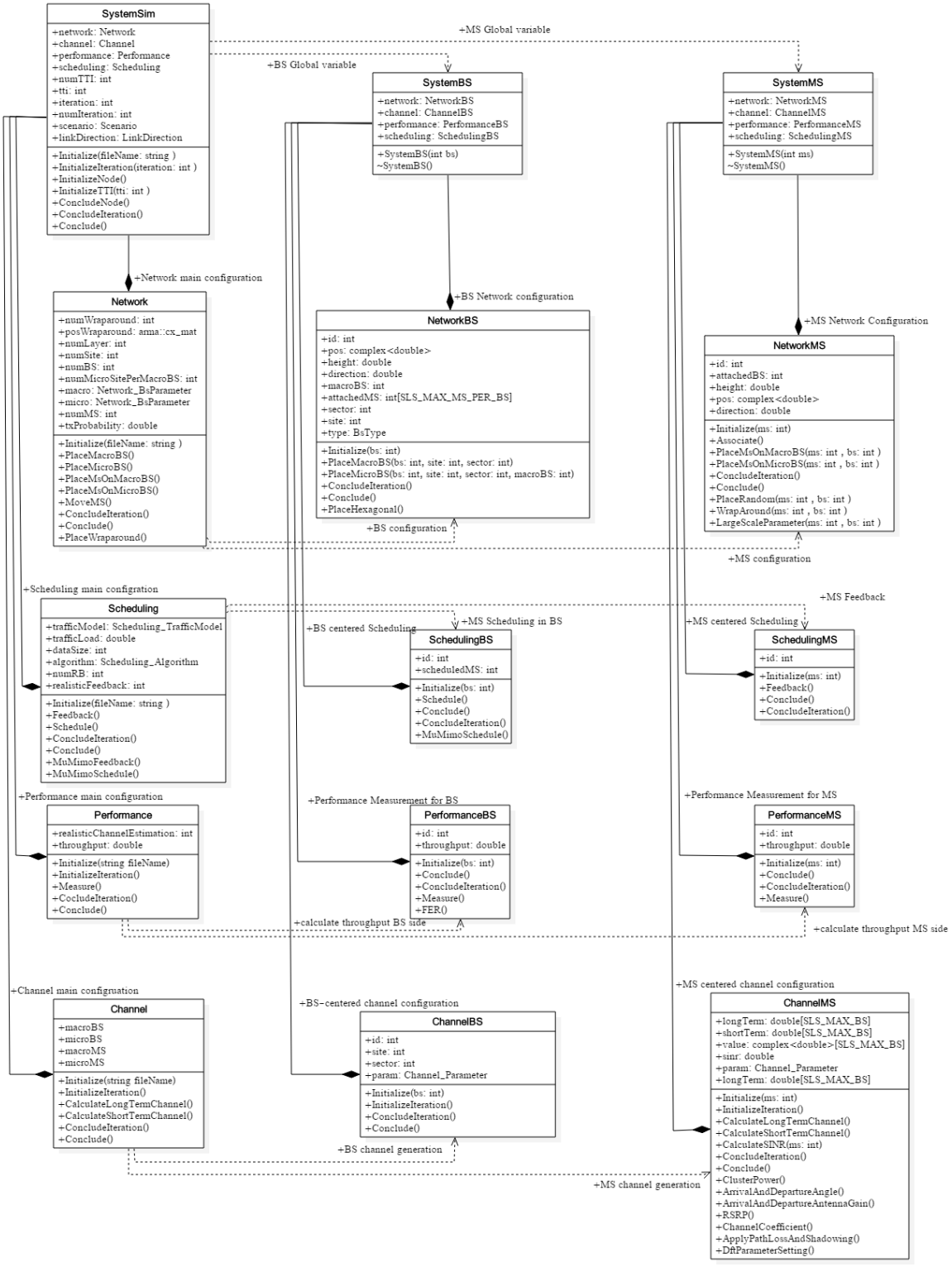

Figure 4-9 UML Diagram

The UML diagram in Figure 4-9 allows you to see the components of the module at a glance, but it is difficult to understand in relation to the simulation process.

In this case, the sequence diagram in Figure 4-10 shows how the module works with the actual simulation procedure.

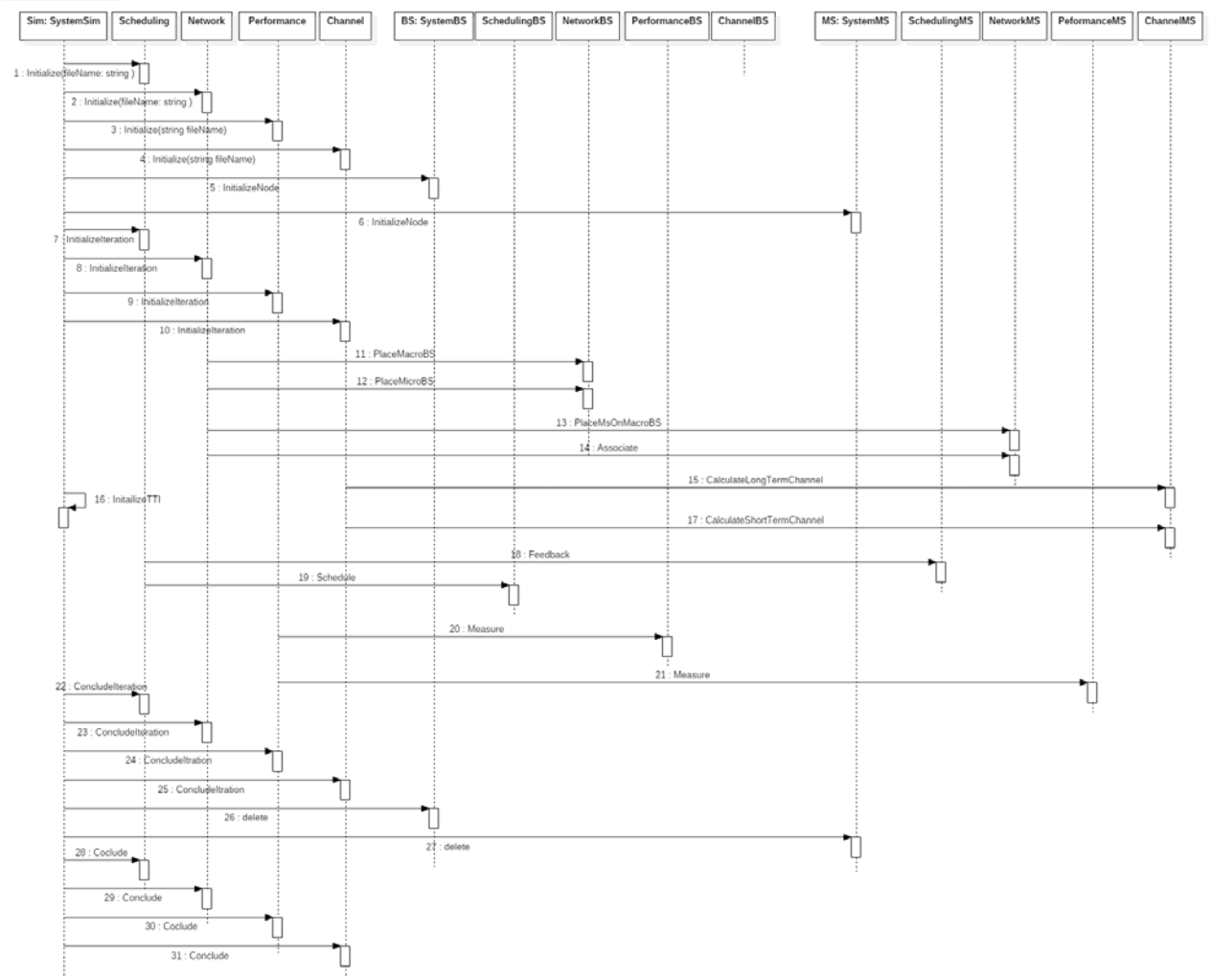

Figure 4-10 Sequence Diagram

4.4 Simulator Inputs

4.4.1 Simulation Parameter

The simulation parameters will be discussed later.

4.4.2 Data from Link Level Simulator

FER curves
EESM(Exponential Effect SINR Mapping)
MIESM(Mutual Information SINR Mapping)

4.5 Simulator Outputs

4.5.1 Performance Result

Average throughput
Median throughput
5% throughput

4.5.2 Network Configuration

BS positions
MS positions
Cell boundaries

4.5.3 Logging/Debugging

Logging data
Debugging data

4.6 Graphical User Interface

There can be several different types of GUIs and the details of GUIs may be discussed in other documentation. Some examples of 3D figures for system level simulation are shown in the following figures.

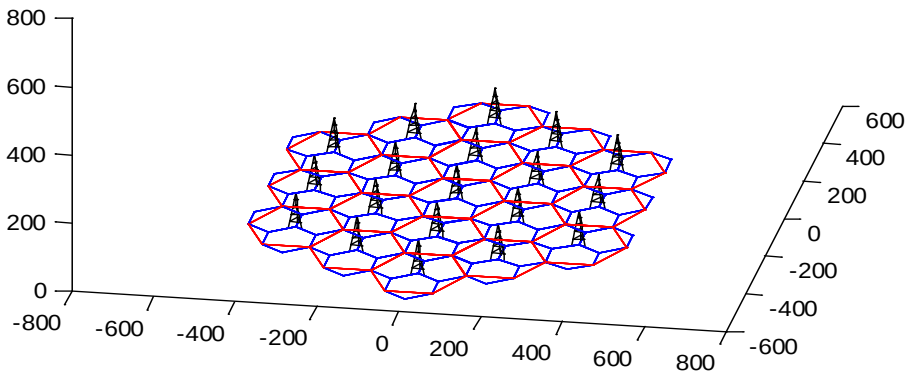

Figure 4-11 Base stations in 3D cell structure

4.7 Math Library

Simulator uses the armadillo math library. Armadillo is a high quality linear algebra library (matrix maths) for the C++ language, aiming towards a good balance between speed and ease of use. Additionally armadillo has high-level syntax (API) deliberately similar to Matlab.

Armadillo useful for algorithm development directly in C++, or quick conversion of research code into production environments (eg. software & hardware products) and provides efficient classes for vectors, matrices and cubes (1st, 2nd and 3rd order tensors), as well as 200+ associated functions; integer, floating point and complex numbers are supported.

Armadillo math library used in combination with the basic mathematical library of C ++.

For example, when using a complex number is declared as follows.

complex<double> angle; // is C++ standard complex library.

angle = MS[id]->network->pos - (Sim.network->posWraparound[j] + BS[i]->network->pos) * exp(-1i * 3.141592 * (double)(4 * (j % 3) - 3) / 6.0); // exp is function on armadillo library.

In this case, the exp function is the exponential function in armadillo.

Note: the complex number i is used as 1i in C ++, unlike MATLAB.

When using a array operation is declared as follows. Following source is for entering the value after declaring a complex matrix using cx_mat in armadillo.

```
arma::cx_mat hexagonal; // Declare a complex matrix.
hexagonal << 0 << A + 0.5i << 1i << -A + 0.5i << -A - 0.5i << -1i << A - 0.5i <<
        2 * A << 2 * A + 1i << A + 1.5 * 1i << 2.0 * 1i << -A + 1.5 * 1i << -2 * A + 1i <<
        -2 * A << -2 * A - 1i << -A - 1.5 * 1i << -2.0 * 1i << A - 1.5 * 1i << 2 * A - 1i;
// Part of entering a value in a matrix

hexagonal = Sim.network->macro.interSiteDistance * hexagonal; // Matrix operation.
```

The matrix can be multiplied or added by referring to this example.

Definitions of other mathematical functions or datatypes are documented in detail on the armadillo homepage (http://arma.sourceforge.net/docs.html). It is similar to MATLAB, so it can be handled flexibly.

5. Module Design

5.1 Overview

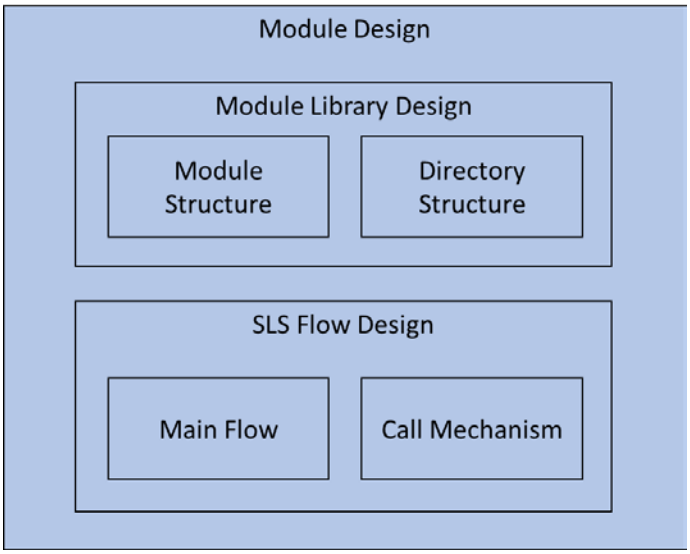

Figure 5-1 Module Design

Module design includes module library design and SLS flow design. Module library design defines module structure and directory structure, and SLS flow design defines main flow and corresponding call mechanism.

5.2 Module Architecture

5.2.1 Modular Concept

5.2.1.1 Modular Design

The system level simulator includes several simulator mains as well as many modules. The “main” part of each simulator selects and uses some combination of modules as required to meet the design purpose and scenarios. Our modular design concept is illustrated by Fig. 5-2, in which a “main” part of SLS specifies the required configuration of 5G modules for the given scenario.

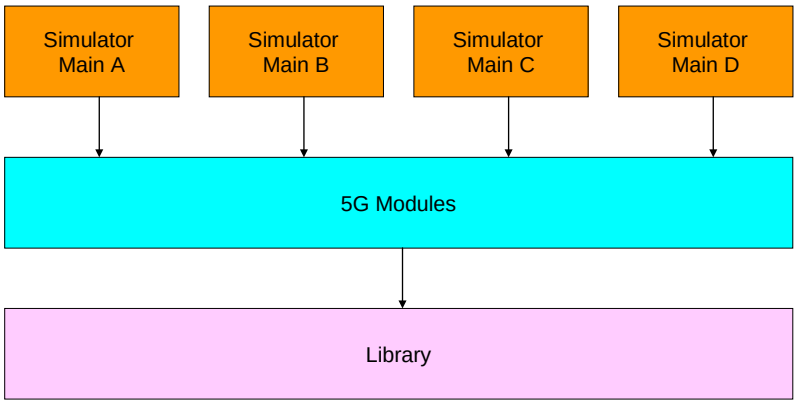

Figure 5-2. Modular Design

5.2.1.2 Multi-Purpose Simulator

There are multiple modules which perform the same function. A simple version is used for education or training. A more detailed but not too complicated version is used for fast prototyping. A full version is also provided to produce the accurate results. All the versions will be also a basis of the flexible prototyping for various system models and evaluation scenarios.

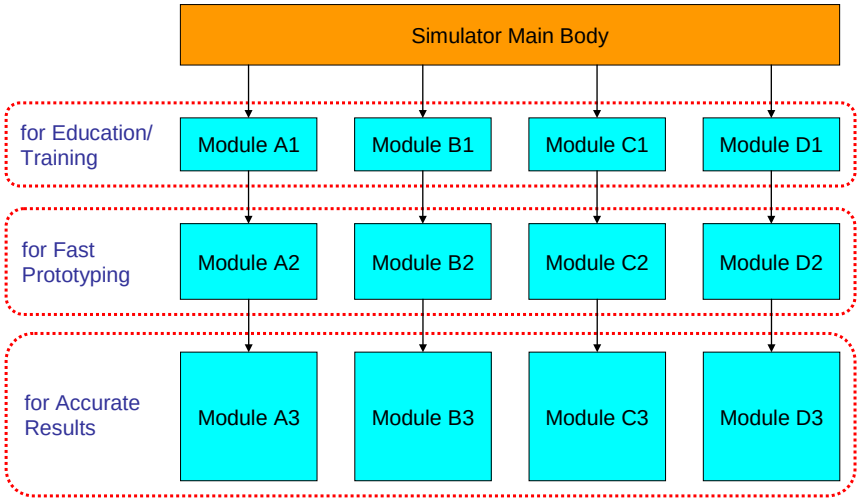

Figure 5-3. Multi-purpose multi-level simulator

Multi-purpose multi-level simulators can be implemented in two ways:

- Multiple files for a single class
- Inheritance

5.2.1.2.1 Multiple Files for a Single Class

There can be multiple files for a single class. For example, both "Performance.h" and "Performance_simple.h" include class "Performance" and the simulation main can select an appropriate version of the class according to the simulation purpose.

If a simple version is selected, one can use

#include "../../LinkPerformance/Performance/Performance_simple.h"

Else if a full version is selected, one can use

#include "../../LinkPerformance/Performance/Performance.h"

Fig. 5-4 illustrates how the simulator with the different purpose can be implemented with the different files within the same class. For example, it show three different classes, Network, Scheduling, and Performance, each of which has the different files for the different purpose.

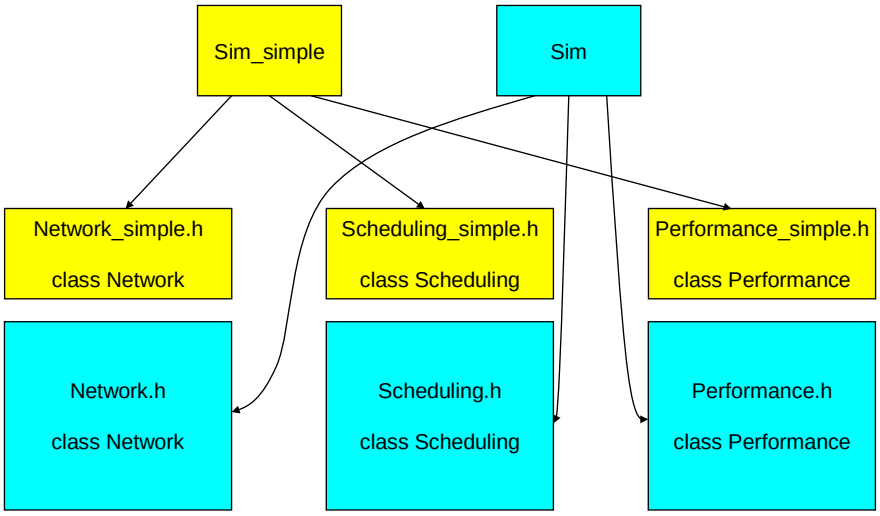

Figure 5-4. Implementation of multi-purpose multi-level simulator

"SystemSimConfiguration.h" includes the simulation type and "include" files.

<SystemSimConfiguration.h>

#define SYSTEM_SIM_CONFIGURATION_SIMPLE

#ifdef SYSTEM_SIM_CONFIGURATION_SIMPLE

#include "../../LinkPerformance/Performance/Performance_simple.h"

#else

#include "../../LinkPerformance/Performance/Performance.h"

#endif

5.2.1.2.2 Inheritance

If two versions of the class are similar or a new one is simply a superset of the old one, inheritance can be used. In this case, only the incremental part is programmed in the child. Fig. 5-5 illustrates a new class named "Network_cellEdge" is inherited from an old class named "Network" by incremental part of programming in the new class.

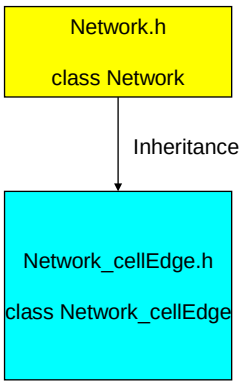

Figure 5-5. Inheritance

5.2.1.3 New Module Design

There will be multiple versions of the same module according to completeness, abstraction levels, used technologies, desired simulation time, and so on. If one wants to implement a new feature, one can select an appropriate version as a starting point. Note that a full version is sometimes not the best choice to modify since it may be too complicated and include many unnecessary features for the new simulator.

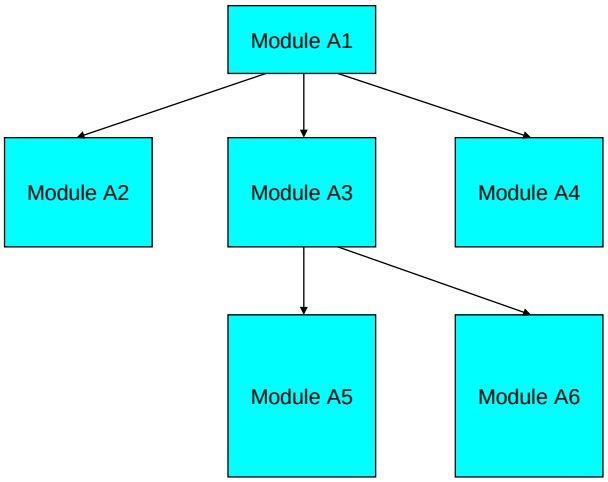

Figure 5-6 Example of module version hierarchy

5.2.2 Module Library

5.2.2.1 Module Structure

5.2.2.1.1 Global Modules

The global modules include

- System level simulation top, which includes simulation parameters and top level methods
- Base station arrays
- Mobile station arrays

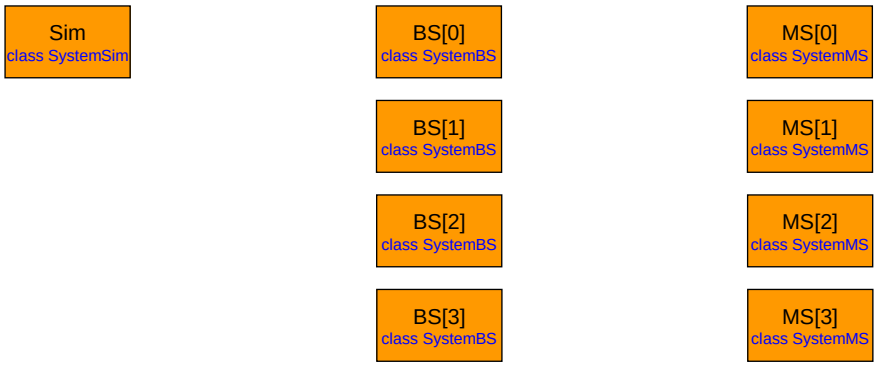

Figure 5-7 Global Modules

Other network elements (such as "SystemRS", "SystemSite", or "SystemBBU") may be defined later.

5.2.2.1.2 Modular Design

The main part of the simulator (besides the simulation top) is decomposed into the following four main modules:

- Network Configuration

  Network, BS placement, MS placement, mobility, association, admission control
- Radio Resource Management

  Scheduling, MS feedback, power control, rate control, hybrid ARQ
- Link Performance

  Performance, FER calculation, transmission and reception, effective SINR mapping
- Channel Model

  Channel, antenna, large scale channel, small scale channel, shot term channel

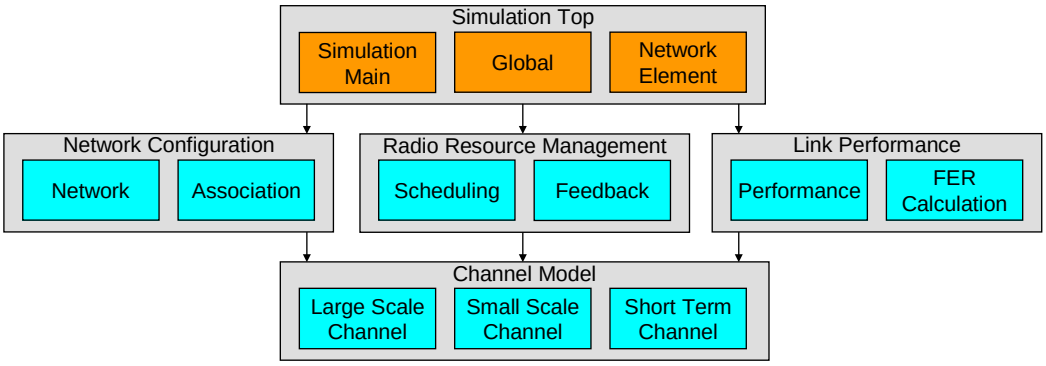

Figure 5-8 Main modules

For example, simulation parameters are placed in the four main modules.

```
class SystemSim
{
   ...
   Network network; // Parameters related to network configuration
   Scheduling scheduling; // Parameters related to radio resource management
   Performance performance; // Parameters related to link performance
   Channel channel; // Parameters related to channel
}
```

Of course, the initialization processes are performed where they are defined.

```
SystemSim::Initialize()
{
   ...
   Network.Initialize(file); // Parameter initialization for network configuration
   Scheduling.Initialize(file); // Parameter initialization for radio resource management
   Performance.Initialize(file); // Parameter initialization for link performance
   Channel.Intialize(file); // Parameter initialization for channel
}
```

Similarly, the variables and methods related to classes "SystemBS" and "SytemMS" are placed in the four modules.

```
class SystemBS
{
```

```
    NetworkBS network; // Network configuration info at BS
    SchedulingBS scheduling; // Scheduling info at BS
    PerformanceBS performance // Link performance info at BS
    ChannelBS channel; // Channel info at BS
}

class SystemMS
{
    NetworkMS network; // Network configuration info at MS
    SchedulingMS scheduling; // Scheduling info at MS
    PerformanceMS performance // Link performance info at MS
    ChannelMS channel; // Channel info at MS
}
```

Table 5-1. Class structure

Directory	Name	Simulation	Base Station	Mobile Station
Simulation Top	Class	SystemSim	SystemBS	SystemMS
	Variable	Sim	BS	MS
Network Configuration	Class	Network	NetworkBS	NetworkMS
	Variable	network	network	network
Radio Resource Management	Class	Scheduling	SchedulingBS	SchedulingMS
	Variable	scheduling	scheduling	scheduling
Link Performance	Class	Performance	PerformanceBS	PerformanceMS
	Variable	performance	performance	performance
Channel Model	Class	Channel	ChannelBS	ChannelMS
	Variable	channel	channel	channel

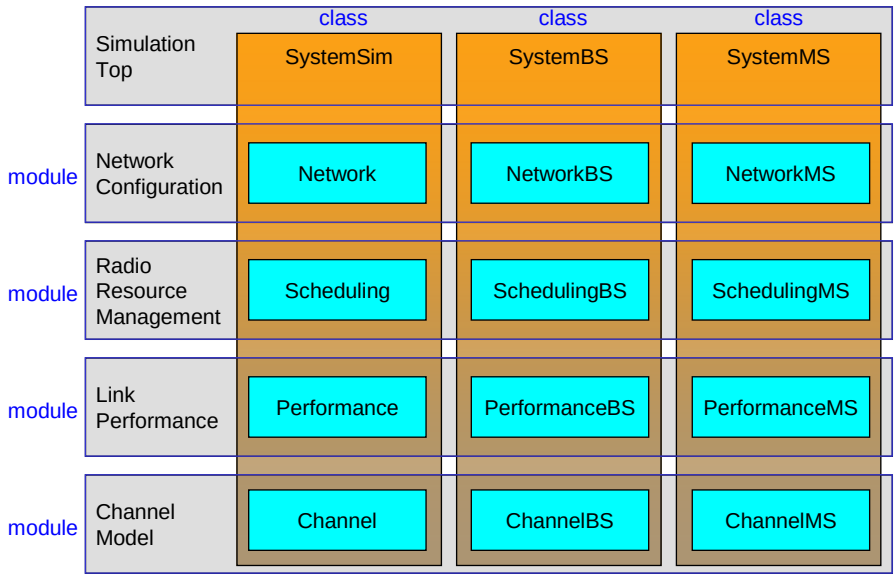

Figure 5-9. Class structure

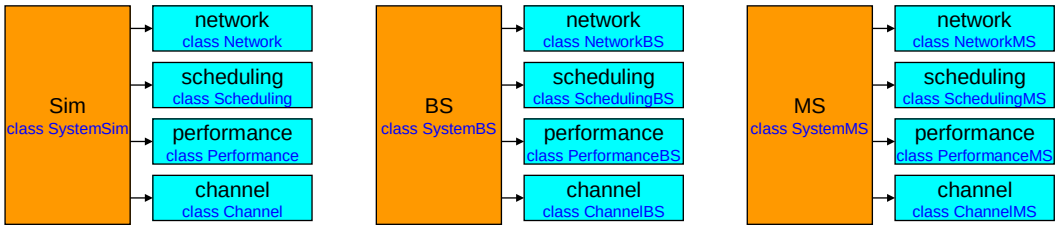

Figure 5-10. Global variables

5.2.2.2 Directory Structure

5.2.2.2.1 Overview

The simulator has the following directories:

- Top layer: Simulation top

  Simulation main, Global definitions and variables (including Network Element)

- Main layer: Main modules
  - ✧ Network Configuration:

    Network, BS placement, MS placement, mobility, association, admission control
  - ✧ Radio Resource Management:

    Scheduling, MS feedback, power control, rate control, hybrid ARQ
  - ✧ Link Performance:

    Performance, Throughput, FER calculation, transmission and reception, effective SINR mapping

✧ Channel Model:

Channel, antenna, large scale channel, small scale channel, shot term channel

- Utility layer: Library and Data

Library, utility, input data, output data, logging

- Architecture-independent layer: GUI

Graphical user interface

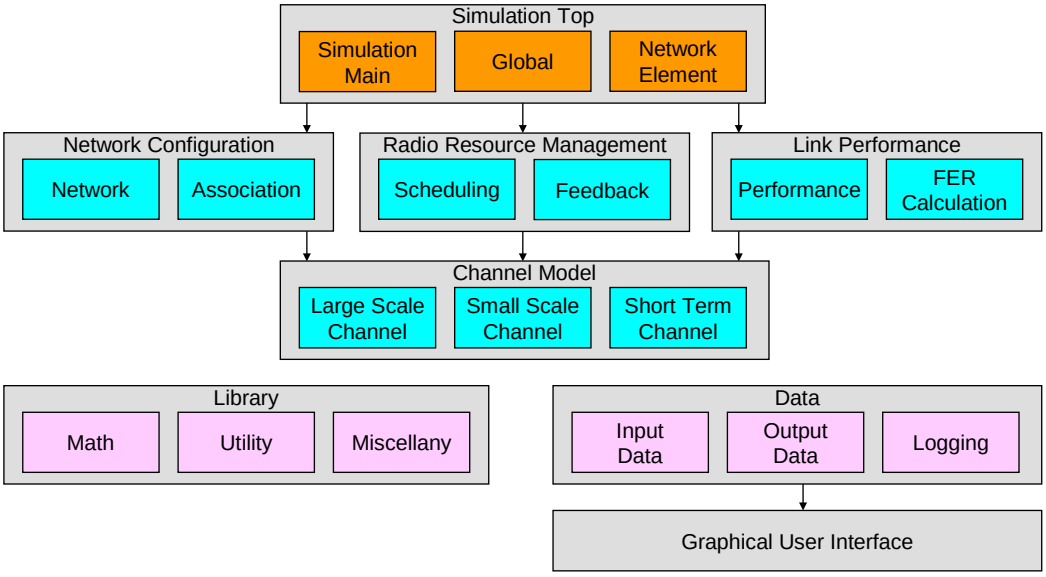

Figure 5-11. Directory structure

In addition, there is another directory named "LinkLevelSimulation", which will be explained later. "LinkLevelSimulation" is not directly used for system level simulation.

5.2.2.2.2 Directory and Files

The detailed direction structure is shown in the following table.

Table 5-2. Directory Structure

Directory	Sub-Directory	Sub-Sub-Directory	File Name
SimulationTop	SimulationMain		BasicMain
			IndoorHotspotMain
			DenseUrbanSingleLayerMain
			DenseUrbanTwoLayerMain
			RuralMain

			UrbanMacroMain
	Global		SystemSimConfiguration
			SLS
			SystemSim
	NetworkElement		SystemBS
			SystemMS
NetworkConfiguration	Network		Network
	NetworkBS		NetworkBS
	NetworkMS		NetworkMS
	CellAssignment		Association
			AdmissionControl
	MobilityManagement		MobilityManagement
RadioResourceManagement	Scheduling		Scheduling
	SchedulingBS		SchedulingBS
	SchedulingMS		SchedulingMS
	Feedback		Feedback
	PowerControl		PowerControl
	RateControl		AdaptiveModulationCoding
			OuterLoopRateControl
	Retransmission		HybridARQ
LinkPerformance	Performance		Performance
	PerformanceBS		PerformanceBS
	PerformanceMS		PerformanceMS
	CalculatingFER		CalculatingFER
	EffectiveSinrMapping		ExponentialESM
			MutualInformationESM
ChannelModel	Channel		Channel
	ChannelBS		ChannelBS
	ChannelMS		ChannelMS
	LargeScaleChannel		LargeScaleChannel
			LargeScaleCorrelation
	SmallScaleChannel		
	ShortTermChannel		
	Antenna		Antenna
Library	Math		
	Utility		
	Debug		Debug
	Miscellany		
Data	InputData	FER	FER.txt
		SystemSimParameter	SimParamTop.txt
			SimParamNetwork.txt
			SimParamScheduling.txt
			SimParamPerformance.txt
			SimParamChannel.txt
		LinkSimParameter	SimParamLink.txt
	OutputData	NetworkConfiguration	PositionBS.txt
			PositionMS.txt
		SystemSimResult	Throughput.txt
		LinkSimResult	FER.txt
			UncodedBER.txt
			CodedBER.txt
	Logging		
	Cache		
GraphicalUserInterface			

There is another directory called "LinkLevelSimulation", which will be described later.

5.3 Function Architecture

Multiple functions can be defined in one module. In this chapter, functions defined for each module are organized.

5.3.1 SystemSim

Table 5-3. Sim functions

Function type	Function name	Remarks	Representative output
void	Initialize(string fileName)	Initialization	System level simulation scenario
void	Conclude()	Concluding the simulation	memory release

5.3.1.1 Network Configuration

Table 5-4. Sim.network functions

Function type	Function name	Remarks	Representative output
void	Initialize(string fileName)	Initialization	Sim.network
void	PlaceBS()	Call a BS generation functions	-
void	PlaceMS()	Call a MS generation functions	-
void	Conclude()	Network conclusion	-

5.3.1.2 Channel Model

Table 5-5. Sim.channel functions

Function type	Function name	Remarks	Representative output
void	Initialize(string fileName)	Initialization	Sim.channel
void	TemporaryGlobalVariableInitialize	-	-
void	LongTermChannel	Call a LongTermChannel fuctions from ChannelMS	-
void	ShortTermChannel	Call a ShortTermChannel fuctions from ChannelMS	-

void	Conclude()	Channel conclusion	-

Function type	Function name	Remarks	Representative output
void	Initialize(string fileName)	Initialization	Sim.channel
void	TemporaryGlobalVariableInitialize	-	-
void	LongTermChannel	Call a LongTermChannel fuctions from ChannelMS	-
void	ShortTermChannel	Call a ShortTermChannel fuctions from ChannelMS	-
void	Conclude()	Channel conclusion	-

5.3.1.3 Radio Resource Management

Table 5-6. Sim.schedule functions

Function type	Function name	Remarks	Representative output
void	Initialize(string fileName)	Initialization	Sim.scheduling
double	GetSpectralEfficiency (double SINR, int & MCS)	This function returns spectral efficiency and MCS for the given SINR	MS[msID].scheduling=>spectralEfficiency MS[msID].scheduling=>MCS
void	Feedback()	Call a feedback fuctions	-
void	Schedule()	Call a schedule fuctions	-
void	Conclude()	Radio resource management conclusion	-

5.3.1.4 Link Performance

Table 5-7 Sim.performance functions

Function type	Function name	Remarks	Representative output
void	Initialize(string fileName)	Initialization	Sim.performance
double	FER(double SINR, int MCS)	This function returns frame error rate for the given SINR and MCS	FERvalue
void	Measure()	Call a throughput measurement functions	-
void	Conclude()	Link performance conclusion	-

5.3.2 SystemBS

Table 5-8 BS functions

Function type	Function name	Remarks	Representative output
-	SystemBS(int bs, int site, int sector, SLS::BsType type)	Declare constructor	BS[bsID]

5.3.2.1 Network Configuration

Table 5-9. BS.network functions

Function type	Function name	Remarks	Representative output
void	Initialize(int bs, int site, int sector, SLS::BsType type)	Initialization	BS[bsID].network
void	PlaceHexagonal()	Place BS in hexagonal form	BS[bsID].network=>pos3D
void	Conclude()	NetworkBS conclusion	-

5.3.2.2 Channel Model

Table 5-10. BS.channel functions

Function type	Function name	Remarks	Representative output
void	Initialize(int bs, int site, int sector, SLS::BsType type)	Initialization	BS[bsID].channel
void	Conclude()	ChannelBS conclusion	-

5.3.2.3 Radio Resource Management

Table 5-11. BS.schedule functions

Function type	Function name	Remarks	Representative output
void	Initialize(int bs)	Initialization	BS[bsID].scheduling
void	Schedule()	Performs scheduling for the associated MS	BS[bsID].scheduling=>scheduledMS BS[bsID].scheduling=>precodingMatrix
void	Conclude()	SchedulingBS conclusion	-

5.3.2.4 Link Performance

Table 5-12. BS.performance functions

Function type	Function name	Remarks	Representative output
void	Initialize(int bs)	Initialization	BS[bsID].performance
void	Conclude()	PerformanceBS conclusion	-

5.3.3 SystemMS

Table 5-13. MS functions

Function type	Function name	Remarks	Representative output
-	SystemMS(int ms, int bs)	Declare constructor	MS[msID]

5.3.3.1 Network Configuration

Table 5-14. MS.network functions

Function type	Function name	Remarks	Representative output
void	Initialize(int ms, int bs)	Initialization	MS[msID].network
void	Associate()	Associate MS to BS	MS[msID].channel=>associatedBsIndex MS[msID].channel=>StrongInterferenceBSindex MS[msID].channel=>WeakInterferenceBSindex
void	PlaceRandom()	Place in a random pattern	MS[msID].network=>pos3D
void	Conclude()	NetworkMS conclusion	-

5.3.3.2 Channel Model

Table 5-15. MS.channel functions

Function type	Function name	Remarks	Representative output
void	Initialize(int ms)	Initialization	-
void	LongTermChannel	Long-term channel generation	MS[msID].channel=>LongTermTimeInterferenceChannel
void	DftParameterSetting	DFT parameter generation	MS[msID].channel=>DftParameter
void	ShortTermChannel	Short-term channel generation	MS[msID].channel=>ShortTermFrequencyChannel
void	GeneralParameters	General Parameters generation	MS[msID].channel=>PathLoss MS[msID].channel=>LargeScaleParameter

void	SmallScaleParameter	Small scale parameter generation	MS[msID].channel=>Delay MS[msID].channel=>ClusterPower MS[msID].channel=>ArrivalAndDepartureAngle MS[msID].channel=>ArrivalAndDepartureAntennaGain
void	CoefficientGeneration	Coefficient generation	MS[msID].channel=>RSRP
void	RSRP	RSRP calculation	MS[msID].channel=>RSRP
void	ChannelCoefficient	Channel coefficient generation	MS[msID].channel=>LongTermTimeInterferenceChannel
void	ApplyPathLossAndShadowing	Shadowing adding	MS[msID].channel=>LongTermTimeInterferenceChannel
void	Pathloss	Pathloss calculation	MS[msID].channel=>PathLoss
void	Delay	Delay generation	MS[msID].channel=>Delay
void	ClusterPower	Cluster power generation	MS[msID].channel=>ClusterPower
void	ArrivalAndDepartureAngle	AoD calculation	MS[msID].channel=>ArrivalAndDepartureAngle
void	ArrivalAndDepartureAntennaGain	AoD antenna gain calculation	MS[msID].channel=>ArrivalAndDepartureAntennaGain
void	DiscreteFourierTransform	DFT performing	MS[msID].channel=>DftParameter
double	AntennaGain	Antenna gain calculation	MS[msID].channel=>ArrivalAndDepartureAngle
arma::mat	ReceiverAntennaGain	Receiver antenna gain	MS[msID].channel=>ReceiverAntennaGainLOSXH MS[msID].channel=>ReceiverAntennaGainLOSXV
arma::cx_mat	TransmitterAntennaGain	Transmitter antenna gain	MS[msID].channel=>TransmitterAntennaGainLOSXH MS[msID].channel=>TransmitterAntennaGainLOSXV
arma::cx_mat	exp_F_urd	Receiver antenna gain matrix	arma::mat F_urd_LOS
arma::cx_mat	exp_F_prd	Transmitter antenna gain matrix	arma::mat F_prd_LOS
double	Distance2D	2D distance	MS[msID].channel=>distance2D
double	Distance3D	3D distance	MS[msID].channel=>distance3D
double	AzimuthAngleOfGlobalCoordinateSystem	global coordinate	MS[msID].channel=>GCSAOA MS[msID].channel=>GCSAOD
double	ZenithAngleOfGlobalC	global coordinate	MS[msID].channel=>GCSZOA

	oordinateSystem		MS[msID].channel=>GCSZOD
arma::mat	GlobalCoordinateSystemAngleToLocalCoordinateSystemAngle	global to local coordinate	arma::mat departureAngleLCS arma::mat arrivalAngleLCS
arma::mat	LocalCoordinateSystemAntennaGainToGlobalCoordinateSystemAntennaGain	local to global local coordinate	arma::mat antennaGainLCS arma::mat antennaGainLCS2
arma::mat	DistanceAngular	-	angularDistance
void	Conclude()	ChannelMS conclusion	-

5.3.3.3 Radio Resource Management

Table 5-16. MS.schedule functions

Function type	Function name	Remarks	Representative output
void	Initialize(int ms)	Initialization	MS[msID].scheduling
void	Feedback()	Channel state information feedback	MS[msID].scheduling=>CQI MS[msID].scheduling=>spectralEfficiency MS[msID].scheduling=>PrecodingMatrix
void	Conclude()	SchedulingMS conclusion	-

5.3.3.4 Link Performance

Table 5-17. MS.performance functions

Function type	Function name	Remarks	Representative output
void	Initialize(int ms)	Initialization	MS[msID].preformance
void	Measure()	Throughput measurement	MS[msID].performance=>throughput MS[msID].scheduling=>averagedThroghput
void	Conclude()	PerformanceMS conclusion	-

6. References

[1] System Level Simulator – Requirement Specification

[2] System Level Simulator - Skeleton Design

[3] 3GPP TR 36.873 V12.0.0, “Study on 3D channel model for LTE,” Rel. 12, Sep. 2014.

[4] 3GPP TR 38.900 V14.0.0, “Study on channel model for frequency spectrum above 6 GHz,” Rel. 14, July 2017.

[5] 3GPP TR 38.802 V14.2.0, “Study on New Radio (NR) Access Technology; Physical Layer Aspects,” Rel. 14, Sep. 2017.