LINE: a fluid performance engine

Version 0.7

Last modified: July 2, 2015

Contents

1 Introduction

LINE is a tool for the performance analysis of software applications, which has been designed to automatically build and solve performance models from a high-level description of the application. This analysis is performed by analyzing a layered queueing network

(LQN) model that is obtained from a model-to-model transformation from the high-level application model. Starting from this release, LINE also offers support for models in the Performance Model Interchange Format (PMIF) and in the Business Process Modeling Notation (BPMN). This support is achieved by model-to-model transformations from these models to the LINE performance model, which is a fluid queueing network model. This document contains installation and usage instructions, and presents the new features included in version 0.7. The appendix also includes a more detailed description aimed at developers.

LINE has two main operation modes:

- The first operation mode is to directly use the MATLAB scripts. This allows for solving LQN, PMIF, BPMN, or closed queueing network models. Example scripts are provided on the examples folder that comes with the source code.
- The second operation mode is to use the LINE binaries. In this manner, LINE can directly interact with the Palladio Bench and the SPACE4Cloud [1] tools. The LINE binaries operate as a server that receives commands to solve application performance models. This allows LINE to operate efficiently, as it only needs to be started up once to solve as many models as needed. In addition, LINE also has a parallel-execution option to further reduce the model solution times. We have included detailed instructions on how to configure and run LINE together with these tools.

2 New features

The main new features in version 0.7 are:

- LINE now supports the performance evaluation of BPMN models. LINE reads the BPMN model in the standard BPMN XML format, which it analyzes by means of a model-to-model transformation to LQN models. Examples of how to use this feature are provided in the scripts *example_BPMN_X.m* included in the examples folder.
- LINE now supports the performance evaluation of PMIF models. LINE reads the PMIF model in the standard PMIF XML format, which it analyzes by means of a model-to-model transformation to LQN models. Examples of how to use this feature are provided in the scripts *example_PMIF_X.m* included in the examples folder.
- LINE now offers the QD-AMVA solver, which has been developed to analyze queueing networks with queue-dependent processing times [2]. This solver by setting a new configuration variable, called *solver*, to the value *QDAMVA*. The default solver is the existing fluid solver.
- The source code in LINE has been re-structured, facilitating its extensibility, particularly to consider different input models and solvers.

The usage of LINE is described in Section 4.

Bug fixes in this version:

- Bug fix in the computation of the mean response times in the mainSolver scripts.
- Bug fix for networks with non-reference delay stations.
- Bug fix in the treatment of the s vector (number of servers) as a column vector. This is fixed in both the parser and the solver scripts.
- Bug fix to return empty resSEFF and resSEFF_CDF objects when the SEFF is not specified in a solver.

3 Installation Instructions

There are two main ways of using LINE: directly with the MATLAB scripts or via the binaries to interact with other tools.

3.1 Using the LINE scripts

This is the quickest way to start using LINE if you have a MATLAB distribution and a basic understanding of MATLAB code.

1. Download the source code of LINE version 0.7. This can be done by checking out the code using an SVN client from the location

   ```
   svn://svn.code.sf.net/p/line-solver/code/trunk/releases/v07
   ```

2. Next, start MATLAB and add the `src` folder, and its subfolders, to the path. This is achieved by typing the command

   ```
   addpath(genpath('C:\path\to\line\src\'))
   ```

3. Go to the folder `src\examples` and open in MATLAB any of the examples provided. Make sure to set the MATLAB current folder to the folder with the examples. This allows MATLAB to locate the sample data files provided. Run the script and see the results either on screen or in the output files generated.

3.2 Using the LINE binaries

There are three alternative ways of using the LINE binaries: from a client, or in conjunction with either SPACE4Cloud or Palladio Bench.

3.2.1 Common steps: LINE and the MCR

The following steps are common to all three alternatives.

1. Download the LINE executable file from the `Releases` section on the LINE website `http://line-solver.sourceforge.net/`.
2. As LINE has been built as a MATLAB application, it requires the installation of the MATLAB Compiler Runtime (MCR), freely available at `http://www.mathworks.co.uk/products/compiler/mcr/`.
   (a) Windows: download and install the MATLAB Compiler Runtime (MCR), version 2012b.
   (b) Linux: download and install the MATLAB Compiler Runtime (MCR), version 2013a.

LINE is now ready to use from a client (see Section 4).

3.2.2 Palladio Bench (with SPACE 4Cloud patch)

1. Export the location of LINE to the PATH system variable, so that Palladio can find it.
2. Download Palladio Bench from `http://www.palladio-simulator.com/tools/download/`. Download is free but registration is required.
3. Unzip the Palladio Bench distribution, and launch the Eclipse SDK therein.
4. In Eclipse, go to *Help→Install new software*. Click the *Add* button and as *Location* specify the URL

   `ftp://ftp.modaclouds.eu/public/space4cloud/eclipse-update-site`
5. Clear the *Group items by category* check box.
6. Select the *PCM Solver Feature*, click *Next* and complete the installation process, which requires re-starting Eclipse.

LINE is now ready to use. Go to *Run→Run Configurations*, and select LINE in the *Solver* tab.

3.2.3 SPACE 4Cloud

To install SPACE 4Cloud follow the instructions above, but in the last step Select both the *PCM Solver Feature* and *SPACE 4Clouds*. Complete the installation, including re-starting Eclipse. The SPACE 4Clouds icon should now appear in the tool bar.

LINE can be used in SPACE 4Clouds by selecting it from the *Performance Engine* list in the *Functionality Selection Panel*.

4 User's Guide

The LINE binaries can be used in three different ways. Before describing each of them we show how to start LINE and the configuration options available when executing LINE.

4.1 Starting LINE

Starting from version 0.5, LINE operates as a server, thus it must be first started, providing some configuration information. After this initialization step, LINE accepts commands to solve models, to close the current connection, or to terminate it. After initialization, LINE can solve as many models as needed by simply submitting the appropriate commands.

To start LINE, execute the following command on the command line:

- In Windows:

```
LINE "LINE.properties"
```

- In Linux:

```
./run_LINE.sh /path/to/MCR/v81/ 'LINE.properties'
```

Here `LINE.properties` is a properties file that contains the configuration information for LINE. Currently, LINE supports the following properties, grouped in three sets:

1. Operation:
   (a) `port`: port on which LINE will listen to commands from the client. Default value: 5463.
   (b) `timeoutConnection`: maximum time, in seconds, that LINE waits for new commands before closing the current connection. Before closing a connection, LINE completes any outstanding jobs. Default value: 30.
   (c) `verbose`: enables verbose screen output if set to 1. Otherwise, limited screen output is provided. Default value: 0.
   (d) `parallel`: allows LINE to exploit the possible gains of solving models in parallel. Three options are offered:
   - SEQ for sequential execution (default);
   - JOB for parallel execution using Matlab parallel engine for batch job execution;
   - PARFOR for parallel execution using the parfor mechanism in Matlab.

(e) `maxJobSize`: maximum number of models to solve as a single parallel job. As the JOB parallel execution option poses some overhead, it is in general a good idea to put together a number of models and solve them as a single job. In case the models are large or computationally heavy to solve, this number should be small. In the case of simple models, this number can be large. Default value: 12.

2. Solution methods:

   (a) `solver`: solution method used to obtain the performance metrics. Two options are offered:
   - FLUID: the fluid solver based on a system of ordinary differential equations [3]. Fast and accurate when the number of servers and users is large (tens). Supports Coxian processing times and random environments.
   - QDAMVA: method based on mean-value analysis that considers queue-dependent processing times [2]. Fast and accurate for a broad range of conditions, including small numbers of servers and users. Does not support Coxian processing times nor random environments.

   Default value: FLUID.

   (b) `maxIter`: maximum number of iterations of the blending algorithm when solving a model with random environments. Supported by the FLUID solver option. Default value: 1000.

3. Results:

   (a) `respTimePerc`: when set to WORKLOAD, it activates the computation of response time percentiles, at the level of the workload. The default value is NONE, which avoids this computation. Other options will be offered in future releases.

   (b) `respTimePercMin`: if the computation of response time percentiles is activated, this parameter determines the minimum percentile to compute. Default: 0.05.

   (c) `respTimePercMax`: if the computation of response time percentiles is activated, this parameter determines the maximum percentile to compute. Default: 0.95.

   (d) `respTimePercStep`: if the computation of response time percentiles is activated, this parameter allows the computation of many percentiles between the minimum and maximum set above. Default value: 0.05. For instance, using the default values, the percentiles computed are $[0.05, 0.10, 0.15, \ldots, 0.90, 0.95]$.

4.2 Using LINE from SPACE4Cloud

When evaluating an application's performance with SPACE4Cloud [1], the tool prompts the user to choose a performance solver, offering LINE as one of the built-in options.

The details on how to use SPACE4Cloud can be found in its documentation, available at `http://www.modaclouds.eu/software/space4cloud/`.

4.3 Using LINE from Palladio Bench

Using LINE with the Palladio Bench tool is as simple as using any of the built-in solvers. The following steps are provided for illustration.

1. Import or create a new project.
2. Go to *Run Configurations*, and select (double-click) *PCM Solver*.
3. Provide a name, and select the Allocation and Usage models from the application model project.
4. In the *Solver* tab, select LINE from the drop-down list, indicate the output directory, and check (or clear) the *verbose* option to activate (deactivate) screen debugging information.
5. Click *Run*. The results will be stored in an XML file on the output directory.

4.4 Using LINE from a client

After LINE has started, a confirmation is given with the screen message `LINE is listening on port 5463`. The next step is to connect to LINE via the defined port, which is illustrated in the example provided in the distribution[1] Once the connection is established, LINE will submit the message `LINE READY`, indicating that it is ready to receive solution commands. A solution command is as follows

```
SOLVE path/to/LQNfile/LQN.xml
```

Here `LQN.xml` is the XML file holding a Layered Queueing Network (LQN) model. This file can be generated, for instance, from Palladio Bench. A command could also be

```
SOLVE path/to/LQNfile/LQN.xml path/to/EXTfile/EXT.xml
```

Here, in addition to the LQN model, the `EXT.xml` file describes LQN extensions, which currently cover two main objects: random environments and Coxian distributions. Details and examples on how to specify these extensions are given in Appendix A. Many of these commands can be submitted, and LINE will process them sequentially or in parallel, depending on the `parallel` property discussed above.

As LINE works as a server, a client establishes a connection with LINE before being able to submit `SOLVE` commands. To close this connection, the client can submit the `CLOSE` command. Also, the connection will be closed by LINE after a timeout, which

[1] In the repository under `http://svn.code.sf.net/p/line-solver/code/trunk/releases/v07/test`.

can be modified with the `timeoutConnection` property. Before closing the connection, LINE will wait for all the SOLVE jobs to complete, ensuring that all models are solved.

Finally, the user can terminate LINE by submitting the command QUIT, which will cause LINE to close the connection and terminate, after all the outstanding (running and queueing) jobs have completed.

Example An example of a client connecting and submitting commands to LINE can be found in the *test* folder of the 0.7 release, that can be downloaded from the *Releases* page on `http://line-solver.sourceforge.net/`.

5 Where to download and how to cite LINE

The LINE binaries and code released can be downloaded from

`http://line-solver.sourceforge.net/`

You can refer to LINE by citing the following paper:

J. F. Pérez and G. Casale, "Assessing SLA compliance from Palladio component models," in *Proceedings of the 2nd Workshop on Management of resources and services in Cloud and Sky computing (MICAS)*, IEEE Press, 2013.

A LQN Extensions

In this section we present an example of an XML file describing an LQN extensions. There are two sets of elements: random environments and Coxian distributions. The example XML and the associated XML schema can be found in the LINE repository[2].

A.1 Random environments

To *define* a random environment (RE), the following elements are involved:

1. **environment**: defines an RE.
   - Attributes
     - numStages: number of stages in the RE.
     - envID: identifier of this RE.
   - Children:
     - stage
2. **stage**: defines each of the stages in the RE.
   - Attributes
     - name: name of this stage.
   - Children:
     - transition
     - stageTime
3. **stageTime**: defines the sojourn times in each stage, which can be exponential or Coxian.
   - Children (exclusive or):
     - meanTime: mean sojourn time in this stage, if these times are exponentially distributed.
     - coxID: id of the Coxian distribution element (see next section) that describes the sojourn times.
4. **transition**: defines a transition between to stages in the RE.
   - Attributes
     - destName: name of the stage reached with this transition
     - prob: probability that this transition occurs. The sum of all transition probabilities in a stage must add up to one.
   - Children:

[2]http://svn.code.sf.net/p/line-solver/code/trunk/doc/support/LQNextensions/

  - resetRule

5. **resetRule**: defines the reset rule applied when a transition occurs. This reset rule determines how the service phases, in a Coxian distribution, evolve when a stage transition occurs.

   - Attributes
     - ruleName: type of reset rule. Supported rules include noReset and fullReset.

The example shown in Figures 1 and 2 defines an RE named *environment* and composed of 3 stages. The sojourn time in the first stage, named SU, is described by a Coxian distribution, with ID *coxStage1*. The XML element describing this distribution is in Figure 2, and is described in detail below. After a visit to this stage, a transition to either stage LC or HC occurs with probability 0.5, following the noReset rule. For the stages LC and HC, the *stageTime* element is not described with a Coxian ID, but with the mean time of an exponential distribution. In this case we simply set the mean sojourn times in these phases to be 100 and 200, respectively. Other stages and transitions are defined similarly.

After defining an RE, we can use it to modify the value of some parameter values as a function of the RE stages. This is done by defining *environmental parameters*, which involves the following elements.

1. **envParameter**: defines an environmental parameter.

   - Attributes
     - id: ID of the LQN processor associated to this environmental parameter.
     - paramName: name of the parameter (in the processor identified above) that is affected by the environment. Current supported options are: speed-factor, and multiplicity.
     - envID: identifier of the RE that affects this parameter.
   - Children:
     - envValue

2. **envValue**: defines the values taken by the parameter in each environmental stage.

   - Attributes
     - stage: stage of the RE to which this value is associated.
     - factor: factor by which the value in the model is multiplied to obtain the actual value of this parameter in this stage of the RE.

The example shown in Figure 1 defines an environmental parameter associated with the *VMContainer_CPU_Processor* processor. The actual parameter is the *speed-factor* of this processor, which is affected by the RE with ID *e1*, defined above. The speed-factor of this processor is: equal to the original in stage LC; 5 times the original one in stage HC; and a fifth of the original in stage SU.

Figure 1: Example XML file with LQN extensions - Part 1

```
<?xml version="1.0" encoding="UTF-8"?>
<lqnExtensions>
<environment numStages="3" envID="e1">
 <stage name="SU" >
   <transition destName="LC" prob="0.5">
    <resetRule ruleName="noReset"/>
   </transition>
   <transition destName="HC" prob="0.5">
    <resetRule ruleName="noReset"/>
   </transition>
   <stageTime>
      <coxID>coxStage1</coxID>
   </stageTime>
 </stage>
 <stage name="LC">
   <transition destName="HC" prob="1" >
         <resetRule ruleName="noReset"/>
   </transition>
   <stageTime>
      <meanTime>100</meanTime>
   </stageTime>
</stage>
<stage name="HC">
   <transition destName="LC" prob="1">
         <resetRule ruleName="noReset"/>
   </transition>
   <stageTime>
         <meanTime>200</meanTime>
   </stageTime>
</stage>
</environment>


<envParameter id="VMContainer_CPU_Processor" ...
    paramName="speed-factor" envID="e1">
   <envValue stage="LC" factor="1"/>
   <envValue stage="HC" factor="5"/>
   <envValue stage="SU" factor="0.2"/>
</envParameter>
```

Figure 2: Example XML file with LQN extensions - Part 2

```
<coxDistribution numPhases="2" coxID="er1">
 <phase meanTime="0.005" completionProb="0" phaseIndex="1"/>
 <phase meanTime="0.005" completionProb="1" phaseIndex="2"/>
</coxDistribution>

<coxDistribution numPhases="2" coxID="c1">
 <phase meanTime="0.005" completionProb="0.5" phaseIndex="1"/>
 <phase meanTime="0.01" completionProb="1" phaseIndex="2"/>
</coxDistribution>

<coxDistribution numPhases="2" coxID="coxStage1">
 <phase meanTime="10" completionProb="0.5" phaseIndex="1"/>
 <phase meanTime="10" completionProb="1" phaseIndex="2"/>
</coxDistribution>


<coxParameter ...
    id="InternalAction_main__Iu1-wMhoEeKON4DtRoKCMw_34_50_Activity" ...
    coxID="er1"/>
<coxParameter ...
    id="InternalAction_addcart__5JEHQMhoEeKON4DtRoKCMw_34_50_Activity" ...
    coxID="c1"/>

</lqnExtensions>
```

A.2 Coxian distributions

To *define* a Coxian distribution, the following elements are involved:

1. **coxDistribution**: defines a Coxian distribution.
   - Attributes
     - numPhases: number of phases in the Coxian distribution.
     - coxID: identifier of this Coxian distribution.
   - Children:
     - phase
2. **phase**: defines each of the phases in the Coxian. distribution.
   - Attributes
     - phaseIndex: index of this phase, as these must be ordered
     - meanTime: mean sojourn time spent in this phase
     - completionProb: probability that the processing times terminates after this phase

The example shown in Figures 1 and 2 defines a Coxian distribution named *er1* and composed of 2 phases. The mean sojourn time in the first phase is 0.005 time units, after which no completion occurs, and the service moves to the second phase. The mean sojourn time in the second phase is also 0.005 time units, after which the service terminates. Notice that this describes an Erlang distribution, with 2 phases, and rate 200 in each phase. Similarly, a second Coxian distribution is defined with id "c1".

After defining a Coxian distribution, we can use it to modify the execution times assumed for specific *activities* in the LQN model. This is done by defining *Coxian parameters*, which involves the following elements.

1. **coxParameter**: defines a Coxian parameter.
   - Attributes
     - id: ID of the LQN *activity* the processing time of which is modified to be Coxian.
     - coxID: identifier of the Coxian distribution used to described the processing times of this activity.

The example shown in Figure 2 defines a Coxian parameter that affects the activity identified as *InternalAction_main__Iu1-wMhoEeKON4DtRoKCMw_34_50_Activity*. The execution time of this activity is set to follow the Coxian distribution *er1*, defined above.

B LINE Developer's Guide

The sequence diagram in Figure 3 depicts the main steps in the operation of LINE. From version 0.5, LINE operates as a server, accepting connections from clients. In addition to the client, Figure 3 includes the four main components that make up LINE:

- LINE Server: Manages the connections with the Client, and performs the calls necessary to the other components to build and solve the LINE PM.
- LINE Parser: Reads the input model description and transforms it into a LINE PM.
- LINE Main Solver: Receives the PM and calls a solver to obtain the steady state distribution of the PM. It uses this distribution to compute mean performance measures. It also calls other solvers to obtain other metrics such as response time distributions.
- LINE Fluid Solver: Provides routines to determine the steady state distribution of a PM. It also provides routines to estimate response-time distributions for a PM.

Figure 3 illustrates how a Client first establishes a connection with the LINE Server. After a connection is established, the Client can submit models for solution. A model specification is composed of two parts: a main LQN model, and an *optional* extension (EXT) file, both in XML format. Once LINE receives a model for solution it parses the basic LQN model first, and then parses the EXT file. Parsing the LQN model has two main steps: first parsing the LQN XML file to reconstruct the LQN model, and then obtaining the LINE performance model (PM). This PM is then extended with the definitions included in the EXT file.

After these steps, the LINE PM is ready to be solved by the LINE solvers. The LINE Main Solver calls the Fluid Solver in two steps: first to find the steady state distribution of the PM, which it then uses to determine *mean* performance metrics; and second, to determine the response time *distribution*. After these step are completed, the LINE Main Solver returns the performance metrics to the LINE Server, which in turn exports them in an XML file and sends the client a "Model Solved" message. The main advantage of this operation is that LINE only needs to be initialized once to accept connections from different clients, and to solve any number of models during any connection.

B.1 LINE classes

We now describe the main classes and scripts that make up the LINE components. As LINE is implemented in MATLAB, it not only consists of a set of classes, but also of purely procedural scripts. We divide the classes in three groups, depending on whether they are used to describe LQN models, PMs, or others. The scripts are instead divided according to the components they belong to, as listed in Figure 3.

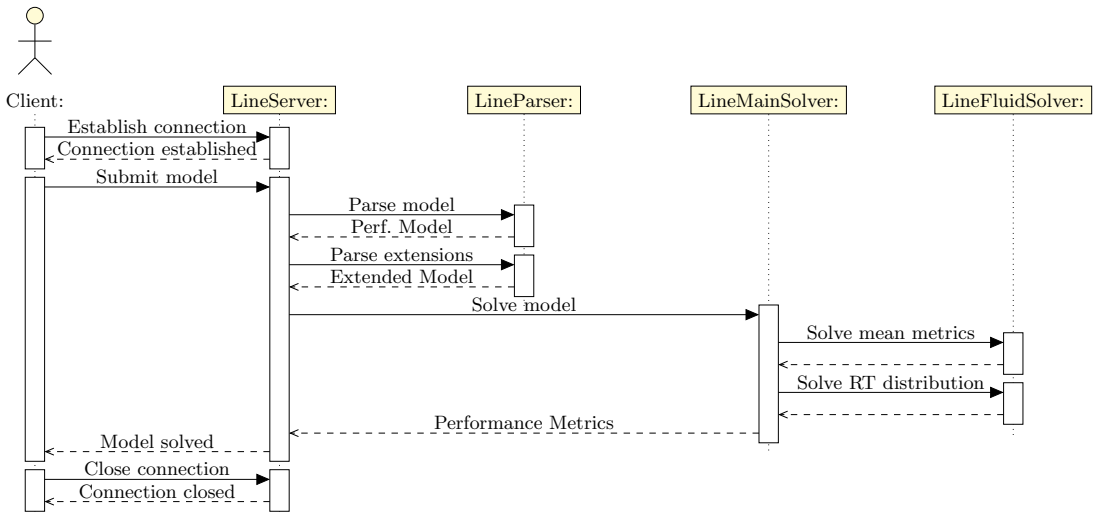

Figure 3: LINE Sequence Diagram

LQN classes This set of classes are used to describe LQN models, and their relations are illustrated in Figure 4. The classes are

- `Processor`: describes a processor in the LQN model. Its properties include a list of the tasks deployed on the processor.
- `Task`: describes a task in the LQN model. Its properties include a list of the entries and activities within the task. The activity graph and precedences describe how the activities are executed.
- `Entry`: describes an entry in the LQN model. Its properties include a list of the activities executed when the entry is called.
- `Activity`: describes an activity in the LQN model, which is the basic execution unit. Its properties include the mean demand of the activity on the resource where it is executed.
- `Precedence`: provides a link between two or more activities, which are used to describe the activity graph.

EXT classes These classes, depicted in in Figure 4, are used to describe two extensions to the LQN models: Coxian distributions for the processing times, and the Random Environments (RE) for reliability modeling. The classes are

- `RE`: describes a random environment, including its stages, transition rates, reset rules, and the parameters in the LQN model affected by the RE.

- `COX`: describes a Coxian distribution, including its states, transition rates, and completion probabilities, as well as the activities in the LQN model that follow this distribution.

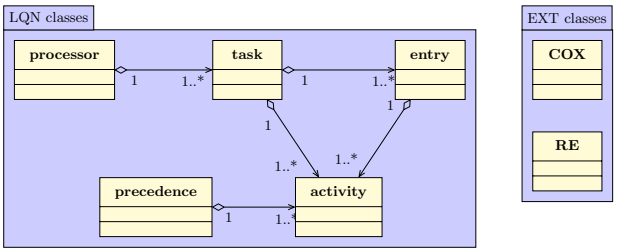

Figure 4: LINE LQN and EXTClasses

PM classes This set of classes, depicted in Figure 5 are used to describe the performance model (PM) underlying LINE. There are four classes that describe different versions of the PM:

- `CMCQNCS`: describes the basic PM underlying LINE. CMCQNCS stands for Closed Multi-Class Queueing Network with Class Switching, which is the basic PM. Its description can be found in [3].
- `CMCQNCSRE`: describes a CMCQNCS extended with a random environment (RE), which is used when an RE is specified.
- `CMCQNCSCox`: describes a CMQNCS extended to handle general (Coxian) distributions for the processing times.
- `CMCQNCSRECox`: describes a CMCQNCS extended to handle general (Coxian) distributions for the processing times, and a random environment.

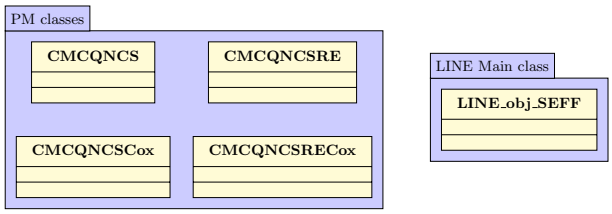

Figure 5: LINE PM and Solver Classes

LINE **Main class** As depicted in Figure 5, the main LINE class is `LINE_obj_SEFF`, which is part of the LINE Server, and is in charge of interacting with the LINE parser to obtain the PM, and its extension, and with the LINE Main Solver to obtain the performance metrics.

The `LINE_obj_SEFF` class is also in charge of implementing one of the key features of LINE: the parallel evaluation. There are three types of operation: Sequential, Parfor, and Batch Engine. The `LINE_obj_SEFF` class has these 3 modes, and one of them is used according to the configuration parameters. To support the Batch Engine mode, this class creates the connection to the local cluster, and maintains a list of the jobs processed and their status.

B.2 LINE scripts

In addition to the set of classes described in the previous section, LINE mainly consists of a number of procedural scripts, which implement the main functionalities belong to each of the main components.

LINE **Server** The scripts in this component are depicted in Figure 6.

- `LINE`: This script is called to start the LINE application. It receives the location of the configuration file as argument.
- `LINEserver`: This script manages the connections and receives the commands from the Client.
- `LINEprotocolXML`: This script parses the commands received from the client and submits the models for solution to the LINE Main class.
- `writeXMLresults_SEFF`: This script exports the results to an XML file. It is used by the LINE Main class.

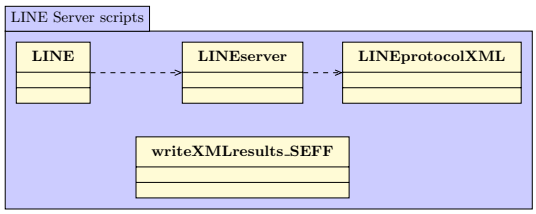

Figure 6: LINE Server Scripts

LINE **Parser** The scripts in this component are depicted in Figure 7.

- `parseXML_LQN`: this script reads the XML file containing the LQN description of the model to solve, and builds a set of objects using the LQN classes.

- `readXML_CMCQNCS_SEFF`: this is the main script to perform the transformation from the LQN objects to the PM description, which is returned in the form of a `CMCQNCS` object.
- `readXML_CMCQNCS_addEntries_SEFF`: this script supports the previous one by recursively exploring the LQN objects to build the PM.
- `parseXML_COX`: this script parses an extension (EXT) file to build one or more `COX` objects, which describe the Coxian distributions to extend the LQN model.
- `parseXML_RE`: this script parses an extension (EXT) file to build one or more `RE` objects, which describe the Random Environment (RE) to extend the LQN model.

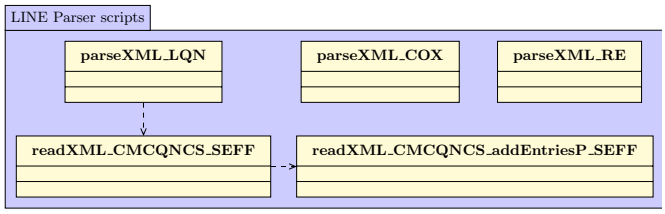

Figure 7: LINE Server Scripts

LINE **Main Solver** The scripts in this component are depicted in Figure 8.

- `CMCQN_CS_analysis_SEFF`: this is the main script of the transformation and operates on a `CMCQNCS` object as PM. It calls the Fluid Solver to obtain the stationary distribution of the PM, which it uses to obtain the mean performance measures. It also calls the Fluid Solver to obtain the response time distributions.
- `CMCQN_CS_RE_analysis_SEFF`: this is a similar script as the one above, but operates on a `CMCQNCSRE` object as PM, thus considering an extended PM with RE.
- `CMCQN_CS_Cox_analysis_SEFF`: this is a similar script as the first one, but operates on a `CMCQNCSCox` object as PM, thus considering an extended PM with Coxian distributions.
- `CMCQN_CS_Cox_RE_analysis_SEFF`: this is a similar script as the first one, but operates on a `CMCQNCSRECox` object as PM, thus considering an extended PM with both RE and Coxian distributions.
- `CMCQN_CS_respTime_SEFF`: this script supports all the other scripts above by computing performance metrics for the intermediate LQN tasks, referred to as SEFF in Palladio.

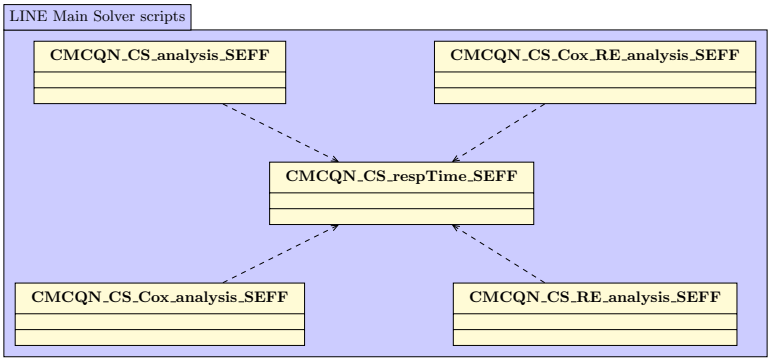

Figure 8: LINE Main Solver Scripts

LINE **Fluid Solver** The scripts in this component, depicted in Figure 9, can be divided in two groups: the first group, composed of the scripts on the left column in Figure 9, focuses on PM models without RE; the second group, composed of the scripts on the right column in Figure 9, focuses on PM models with RE. The scripts that solve PMs without RE are

- `CMCQN_CS_fluid_ps`: this script sets up the fluid model that represents the PM (`CMCQNCSCox` object), and solves it using MATLAB ODE solvers. Returns the stationary distribution of the PM.
- `CMCQN_CS_fluid_ps_RT`: this script sets up a fluid model to compute response time distributions.
- `CMCQN_CS_fluid_analysis`: this script defines the ODE system of the fluid model, which is used by the MATLAB ODE solvers.

The scripts that solve PMs with RE are defined similarly.

References

[1] D. Ardagna, M. Ciavotta, M. Shokrolahi, G. Gibilisco, G. Casale, and J. Pérez. MODAClouds Deliverable D5.4.1. Prediction and cost assessment tool - Proof of concept. http://www.modaclouds.eu/software/, 2013.

[2] G. Casale, J. F. Pérez, and W. Wang. QD-AMVA: Evaluating systems with queue-dependent service requirements. In *Proceedings of IFIP PERFORMANCE*, 2015.

[3] J. F. Pérez and G. Casale. Assessing SLA compliance from Palladio component models. In *Proceedings of the 2nd MICAS*, 2013.

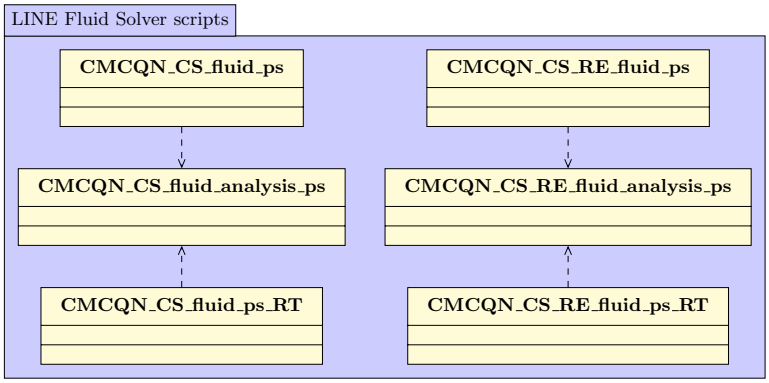

Figure 9: LINE Fluid Solver Scripts