Simulation in Produktion und Logistik
Entscheidungsunterstützung von der Planung bis zur Steuerung
Wilhelm Dangelmaier, Christoph Laroque & Alexander Klaas (Hrsg.)
Paderborn, HNI-Verlagsschriftenreihe 2013

Hybrid Models of Queuing Networks in Matlab: Experiences in Modeling, Simulation and Data Analysis

Hybride Modelle von Warteschlangennetzwerken in Matlab: Erfahrungen im Bereich Modellierung, Simulation und Datenanalyse.

Daniel Huber, Arizona State University, Tempe (USA), d@nielhuber.de

Abstract: This paper presents how Matlab was used in creating and experimenting with a hybrid model of a series of M/M/1 queues with limited queue capacity. In the model a discrete event model is coupled with a model based on hyperbolic conservation law. The modelling concept for building discrete event models in Matlab is discussed and parts of the hybrid model are presented in detail, especially how to couple the discrete event and continuous models. After that the use of user-written scripts and functions to efficiently conduct a large set of simulation replications by using parallel processing is presented.

1 Introduction

A hybrid model, consisting of a discrete event simulation (DES) and a continuous part was developed to improve existing methods of model simplification. The hybrid model is the simplification of the complex original model, which is a DES model of a flow shop system. In the most abstract form, these systems can be modelled as a series of M/M/1 queues. In a general DES simulation tool, this is modelled as a series of pairs of queues and servers with a source in front and a sink at the end. In basic queuing theory the queue in a M/M/1 system has unlimited capacity, which is not a realistic assumption for most real life flow shop applications. Thus the queues in the original model have a limited capacity. The original model is shown in Figure 1.

Figure 1: *Original Model*

A common method to simplify DES models like this is to identify the bottleneck server in the line, keep it untouched and aggregate all other queues and servers to one delay in front and after this bottleneck (Johnson et al. 2005, Huber and Dangelmaier 2009). Though the quality of this simplification method is relatively good, a research project was started to investigate if using a continuous model of the aggregated queues and servers to control the delay could further improve the quality of the results.

Since in the beginning of the project it was not clear which form of continuous model would be used, a development platform was needed that allowed easy coupling of DES with general continuous models, was inexpensively accessible and was a mature product.

The choice fell on Matlab (www.mathworks.com/products/matlab/): it has a DES extension 'SimEvents', which can use Matlab functionality; the Arizona State University, like many universities, has floating licences and it is a well maintained software tool with frequent updates and a huge user base.

In the following sections the author will give a short introduction into Matlab, a more detailed presentation of its discrete event-modelling environment SimEvents, a detailed view on parts of the developed hybrid model and a description of how Matlab can be utilized for efficient experimentation. The theoretical basis of the continuous model, the development and the performance of the hybrid model are not focused in this paper.

2 Modelling Concept

Matlab is a software tool for numerical computing focusing on matrix manipulation. It is also a 4th generation programming language. It allows plotting of functions and multidimensional numerical data, implementation of algorithms, symbolic computing and the graphical modelling, simulation and analysis of multidomain dynamic systems. The latter feature is provided by Simulink, which is tightly integrated into Matlab. The Simulink extension SimEvents is designed to model and simulate general discrete event systems. To give the reader an idea how modelling and simulation works in SimEvents and how Matlab and Simulink functionality can be accessed and integrated, the modelling concept shall be presented.

In general, as in most DES tools, entities are created in a source and terminated in a sink. Sources and sinks are building blocks of the model, as are servers, queues, random number generators, etc. Each of these material flow blocks has zero or more input and output channels for entities. Links are used to connect two channels. Besides these material flow channels there are channels for numerical information (input or output data) and channels for control signals. Links for information or control signals can be split, to deliver the information/signal to more than one recipient. These links can also be joined, creating a vector with the values of all joined links.

A typical example for a material flow block is the server, it has one input and output channel for entities. It has several pre-defined statistics, like number of departed entities or utilization. For every activated statistic, an information output channel is created. This channel can then be connected to a data storage block, a data visualization block or to a block that uses this information for control decisions. The

service time of the server can either be entered in a dialog or be acquired from a random number generator (RNG). If the latter is selected, an information input port is created to which a RNG-block is connected. In the RNG-block the desired random distribution can be defined.

To allow entity-based evaluations, there are blocks to write and read attributes to/from entities, like time stamps or identification numbers. Entities can be routed through a network of blocks by using switches and combiner-blocks. To stop the flow of entities there are gate-blocks, that open or close according to an input information or signal. In these blocks it can be defined whether they should listen to an information link or a control link. If an information link is chosen, it must be defined whether the gate changes its status when the information value increases, decreases or both. This method of listening to inputs is implemented in many building blocks for control and event or signal generation.

Function-call generators are blocks to generate control signals. They can be time-based (generate signals at specific times), entity-based (when an entity passed) or information-based (when information changes). The generated signals can now be used to execute function-call-subsystems. These subsystems have information input and output channels and an embedded Matlab function that transforms the inputs into outputs. The embedded Matlab function can make this transformation by itself or execute a function located in the Matlab folder, exterior to the SimEvents model file. Matlab functions are written in Matlab's own programming language, whose syntax is similar to C. A short example of a Matlab function is given in Algorithm 1.

Algorithm 1: *An example for a Matlab function*

```
function result = example(A, x)
%find lowest index of column in matrix A with a value >x
[row,column,v] = find(A>x);
tmp =  min(column);
%transpose col-vector and multiply element-wise with vector
result = transpose(A(:,tmp)) .* [1 2 3 4 5];
end
```

Functions in Matlab, as in general programming languages, take a set of input variables and produce a set of output variables. There are also scripts, which are a set of Matlab commands, but these do not have input or output variables and instead modify, create or visualize variables in the workspace. The Matlab-workspace consists of all variables the user creates and stores during a session. When using the term function, either a function that comes with Matlab or a user-written function can be meant.

In Matlab all variables are matrices of different dimension, having a specific matrix data-type. This has to be kept in mind when calling exterior Matlab functions from embedded functions. In Simulink and SimEvents variables must have a defined, unchanging dimension and they are of a different data-type than standard matrix data-type. Also some Matlab functions, like the function to generate gamma distributed random numbers, do not work in embedded functions without a special declaration. This may result in additional debugging time when functions are called in embedded functions that were developed independently of the SimEvents model.

3 Example: Hybrid Model of a Series of M/M/1 queues with limited queue capacity

To see a practical implementation of the modelling concept and how the DES can be coupled the continuous model, the hybrid model shall be presented.

The conceptual model is shown in Figure 2. The Source, Sink and Block-Timer can be seen as the environment for the actual hybrid model. When comparing the simplification to the original model, these three blocks are the same. The Block-Timer is used to simulate a bottleneck machine at the end of the line. Remember that the common method for simplification is to leave the bottleneck untouched and aggregate the line before and after to delays, and that the hybrid model is intended to enhance the quality of the delay.

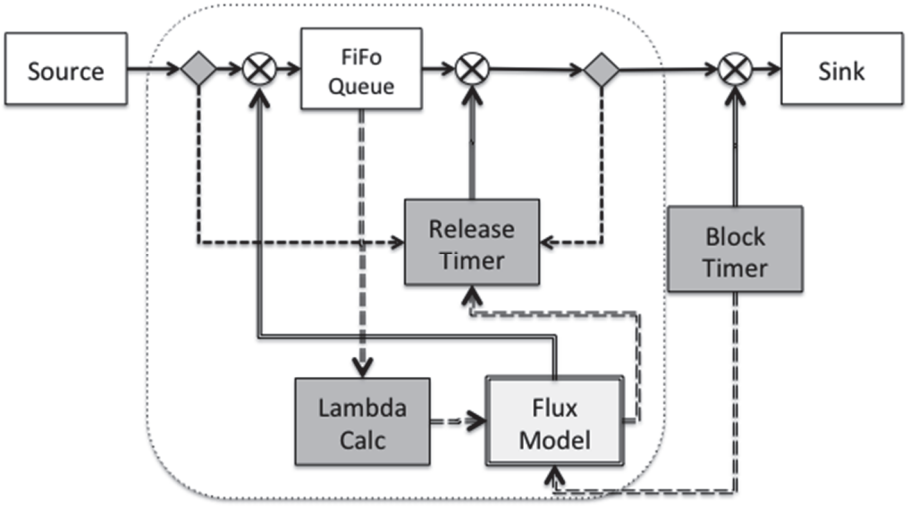

Figure 2: *The Concept Model*

The diamond shaped blocks are function-call generators, which send a signal to the Release-Timer. The generator in front of the queue is only used for the first entity entering an empty queue. The generator behind the queue signals the Release-Timer to generate a new inter-departure time and open the gate (circle with cross) at this time for one entity and let it move from the queue to the sink. Veeger et al. (2009) have shown that controlling a simple model by using state dependent inter-departure times can provide very good result quality. The Lambda-Calc block calculates the arrival rate of entities by using the number of arrivals in a specific time interval. The continuous model used here is a hyperbolic conservation law of the form $\frac{\partial \rho}{\partial t} + \frac{\partial F}{\partial x} = 0$, which describes the time evolution of the WIP-density ρ at the position in line x. F is the flux which is dependent on ρ and x. The flux at x=0 is the arrival rate λ and the flux at x=1 is the reciprocal value of the inter-departure time. A detailed description of this model can be found in (Armbruster et al. 2011). This continuous model, the Flux-Model, takes the arrival rate as input and generates a departure rate as output, which is used by the Release-Timer to generate a state dependent inter-

departure time. To be able to react on blockages of the Block-Timer the Flux-Model also receives information about the block status. To have a good approximation of the behaviour of the original system, the Flux-Model implements a function to block the entrance to the queue, dependent on the density at x=0.

The implementation of this conceptual model can be seen in Figure 3. 1a-b are the links to the source and sink, 2a-b are the function-call generators, 3a-b are the gates, 4 is the FiFo Queue, 5a-c is the Release-Timer, 6 is the Lambda-Calc, 7 is the Flux-Model and 8a-b are the links with the information about the blocking status.

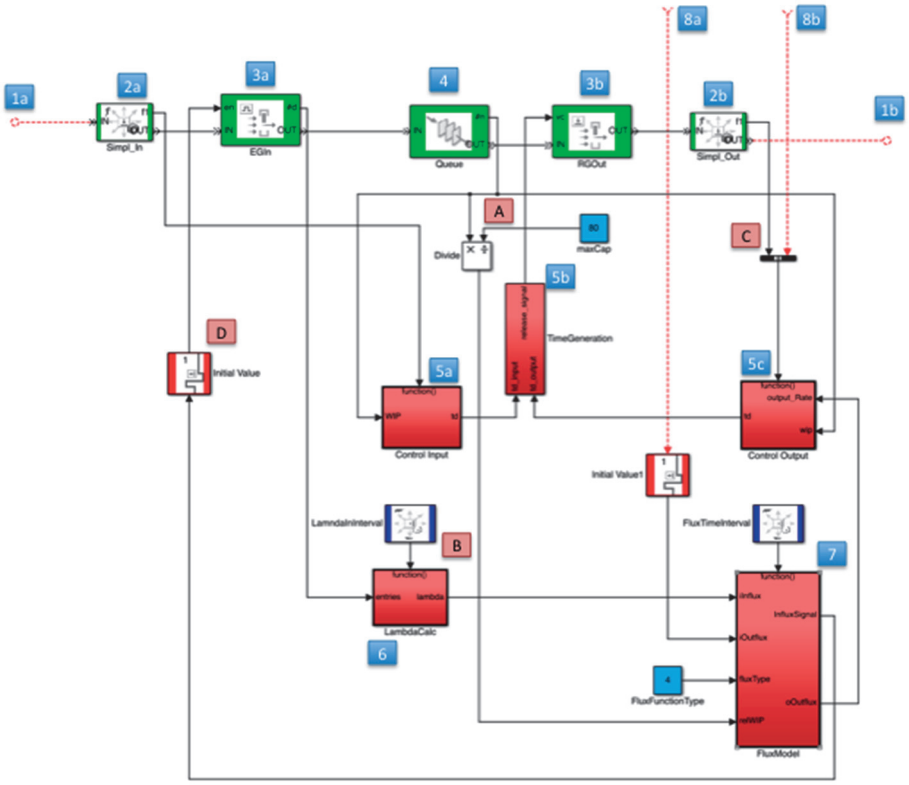

Figure 3: *Implementation of the Hybrid Control Loop*

The statistic 'Number of entities in block' in the queue is activated and at A the information is split to 5a, 5c and a Divide-block. This block calculates the relative fill-level of the queue by dividing the information of the queue with the capacity of it. This value cannot be accessed from the queue, thus a Constant-value-block is used to provide the queue capacity as output information. The relative fill level is then used as an input of the Flux-Model. At B, a time-based function-call generator triggers the Lambda-Calc block in a defined interval. At C, two control signals are joined, one signalling an entity has left the queue, the other one signalling the blockage has ended, both resulting in the generation of a new inter-departure time.

Since the output values of the Flux-Model are not defined before its first execution, the input to the gate must be set to initial value at D.

In Figure 4, the contents of the Flux-Model subsystem are shown. The function-block has 5 inputs and 3 outputs. The output oRho is used as input iRho in order to save the state of the flux model from one iteration to the next. However, since the output of a function is not defined before execution, a special memory-block has to be used to save the last output and before it can be used as an input.

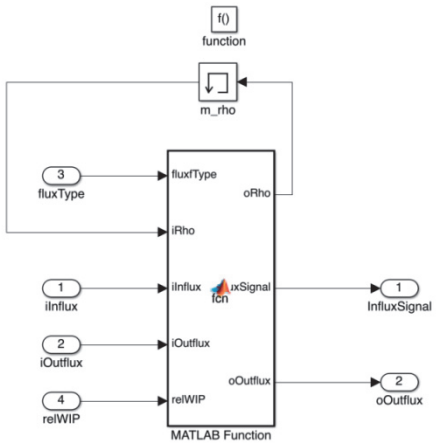

Figure 4: *Flux-Model Subsystem*

Algorithm 2 shows the content of the function-block in Figure 4. The partial differential equation of the flux model is solved using the iterative Lax-Friedrich method, implemented in the function 'Fluxmodel'. Each time the subsystem receives a function-call, Algorithm 2 is executed and with it a new iteration of the Lax-Friedrich. In each iteration, the time in the flux-model increases by a constant amount?, meaning that the function-call has to occur in a constant DES-time interval.

Algorithm 2: *Function of the Flux-Model Subsystem*

```
function [oRho, InfluxSignal, oOutflux]  = fcn(fluxfType,
iRho, iInflux, iOutflux, relWIP)
oRho = zeros(42,1);

%call the flux model
[tmp, oOutflux] = Fluxmodel(fluxfType, iRho, iInflux,
iOutflux);
oRho(:,:) = tmp(:,:);

%propablility function to close input gate
InfluxSignal = blockFunction(oRho(2));
end
```

The variable oRho in Algorithm 2 has to be defined specifically and tmp has to be used to receive the output of the function for two reasons. First, the returned vector of the function 'Fluxmodel' is not in a datatype usable in SimEvents (oOutflux is ok, since it is a single numerical value). Second, during compilation of the model, SimEvents requires that the dimension of every variable is fixed. The complier is unable to check if the dimension of a return of an external function is of fixed dimension. When assigning tmp to oRho the datatype is cast to the one of oRho. This example shows that embedded functions have to be written with special care and sometimes require seemingly unnecessary code.

The evaluation of the hybrid model is done by generating a timeline of the relative fill-level of the queue, the output rate measured at the sink and the cycle time between source and sink.

4 Experiment Execution

An important feature of simulation tools that allows efficient execution of many model scenarios is the ability to store input data external to the model file and to have an interface to access output data. In Matlab, the parameters of building blocks can be modified by a function. In Algorithm 3 the parameter 'period' of block 'FluxInterval' in Model 'HFM01' is changed to 0.4.

Algorithm 3: *Changing the parameters of building blocks in a function*

```
set_param('HFM01/FluxInterval', 'period',num2str(0.4));
copyfile('HFM01.slx', 'tmp_HFM01_1.slx');
```

After modification, the model file is copied to a temporary model-file. With this method implemented in a loop and reading data from some table in the Matlab workspace a set of model files can be generated that later can be simulated in a parallel or batch process. In Algorithm 4 a parallel execution is shown.

Algorithm 4: *Parallel execution of simulation replications*

```
parfor j = 1 : reps
%load model
load_system(modelname);
%shuffle RNG seeds
seed = mod(floor((j/reps) * now * 8640000),2^31-1);
se_randomizeseeds(modelname, 'GlobalSeed', seed);
s = rng('shuffle','twister');
%get output data
simout = sim(modelname);
wip = simout.get('wip');
wip_timevalues = [wip.time, wip.signals.values];
wipa(:,:,j) = wip_timevalues(:,1:2);
%close model
bdclose(modelname);
end;
```

By using the parfor-loop, instead of a for-loop, all j iterations of this loop are executed in parallel on the available Matlab-workers. When working on a single computer, for each CPU-core available a Matlab-worker can be started. More workers can be started over the network, if available.

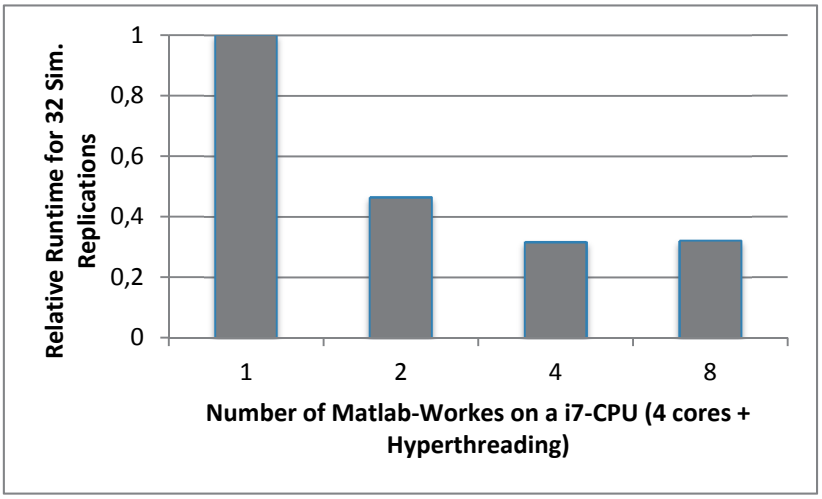

Figure 5: *Scaling of Runtime when Using Increasing Number of Workers*

In Figure 5 the scaling of the parallel execution of simulation replications is shown. The CPU used by the author has only 4 physical cores (8 with hyper-threading). The speed-up when using 2 and 4 workers is significant; using 8 workers did not improve the runtime. Since the used system has 8GB of memory and every worker allocates ca. 500-700 MB, the memory was the bottleneck and writing parts of the memory to disk increased the runtime.

In the parfor-loop the output data of each replication can be processed as necessary, but to generate data used outside the loop, data has to be written into a 'global' matrix (wipa in Algorithm 4). By using this matrix, statistical figures over all replications can be calculated after finishing the parfor-loop.

By implementing all of the above mentioned methods, calculating the mean WIP and throughput rate as timelines (matrices W ant T) of 32 replications for a certain scenario can be started like this:

Algorithm 5: *Starting a set of simulation replications*

```
[W, T] = start_Exp('HMF01', runtime, scenario_id, util, 32);
```

5 Data Analysis

Matlab comes with many potent statistical functions, making it easy to calculate means, standard deviations, minima, maxima, histograms regressions, etc. When opening matrices (aka. tables) in the workspace, several 1-click plots are available to visualize data (bar, scatter, x-y, …). When creating 3-dimensional plots, it is possible to rotate the surface and zoom in and out. When plotting data, Matlab also offers a tool to fit a function (exponential, polynomial, etc.) to the data. Since all plots and statistical analyses can be created by using the Matlab-language, this can be automated, which is especially helpful when working with many scenarios or needing to repeat slightly modified experiments. All variables in the workspace can be saved or easily exported to a spreadsheet by using copy-and-paste of selected cells.

6 Conclusion

Matlab/Simulink with the extension SimEvents was valued a very flexible modelling, simulation and analysis tool. The test platform (Version 2012b for MacOS) was very stable, although there were some usability problems with Simulink. The author developed most functions of the flux model independent of the SimEvents model; so integrating them into embedded functions was frustrating, since some 'beautiful' code had to be changed. When implementing the parallel execution of simulation replications, several problems were encountered, but the user forum offered solutions for all of them, either similar problems were already solved or questions from the author were answered within 24 hours.

Especially positive were the capabilities to utilize parallel processing, automatize the entire experimentation and analysis process, couple a DES with a general partial differential equations model and readily visualize data. The author does not want to imply that he used the most efficient functions available in Matlab for this problem, many powerful and helpful functions may remain to be discovered, but the author would like the reader to consider using Matlab for a new project.

Acknowledgement

This research was financially supported by the Deutsche Forschungsgemeinschaft grant HU1912/2-1.

References

Armbruster, D; Göttlich, S.; Herty, M.: A Scalar Conservation Law with Discontinuous Flux for Supply Chains with Finite Buffers, SIAM Journal on Applied Mathematics, 2011, 71, 1070-1087.

Huber, D.; Dangelmaier, W.: Controlled Simplification of Material Flow Simulation Models. In: Rossetti, M. D.; Hill, R. R.; Johansson, B.; Dunkin, A.; Ingalls, R. G. (Ed.), Proceedings of the 2009 Winter Simulation Conference, Institute of Electrical and Electronics Engineers, Inc., 2009, 839-850.

Johnson, R. T.; Fowler, J. W.; Mackulak, G. T.: A Discrete Event Simulation Model Simplification Technique. In: Kuhl, M. E.; Steiger, N. M.; Armstrong, F. B.;

Joines, J. A. (Ed.), Proceedings of the 2005 Winter Simulation Conference, Institute of Electrical and Electronics Engineers, Inc., 2005, 2172-2176.

Veeger, C.; Etman, P.; Rooda, J.; van Herk, J.: Cycle Time Distributions of Semiconductor Workstations using Aggregate Modeling. In: Rossetti, M. D.; Hill, R. R.; Johansson, B.; Dunkin, A. & Ingalls, R. G. (Ed.), Proceedings of the 2009 Winter Simulation Conference, Institute of Electrical and Electronics Engineers, Inc., 2009, 1610-1621.