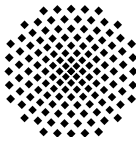

Universität Stuttgart

INSTITUT FÜR
KOMMUNIKATIONSNETZE
UND RECHNERSYSTEME
Prof. Dr.-Ing. Andreas Kirstädter

IKR Simulation Library 4.0

Simulation Example

November 16, 2017

Editors

Marc Barisch
Christoph Gauger
Marc Necker
Jörg Sommer
Joachim Scharf
Sebastian Scholz

1 Introduction

This document presents a simple example which demonstrates the execution of a complete performance evaluation study, explains all steps from setting up the simulation to designing the graphs.

Besides a classical queuing model, we also show an example of a simple ACK/NACK protocol implementation in Section 6.

2 Queuing Model

2.1 Model description

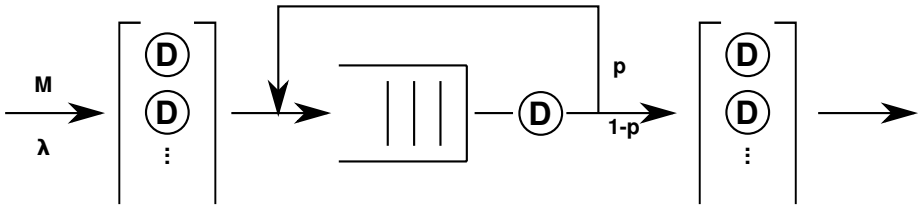

Figure 2.1: Model

Fig. 2.1 shows the general model. Packets arrive at an infinite server with negative exponentially distributed interarrival times (mean rate λ) and a constant packet length L. The infinite server introduces a constant delay to all packets. The packets are then sent into an unbounded queue. Following the queue the packets are processed by a phase where the service time depends linearly on the packet size. After processing packets are fed back into the queue with a probability of p. With a probability of 1-p they are send towards a second infinite server. After processing in the infinite server the packets leave the system.

2.2 Simulation Model

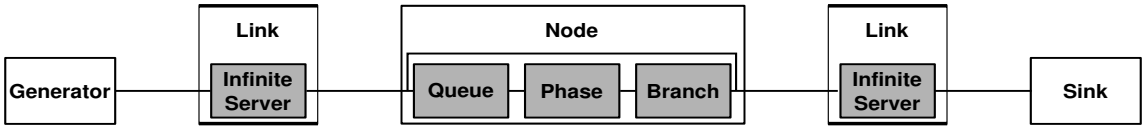

Figure 2.2: Simulation Model

The general model of Fig. 2.1 is realized by the components given in Fig. 2.2. A Generator produces negative exponentially distributed messages of constant length L. These messages are sent to the link. After the first link follows a node wich processes the packets and sends them into the second link. Finally, the messages are absorbed by a sink.

2.3 Steps from Simulation towards results

1. First of all the model has to be created. The source code of the simulation model used here can be found in the provided source archives.
   Hint: In the infrastructure of the IKR the example can be found under /net/arch/ikr/simlib-java/.
2. Import the example in Eclipse and export a runnable Jar file (c.f. Section 3.1)
3. Create the simulation tree (c.f. Section 3.2)
4. Run the simulation (c.f. Section 3.3)
5. Extract results from log files (c.f. Section 4.2)
6. Create graphs (c.f. Section 4.3)

3 Simulation

3.1 Importing the example in Eclipse

We suggest to use an advanced IDE to develop your simulation model. Our standard approach ist to use Eclipse, but any other IDE is suited as well. In this tutorial we will use the provided source code, import it into a new Eclipse project, execute it locally and finally export it as runnable Jar file.

3.1.1 Create a new Eclipse Project

Open the context menu by right clicking in the package explorer. In the context menu select a new java project.

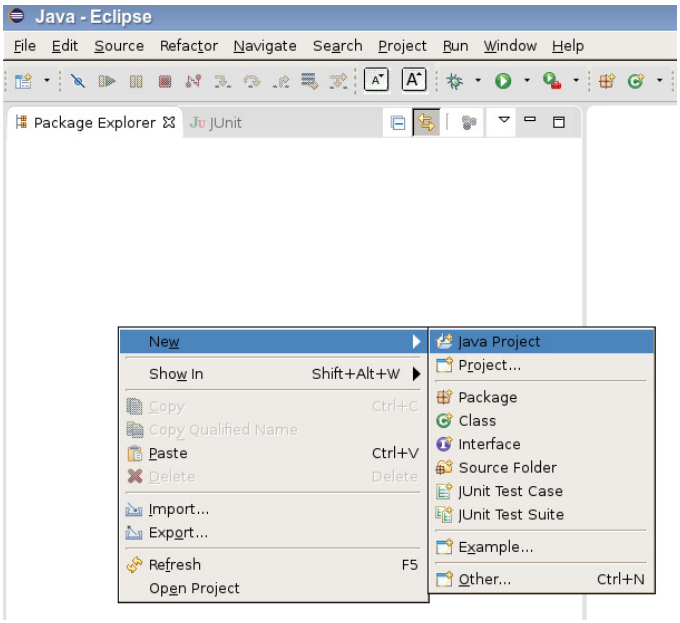

Figure 3.1: Create new Eclipse project

In the new dialog enter a name for your new project. The remaining settings can be left to their defaults.

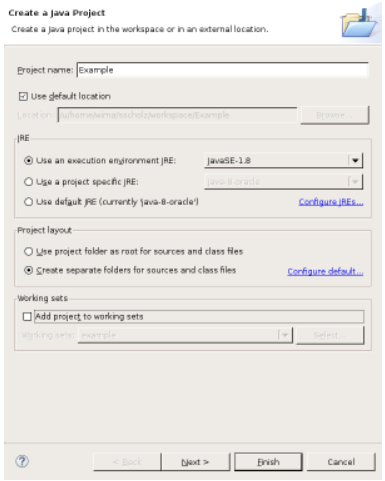

Figure 3.2: Create new Eclipse project

3.1.2 Import the code of the example project

To import the code of the existing example, select the src folder of the newly created project, click right and select import.

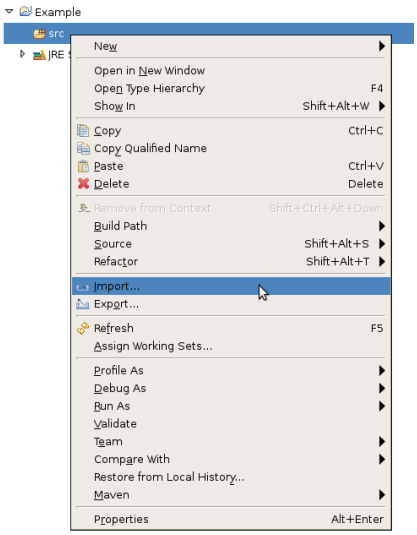

Figure 3.3: Import the example code

In the next window select archive file and press next.

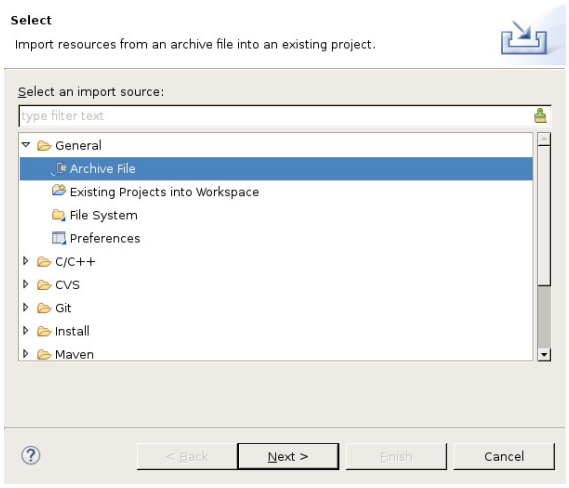

Figure 3.4: Import the example code

Then select the jar file containing the source files. The remaining settings can be left to their defaults.

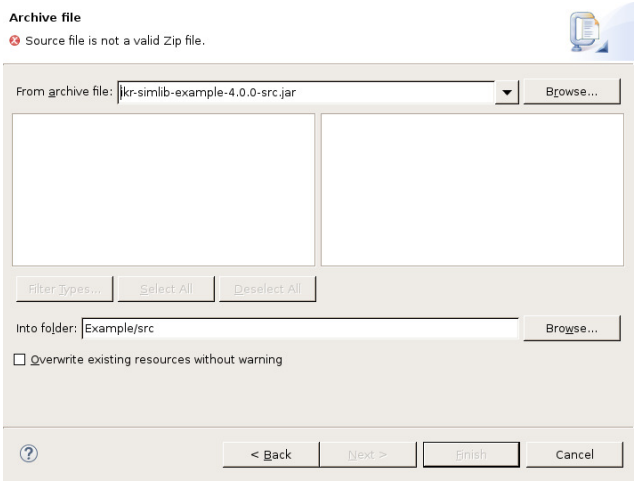

Figure 3.5: Create new Eclipse Project

The example requires that the IKR SimLib is included in the Java class path. To add it, right click on the new project to open the context menu, and select the Properties item.

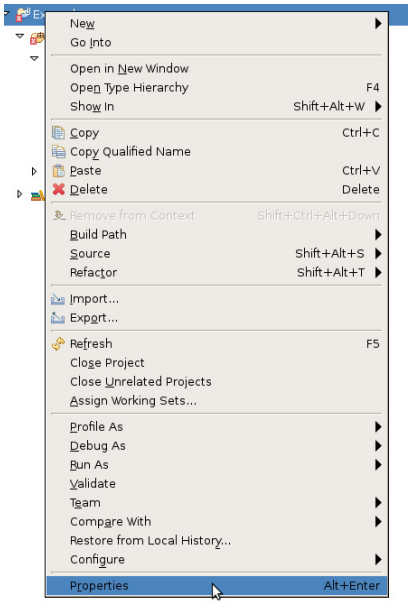

Figure 3.6: Configure the build path

Navigate to the Java Build Path configuration and add an external jar.

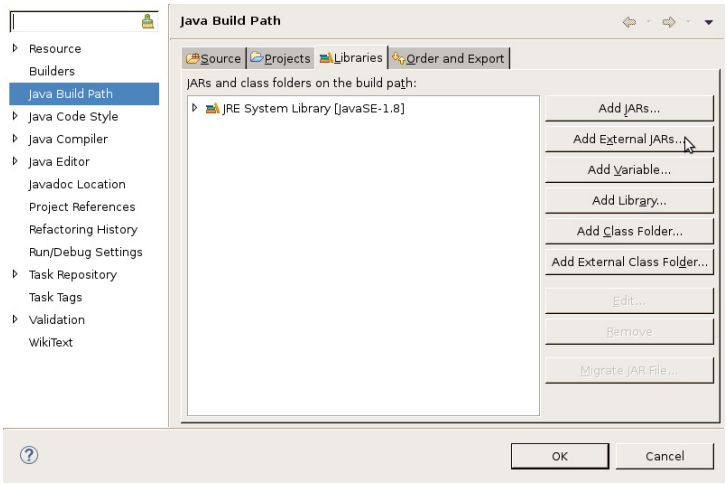

Figure 3.7: Configure the build path

Finally copy or move the file sim.par provided into the root directory of the newly created project.

3.1.3 Running the simulation

It is also possible to run the simulation directly within Eclipse. To do so open the class Main and press the the run button of Eclipse. This will run the simulation with the default settings and generate the file sim.log in the root directory of the project. Viewing the results is possible by navigating in the shell to the roor of the project and executing

```
STResultViewer --rfn sim.log
```

Alternatively you can open the file directly in a text editor of your choice.

3.1.4 Exporting the simulation model

To use your implemented model, it is convenient to export it as a runnable jar file. This can be done using the assistant of Eclipse. Select your project and open the context menu with a right click. Select Export.

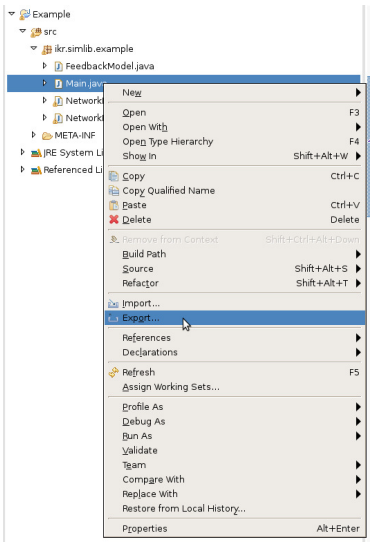

Figure 3.8: Export the model

In the next dialog select Runnable Jar file and click on Next.

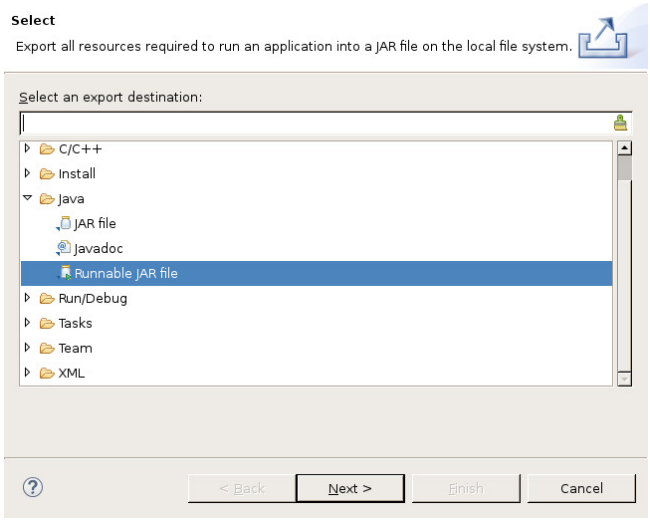

Figure 3.9: Export the model

Then select the launch cofiguration. In our case it is the Main class from the Example project and provide a location for the new jar file. All other properties can be left to their defaults.

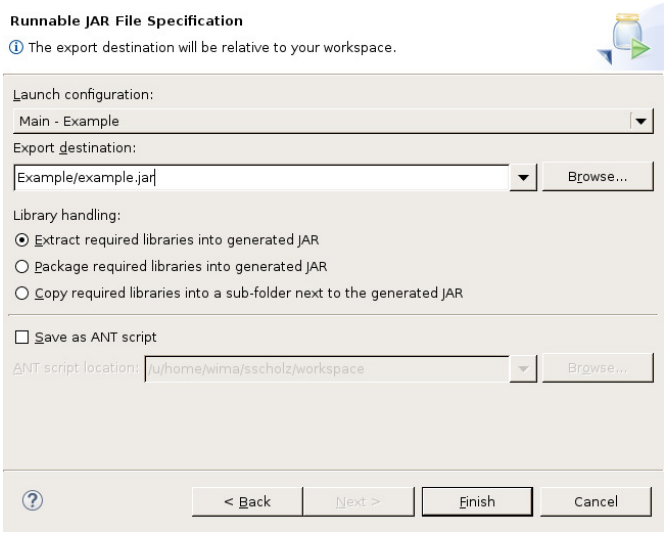

Figure 3.10: Export the model

3.2 Creating the simulation tree

To obtain results by simulation, the simulation program has to be executed several times with different value sets for the parameters of the simulation. The user has to define these parameters. In our example these parameters are the mean IAT (InterArrival Time) and the message length.

The tool SimTree is used to run the simulation program with varying parameter value sets and to collect and evaluate the simulation results. SimTree contains a build-in help system. A short introduction to its operations may be obtained by running the command

`SimTree intro`

SimTree maintains all data for a simulation study in a directory of the filesystem, the so-called study root. To use SimTree in our example a new subdirectory is created and prepared for using by SimTree by running the command.

`SimTree prepare`

Unless otherwise defined, SimTree uses the current subdirectory as study-root.

The parameters to be varied (The customizable simulation parameters) must be defined in a template parameter file which is located in the subdirectory MetaDeta below the study-root, i.e. ./`MetaData/sim.par.template.` The parameters are defined therein by variables in the format '`%%Variable%%`'. This file is provided together with the simulation example. The mean IAT of the Generator is defined by '`%%IAT%%`', the message length by '`%%MsgLen%%`'.

SimTree maintains the parameter value sets in a results tree which is located in the subdirectory `./Results` below the study-root. In the next step the results tree is populated with the simulations parameters and their values obtained during the simulation study. SimTree associates a type to each simulation parameter. The type could be `int`, `float`, `string` or `boolean`. The parameters in our example are both of type `int`. The values of parameter IAT and parameter MsgLen shall vary over the values 1, 2, 5, 10, 20 and 50, 100, 150, 200, 250, 300, 350, 400, 450, 500, 700 respectively. Following SimTree command creates the results tree with both parameters and their respective vealues:

```
SimTree add --simulation-parameter IAT int 1 2 5 10 15
--simulation-parameter MsgLen int 50 100 150 200 250 300 400 500
```

Most SimTree options have an abbreviation. Thus, the following command is equivalent to the previous one:

```
SimTree add --sp IAT int 1 2 5 10 15 --sp MsgLen int 50 100 150
200 250 300 400 500
```

3.3 Running the simulation

Now the simulation program has to be called for all combinations of values of both parameters. This is also done by SimTree. SimTree can execute the simulation program on the local host or on remote hosts. For now, the local host should be used. For this purpose, run `SimTree` with the option `simulate` and the parameter `--local-process-slots` which specifies the number of simulations on the local machine to be run in parallel. Further, pass the name of the simulation binary with `--simulation-binary` and the number of batches to run with `--batches`. The file `run.sh` must be executable.

```
SimTree simulate --local-process-slots 4 --simulation-binary
run.sh --batches 10
```

Again an abbreviated version of the command is also possible:

```
SimTree simulate --lps 4 --sb run.sh --b 10
```

The argument of parameter `--simulation-binary (--sb)` is the relative path of the simulation binary or script. If instead of a relative path only a file name is given (as in the above example), the binary is assumed to be located in the subdirectory `./MetaData` of the study-root. Following bash script is an example of `run.sh` that starts a java-based simulation program which is archived in a jar file:

```
#!/bin/sh
java -Xms128M -Xmx1024M -cp /net/arch/ikr/simlib-java/ikr-sim-
lib-example-4.0.0.jar:/net/arch/ikr/simlib-java/ikr-simlib-
4.0.0.jar ikr.simlib.example.queuingModel.Main "$@"
```

In case you want to use a runnable jar file your created from your own simulation model created in step 3.1.4, the file should look like the following:

```
#!/bin/sh
java -Xms128M -Xmx1024M -jar $(dirname $0)/example.jar "$@"
```

In this case either provide the absolute path to the jar file or copy it into the `MetaData` directory of the studyroot and use the file as provided above.

The options `-Xms` and `-Xmx` define the initial and maximum Java heap size respectively. The argument `-cp` (classpath) tells the Java Virtual Machine where to look for user-defined classes, packages and jar archives. `ikr.simlib.example.queuingModel.Main` defines the class which includes the `main()` method to start the application. SimTree passes simulation options, e.g. number of batches, simulation parameter filename, and final result filename, to the specified simulation binary. In above example, the bash script appends these options to the command line of the java program at the place of the argument `"$@"`.

Depending on the complexity of a simulation and number of runs, it may take several minutes (or even several days for more complex simulations) until the simulation is finished. A particular results tree including all completed runs and running tasks can be shown running following command within the study-root directory of a simulation:

```
SimTree list
```

An example output looks as follows:

```
IKR SimTree Version 2.7.0beta15 (C) 2013 University of Stut-
tgart, IKR
[Notice]  Using StudyRoot /u/home/wima/sscholz/tmp/SimlibExam-
ple/.
[Notice]  Launching SimTree list
[Config]  Using the following merged options
[Config]    (AutoDef) --print-format %%indent%%%%content%%
[%%spname%% %%value%%]
[Notice]  Simulation method is BatchMeans
[Notice]  Result tree contains 2 parameters
[Notice]    Parameter IAT, Type int
[Notice]    Parameter MsgLen, Type int
[Notice]  Meaning of symbols
[Notice]    F - - - - - - -  Final results exists
[Notice]    - E - - - - - -  Directory is empty
```

```
[Notice]    - - L - - - - -  Directory is locked
[Notice]    - - - B - - - -  Default batches exists
[Notice]    - - - - S - - -  Seeded results exists
[Notice]    - - - - - R - -  Export results exists
[Notice]    - - - - - - X -  Seed directories are locked
[Notice]    - - - - - - - W  Directory is not empty (may contain
foreign files, see Option --force-delete-foreign-files)
[Notice]  Content of Study root
F - - B - - - -  [IAT 1] [MsgLen 100]
F - - B - - - -  [IAT 1] [MsgLen 250]
...
F - - B - - - -  [IAT 2] [MsgLen 250]
F - L B - - - -  [IAT 2] [MsgLen 300]
- E - - - - - -  [IAT 2] [MsgLen 400]
- E - - - - - -  [IAT 2] [MsgLen 500]
...
- E - - - - - -  [IAT 15] [MsgLen 400]
- E - - - - - -  [IAT 15] [MsgLen 500]
[Notice]  40 simulation parameter value sets listed
[Status]  0 Errors, 0 Warnings
```

Activity and memory consumption can be checked using the `top` command. The directory `Results` contains the simulation results.

Longer simulation studies can be started in a non-interactive way with the `nohup` command. To do this instead of calling `SimTree simulate ...` directly, execute

```
nohup SimTree simulate ... &
```

4 Visualization of the simulation results

When all simulation runs are completed SimTree writes the results in log files (default `FinalResults.log.bz2`). Following tools can be used for the visul representation of simulation results:

- **xmgrace:** A tool for drawing graphics from values in ascii files which contain the values in a tabular style. xmgrace and the according documentation is available at [2] . Additional documentation for xmgrace is available by man pages (e.g. type `man xmgrace` on the shell command line).
- **Matlab:** A tool with a numerical computing environment and its own programming language. Matlab's toolbox `plottools` supports also drawing graphics from different formats such as ascii and XML files. The Matlab Getting Started Guide is available at [3] .

The objective of this example is to draw a diagram containing the transfer time as a function of the message length with respect to the service time. A further parameter is the interarrival time which yields a set of lines in each diagram.

The previously mentioned log files include the raw results. These files need first be processed in order to extract the data in a tool compatible format.

4.1 First overview

To get a first overview you can use the integrated graphical user interface of `SimTree`. To start the graphical user interface execute

```
SimTree gui
```

in the simulation directory.

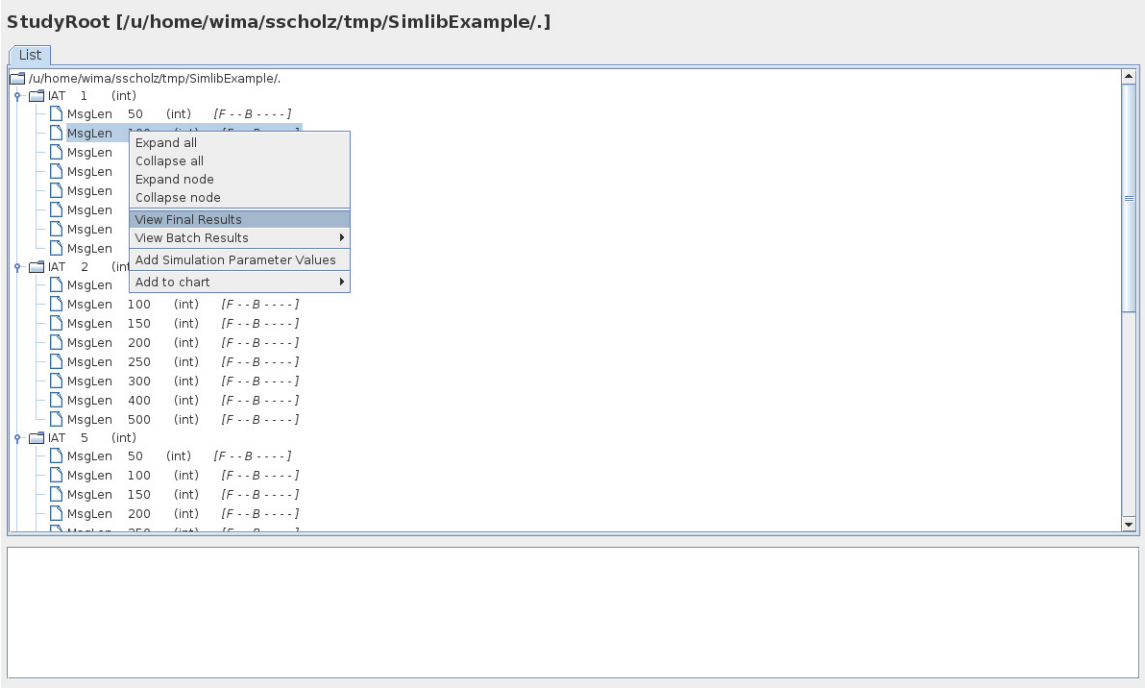

Figure 4.1: SimTree gui

The simulation results can be shown by clicking on "View Final Results".

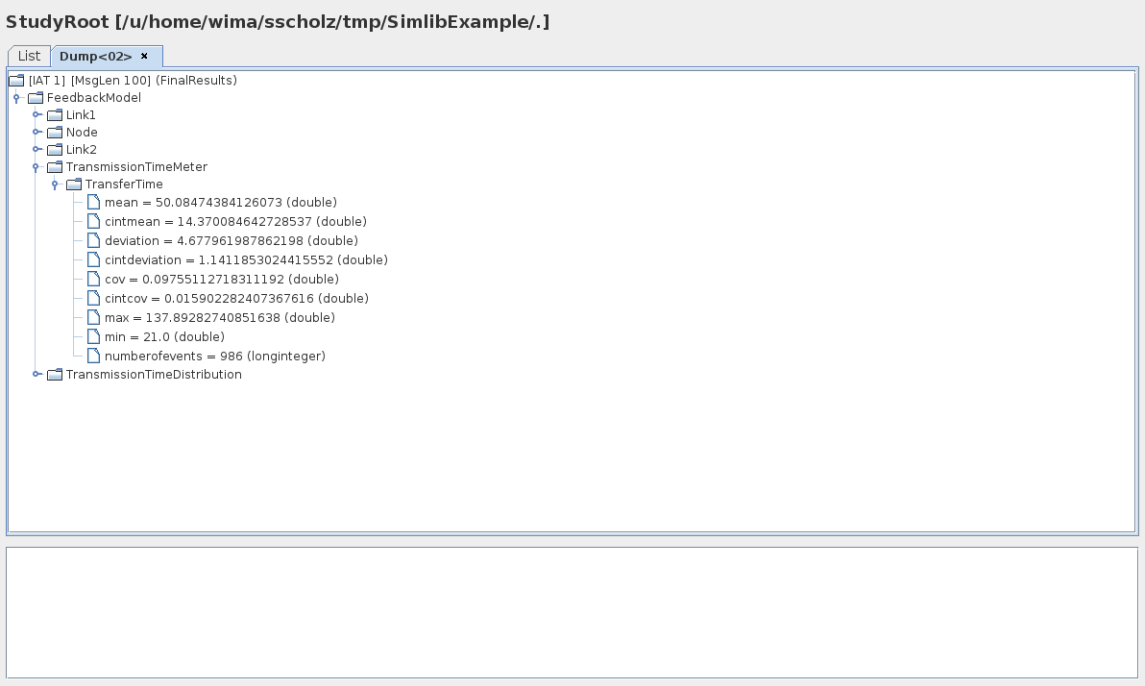

Figure 4.2: Show Final Results in SimTree gui

4.2 Value extraction

The first line in the diagram we are going to draw within the scope of this example only considers simulation runs with an IAT of 10. The results of these runs can be found in the subdirectory `Study_Root/Results/ParaI__IAT__10`. All simulation results need to be extracted and saved in a format compatible with the input format of graph-creating tools like xmgrace and Matlab. This is done by executing following SimTree command within the study-root directory:

```
SimTree eval --simulation-parameter IAT 10 --column-definition
0 sp MsgLen --column-definition 1 rp "FeedbackModel:Transmis-
sionTimeMeter:mean" --column-definition 2 rp "Feedback-
Model:TransmissionTimeMeter:cintmean" > data10.dat
```

Alternatively you can use the following abbreviated version of the command

```
SimTree eval --sp IAT 10 --cd 0 sp MsgLen --cd 1 rp "Feedback-
Model:TransmissionTimeMeter:mean" --cd 2 rp 2 rp "Feedback-
Model:TransmissionTimeMeter:cintmean" > data10.dat
```

The option `eval` allows the evaluation of simulation results by creating tables. Each column of this table has to be specified by the parameter `--column-definition`. In addition, each column has a number, a type and an additional argument which defines what appears in this column. Column numbers starts from 0, are consecutive and do not include spaces. The type may be either a simulation-parameter or a result-path. The simulation-parameter and result-path types are abbreviated with `sp` and `rp` respectively. The third argument depends on

the column type: in case of a simulation-parameter it is the name of the simulation parameter and in case of a result-path it is the hierarchy path of the `SimNodes` (i.e. Feedback-Model:TransmissionTimeMeter:TransferTime:mean is constructed by the headings of the <ResultNode> tags from `FinalResults.log.bz2` and thus represents the hierarchical structure). For futher options use:

```
SimTree eval --help
```

4.3 Graph Creation

4.3.1 Basic Graph

Invoke *xmgrace* and Import the data file `data10.dat` via the menu *"Data->Import->Ascii"*. Using the file extension *dat* is expected by *xmgrace*. Fig. 4.3 shows the import dialog

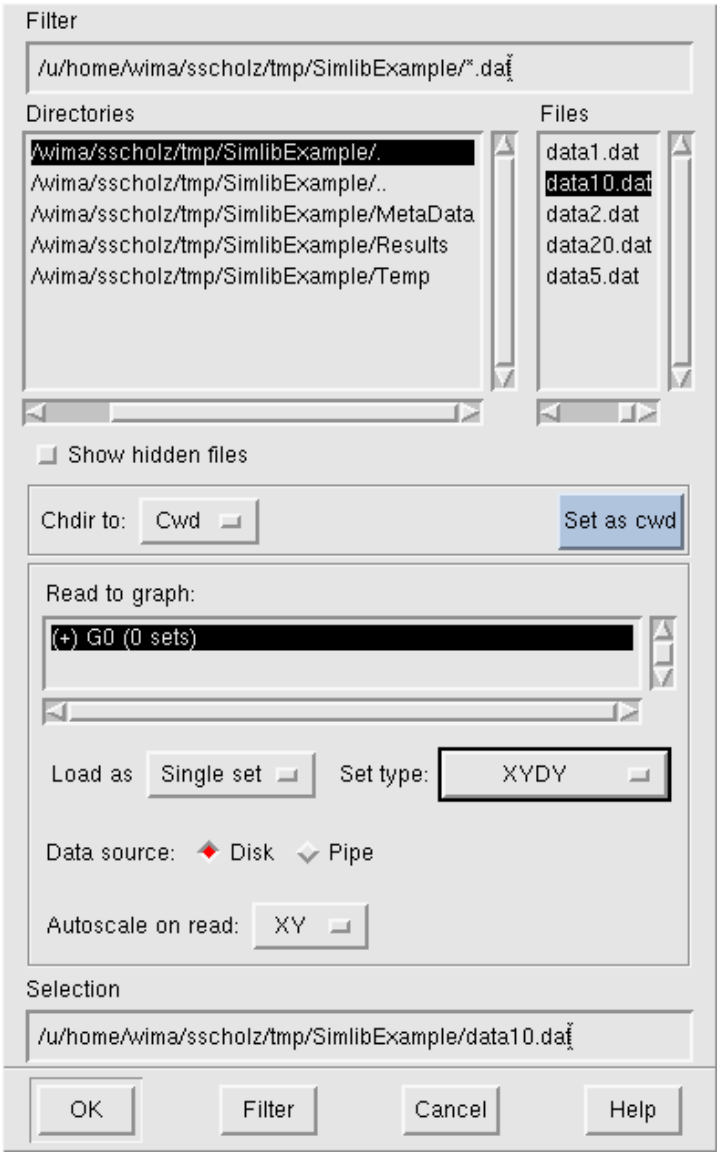

Figure 4.3: Import of the date file to xmgrace

of xmgrace. Of utmost importance is the field *set type*. It describes the type of input data. The format *XYDY* complies with an input file with three columns the first one of which includes

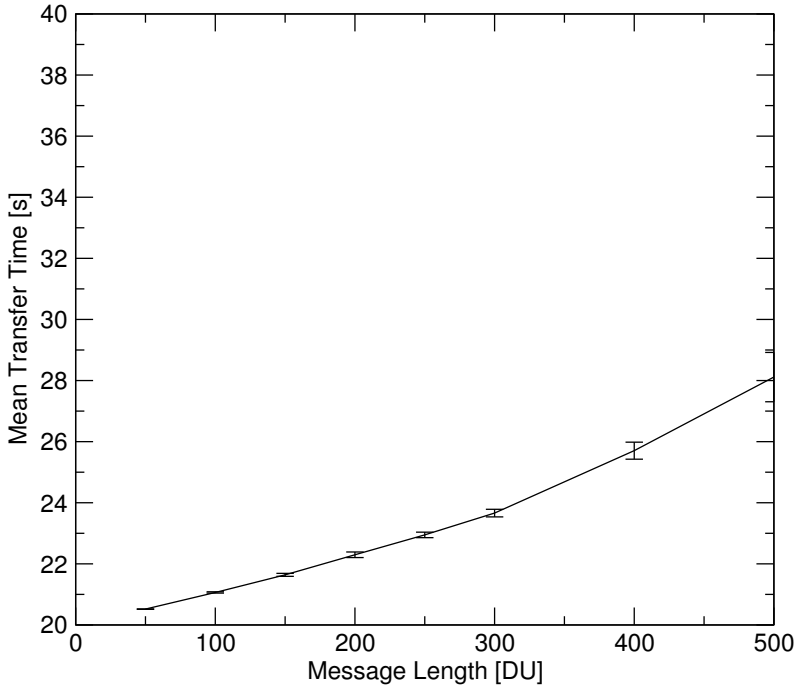

Figure 4.4: Result of the first import

values for the x-axis, the second one for the y-axis and the third column contains the error bar. The error bar characterizes the variation of the y value and shows how accurate the value is. In general a longer simulation results in smaller error bars.
It is also possible to determine the diagram type in the *.dat* file itself by inserting `@type xydy` in the first line.

The result of importing file `data10.dat` to *xmgrace* is shown in Fig. 4.4

Repeat the steps of section 4.1 (i.e. run `SimTree eval ...`) for the remaining interarrival times (i.e. IAT = 1, 2, 5 and 20). This time turn the option *Autoscale on read* to *None* (for all four imports) to avoid an inconvenient scale modification. Finally, Fig. 4.5 shows the result.

What can be clearly seen in Fig. 4.5 is the constant delay of 20s introduced by the two infinite servers.

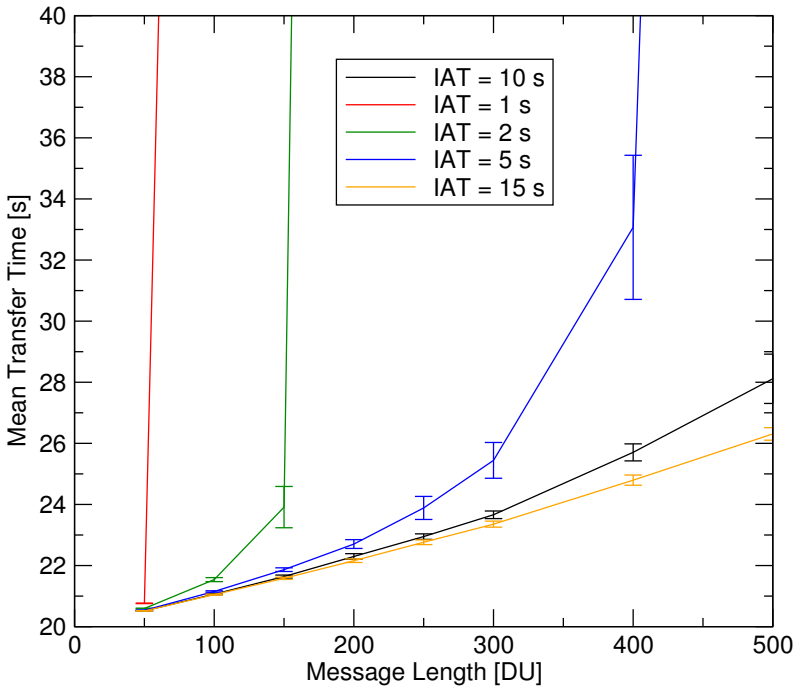

Figure 4.5: All curves

5 Hints for using the IKR computing infrastructure

If you perform simulation in the IKR, you can make use of the provided computing infrastructure, which consists of several powerful computing nodes. Besides that also a job scheduler exists, that automatically assigns simulation jobs on the available nodes. To make use of the infrastructure you have first to enable communication with the job scheduler by editing the configuration file of SimTree. This file is called `simtree.properties` and is located in the MetaData directory of your studyroot. Open this file and search for the following line and set the parameter to `true`:

```
# enable the cooperation with the SimTree Global Scheduler
st.b.enable_global_scheduler_cooperation.1 = false
```

Instead of running the SimTree simulate locally, you have first to connect to one of the available computing nodes via ssh, e.g. `ssh cnode01`. The tool `cnodestate` provides an overview of the computing nodes and their utilization.

After connecting to a cnode, run the simulation using the following command:

```
SimTree simulate --gps --sb run.sh --b 10
```

The parameter `--gps` (--global processing slots) tells SimTree to request slots from the job scheduler. By default the slot request includes one CPU core and reserves 5GB of memory on the computing node. If you have other requirements, e.g. if you used the java parameter `-Xmx` to increase the heap size, you can start the simulation with the following command:

```
SimTree simulate --gps --sb run.sh --b 10 --pr cores=2 mem=8G
```

This will request 2 CPU cores and 8GB of memory.

For longer simulations use either the `nohup` or `screen` commands

6 ACK/NACK Protocol

In this section, we show how a simple network protocol can be implemented. When executing the model, make sure that the correct parameter file is used (sim-ackNack.par). The parameter file can be specifed with the command line parameter "-p sim-ackNack.par".

6.1 Model description

The purpose of the model is to measure the one way delay of packets betwenn generator and sink. The packets are transmitted by the sender using the ACK/NACK protocol. The receiver tries to decode the packets. If this is possible, the packet is forwarded to the sink and an ACK is sent back to the sender. If decoding is not successful a NACK is sent to the sender. In this case the sender has to repeat the previously sent packet, otherwise the next packet may be transmitted. For simplicity we assume that no packets are lost completely and only the forward packets are disturbed. ACK or NACK messages are always received correctly.

Fig. 6.1 shows all components of the model.

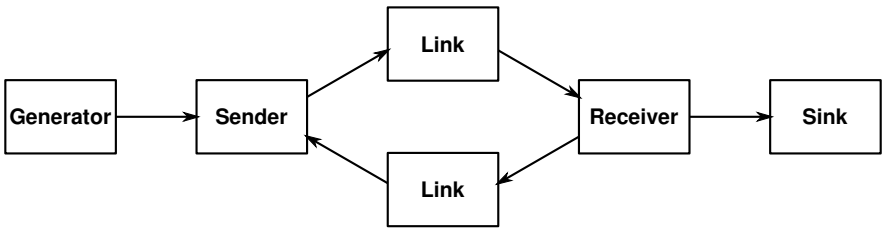

Figure 6.1: Model

6.2 Possible Extensions

- Implement the behavior of the Sender using a real state machine
- Change the generator, such that the packet size is variable
- Add a sequence number in the packets generated by the generator
  Hints:
  - Derive a new class from the class `Message`
  - Create a `MessageFactory` and pass it to the generator
- Include a time-out to allow complete losses of packets

References

[1] P. J. KÜHN, A. KIRSTÄDTER, “Performance Modelling and Simulation,” Scriptum of the lecture, IKR, University of Stuttgart, Edition 2012.

[2] http://plasma-gate.weizmann.ac.il/Grace/

[3] http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/getstart.pdf