Probabilistic state space models – A theoretical framework with practical relevance

Peter Junglas[1]*

[1]Department of Engineering "Dr. Jürgen Ulderup", PHWT Vechta/Diepholz, Schlesierstr. 13a, 49356 Diepholz, Germany; *peter@peter-junglas.de

Abstract. Corresponding to the modeling purpose discrete models can be defined using very different approaches: For a precise description and thorough analysis one of the many different mathematical descriptions can be applied, while a working practitioner often will describe a model within a concrete simulation environment. To demonstrate that mathematical models are useful for practical purposes as well, we will present a simple state-space model for a stochastic discrete system. By means of a concrete example we will show, how the use of this model makes the practical modeling process much easier and leads to a more concise concrete implementation.

Introduction

For the description of discrete systems a large number of different mathematical models can be applied, ranging from the simple finite state machine [1] to the complex PDEVS formulation [2]. If one includes stochastic processes, the models get more complicated, well-known examples being the non-deterministic finite automaton [1] and the generalized semi-Markov process [3].

Using mathematical models brings considerable benefits: First of all it allows a complete and precise specification of a model. Secondly the whole machinery of mathematics can be used for the analysis of important properties of the models like the reachability of states, reducibility of the state space or the existence of equilibrium configurations [3].

These points are of a more theoretical nature, but there is a third advantage, often overlooked, which is important for the practitioner: A mathematical model can simplify the actual modeling and implementation in a concrete simulation environment considerably.

To illustrate this point we will present a pedagogical example and implement it in Simulink in a straighforward manner. This turns out to be more difficult than expected and leads to a structurally complicated solution. Next we will introduce a simple mathematical model using a probabilistic state space representation and show, how the reformulation of the example problem using this model leads to a much simpler and clearer implementation. Finally we will turn to the soundness of the basis by having a quick look at the mathematical status of some important simulation programs.

1 A pedagogical example

The model used in the following describes class sizes of a three-year third level school and the number of prospected school graduates. It is formulated in three steps of increasing detail, based on an example from [4] and extended here. The basic outputs are the class sizes $x_i(k)$, $i = 1\dots3$, at the beginning of year k and the number of graduates $x_g(k)$.

As a first step we assume given constant rates R_i of students, who have to repeat year i, and D_i of dropouts during year i. The model is then defined by

$$\begin{aligned}
x_1(k+1) &= x_{in}(k) + R_1 x_1(k) && \text{(1a)}\\
x_2(k+1) &= (1 - R_1 - D_1)x_1(k) + R_2 x_2(k) && \text{(1b)}\\
x_3(k+1) &= (1 - R_2 - D_2)x_2(k) + R_3 x_3(k) && \text{(1c)}\\
x_g(k+1) &= (1 - R_3 - D_3)x_3(k) && \text{(1d)}
\end{aligned}$$

The resulting class sizes are non-integer, representing "mean values" over several years.

In the next step the mean values are replaced by integer random numbers following a binomial distribution B(n,p). In the defining equations the number of repeaters and dropouts are replaced according to

$$\begin{aligned}
R_i x_i(k) &\rightarrow \xi_{R,i} \sim B(x_i(k), R_i)\\
D_i x_i(k) &\rightarrow \xi_{D,i} \sim B(x_i(k), D_i)
\end{aligned}$$

leading to integer-valued class sizes.

Finally we add a rate M_i ($i = 2,3$) of pupils who return to the previous class at midterm – a realistic possibility at German schools. The concrete numbers are again drawn from a Binomial distribution: $\xi_{M,i} \sim B(x_i(k), M_i)$. This seemingly simple extension has drastic consequences for the modeling, because it leads to changes of the class sizes at half-integer years.

2 Straightforward implementation

The implementation of the simplest model in Simulink can be done easily. One starts by introducing a generic component for a class with one input i for new pupils and three outputs: the class size c at the end of the term, the number t of pupils transferred to the next class and (for completeness) the number d of pupils that have dropped out during the current year. It is built with one `UnitDelay` block storing the class size $x(k)$ and the usual arithmetic blocks to implement the following relations derived from the model equations (1)

$$x(k+1) = i(k) + Rx(k) \tag{2a}$$
$$c(k) = i(k) + Rx(k) \tag{2b}$$
$$t(k) = (1 - R - D)x(k) \tag{2c}$$
$$d(k) = Dx(k) \tag{2d}$$

The complete model is then just a chain of three class components (cf. Fig. 1).

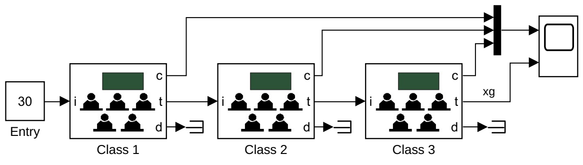

Figure 1: Simple school model

To replace the fixed rates with binomial random variables, one creates a component that outputs a random value $\xi \sim B(n, p)$, where p is given as parameter, while n comes from an input. This can be done with a simple Matlab function. The corresponding class is shown in Fig. 2.

The real challenge is of course the inclusion of the midterm downgrading. On the upper level this is simple enough: Just add another output m to the class component that gives the number of downgraders and route it

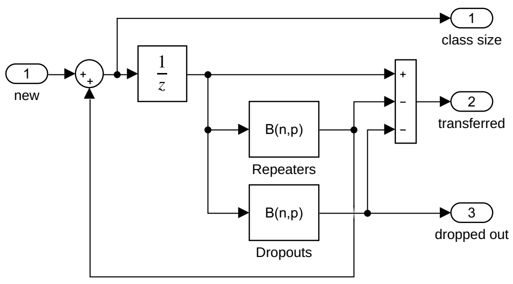

Figure 2: Class component with random values

back to the input of the lower class (cf. Fig. 3). Due to the different timing of the values, m cannot simply be added to the normal input. But a `TimeSwitch` that uses a standard `Switch` component to route its two inputs according to the time (integer or half-integer) solves this problem.

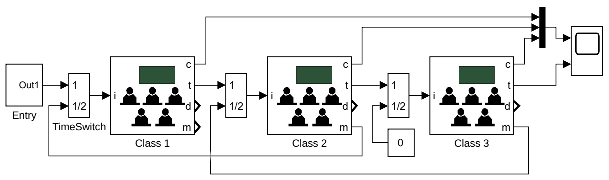

Figure 3: Complete school model

The implementation of the new class component starts by adding a binomial block for the downgraders, whose output is routed back internally to another `Time-Switch`. But the interesting question is of course, how one realises the different sample times: The class size changes every half year, the numbers of repeaters and dropouts at the beginning of a year and the number of downgraders at midterm.

Apparently Simulink offers an easy solution: Setting the `SampleTime` parameter of the `UnitDelay` to 0.5 and adding two `Rate Transition` blocks, which convert the signal to sample times `[1,0]` and `[1, 0.5]`, should do the trick. The corresponding class component is shown in Fig. 4, where the colors denote the different sample times.

But running the model we get the error message

> Determinism of data transfer between 'school3a/Class 1/Unit Delay' and 'school3a/Class 1/Sum3' cannot be ensured because either or both blocks have non-zero sample time offset. You can resolve this by using a rate transition block whose parameter 'Ensure deterministic data transfer' is unchecked

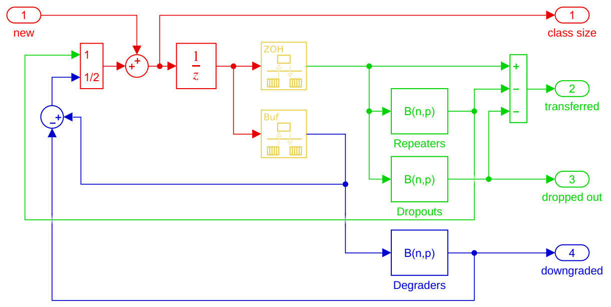

Figure 4: Complete class model

The suggested resolution actually leads to a working model: After unchecking the option `Ensure deterministic data transfer` of the second `Rate Transition`, the model runs without problems, and the results are as expected. But do we really know what is going on here?

Changing the sample rate is generally a non-trivial business, but a look at the relevant Simulink documentation [5] shows that there are more troubles looming around than one probably thought of, e. g. problems with timing when using multicore cpus. Without a precise understanding of the `Rate Transition` block, one cannot be sure that the `Class` component still works, when it becomes part of a very complex model – as real world components usually do.

The incremental modeling approach has lead into murky ground, since the simple structure of the model got lost on the way. Instead of relying on only half-understood remedies, we will therefore start afresh, this time with a solid foundation in the form of an underlying mathematical model.

3 A probabilistic state space model

We start with the well-known state space description

$$\begin{aligned} z(k+1) &= G(z(k), v(k)) &\quad (3a)\\ w(k) &= H(z(k), v(k)) &\quad (3b)\end{aligned}$$

The integer k is the number of the time step, while z denotes the internal state, v the external input and w the output of the model, all possibly being vectors. This model can be easily implemented in Simulink by using the generic model shown in Fig. 5 and providing components for the functions $G(z, v)$ and $H(z, v)$.

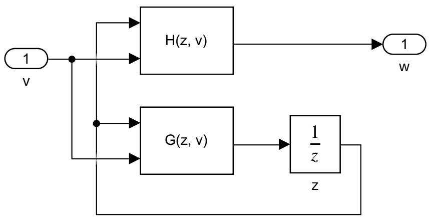

Figure 5: Generic state space model

For the stochastic examples we have to enlarge the state-space description. A common approach is the inclusion of an additional disturbance term $d(k)$ [6]:

$$\begin{aligned} z(k+1) &= G(z(k), v(k), d(k))\\ w(k) &= H(z(k), v(k), d(k))\end{aligned}$$

In our case the disturbance is stochastic and its distribution depends on the state z. Therefore we define a random vector $\xi(z)$ and the *probabilistic state-space description*

$$\begin{aligned} z(k+1) &= G(z(k), v(k), \xi(z(k))) &\quad (4a)\\ w(k) &= H(z(k), v(k), \xi(z(k))) &\quad (4b)\end{aligned}$$

Again a generic Simulink implementation can be easily given, cf. Fig. 6. The additional component `Random` computes the value of the random vector $\xi(z)$ using the current state vector $z(k)$.

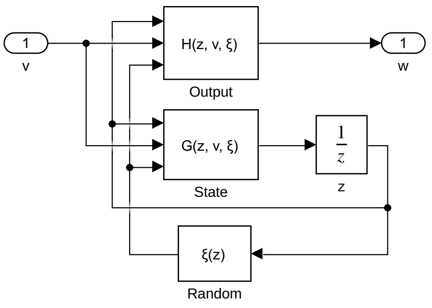

Figure 6: Generic probabilistic state space model

4 Theory-based implementation

We will now reimplement the example models using the mathematical descriptions and corresponding generic models defined above.

To reformulate the first model using eq. (3) we identify the state variable z with the class size x. Then we rewrite the defining equations (2) by setting $v \equiv i$, $w \equiv (c,t,d)'$ and get

$$\begin{aligned} G(z,v) &= v + Rz \\ H(z,v) &= \begin{pmatrix} v + Rz \\ (1-R-D)z \\ Dz \end{pmatrix} \end{aligned}$$

The Simulink implementation of G and H is completely trivial, but so was the version we started with. Therefore we didn't gain much here expect for making contact to standard mathematical formulations.

For the implementation of the second model we now use the probabilistic state-space description eq. (4) and write

$$\xi(z) \equiv \begin{pmatrix} \xi_R(z) \\ \xi_D(z) \end{pmatrix} \sim \begin{pmatrix} B(z,R) \\ B(z,D) \end{pmatrix}$$

$$\begin{aligned} G(z,v,\xi) &= v + \xi_R \\ H(z,v,\xi) &= \begin{pmatrix} v + \xi_R \\ z - \xi_R - \xi_D \\ \xi_D \end{pmatrix} \end{aligned}$$

The component `Random` simply combines the outputs of two `Binomial` blocks into a vector.

The final example model is time-dependent, since its behaviour changes between full and half years. The standard trick here is to include the time as a component of the state vector. In our case we only need to know whether we are at full or half term. Therefore we enlarge the state vector writing

$$z = \begin{pmatrix} x \\ s \end{pmatrix}, \quad z(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

where x is again the class size and the variable s is 0 or 1 at full resp. half term. Its state equation then simply is

$$s(k+1) = 1 - s(k).$$

Now we have to define the functions $\xi(z)$, $G(z,v,\xi)$ and $H(z,v,\xi)$, especially their behaviour at s = 0 and s = 1. First we define the random vector $\xi = (\xi_R, \xi_D, \xi_M)'$. The first two components describe the numbers of repeaters and dropouts, so they should be 0 at midterm. Correspondingly the number ξ_M of midterm downgraders should be 0 at the beginning of a term. Thus we have

$$\begin{aligned} \xi_R(z) &\sim \begin{cases} B(x,R) & | \quad s=0 \\ 0 & | \quad s=1 \end{cases} \\ \xi_D(z) &\sim \begin{cases} B(x,D) & | \quad s=0 \\ 0 & | \quad s=1 \end{cases} \\ \xi_M(z) &\sim \begin{cases} 0 & | \quad s=0 \\ B(x,M) & | \quad s=1 \end{cases} \end{aligned}$$

Similarly we at arrive at the state equation

$$\begin{aligned} G_1(z,v,\xi) &= \begin{cases} v + \xi_R & | \quad s=0 \\ v + x - \xi_M & | \quad s=1 \end{cases} \\ G_2(z,v,\xi) &= 1 - s \end{aligned}$$

Finally we enlarge the output vector to include the downgraders writing $w \equiv (c,t,d,m)'$ and get the output function

$$\begin{aligned} H_1(z,v,\xi) &= \begin{cases} v + \xi_R & | \quad s=0 \\ v + x - \xi_M & | \quad s=1 \end{cases} \\ H_2(z,v,\xi) &= \begin{cases} x - \xi_R - \xi_D & | \quad s=0 \\ 0 & | \quad s=1 \end{cases} \\ H_3(z,v,\xi) &= \xi_D \\ H_4(z,v,\xi) &= \xi_M \end{aligned}$$

This formal approach guarantees a completely precise problem specification, though it may seem a bit tedious. On the upside the Simulink implementation is now merely a matter of routine: We use the generic model from Fig. 6 adding a `Demux` block to get the single output ports. Next we implement the equations for `Random` (cf. Fig. 7), `State` (cf. Fig. 8) and `Output` (along the same lines).

The auxiliary component `PhaseSwitch` (cf. Fig. 9) helps to implement the case switches. Instead of a clock (as in the first implementation) it now simply uses the value of the state variable s.

The complete `school` model looks almost like before (cf. Fig. 3), but the `TimeSwitch` components at the class inputs are replaced by simple summation blocks. This is possible now, since all signals have the same sample time 0.5 and have meaningful values at all times.

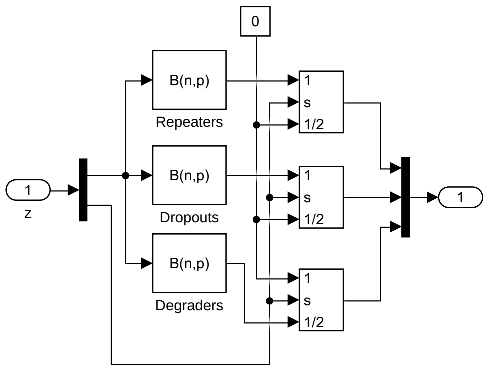

Figure 7: Implementation of the `Random` function

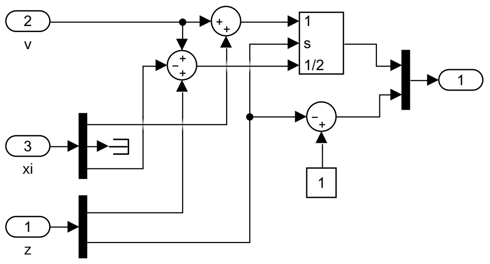

Figure 8: Implementation of the `State` function

5 Mathematical foundation of simulation tools

Even if a model is based on a sound mathematical foundation, its simulation results may be not, namely when it is implemented using a simulation environment that is itself not precisely defined. The Simulink blocks that have been used in the above model, are very simple and can be easily described mathematically. But already the example of the `Rate Transition` block has shown that this may not always be the case. Therefore lets finally have a quick look at the mathematical status of some important tools.

While the model descriptions of continuous systems often boil down to differential or differential-algebraic equations, there is no generally accepted mathematical formulation of hybrid systems containing a few discrete events or even complex state charts [7]. So even the framework, in which to formulate a mathematical model of a simulation tool, is still under construction.

For Stateflow [8], a Simulink add-on to model hierarchical state machines and flowchart diagrams, the situation has been described as follows:

> However, Stateflow lacks any formal definition. The semantics of a program is given by the result of its simulation within the Mathworks tools. This absence of formal definition is a big obstacle to static analysis, verification, or automatic test-cases generation of Stateflow designs.[9]

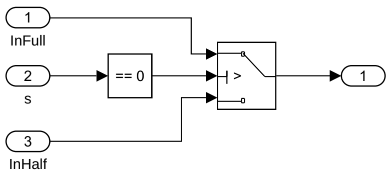

Figure 9: Implementation of `PhaseSwitch`

Therefore the authors proceed to define operational [10] and denotational [9] semantics for Stateflow in a formal way. One immediate result is the insight that Stateflow is structurally different from other, seemingly very similar formulations of statechart diagrams. A different way to formalize a hybrid Simulink/Stateflow model has been proposed in [11] with applications to the formal verification of such complex systems as the guidance control of a lunar lander or the control system of a high-speed train.

Another simulation program in the Matlab/Simulink tool chain is SimEvents [12] that implements the transaction-based modeling paradigm for discrete event systems. Earlier releases had some serious deficiencies [13], therefore Mathworks came up with a complete redesign with version 5. A very interesting feature is that the design is based on a unifying theoretical description [14]. Unfortunately, Mathwork has chosen a new design instead of relying on the well-known PDEVS formalism [2] and doesn't provide these internal specifications in general.

The discrete-events simulation program Arena [15] from Rockwell Automation implements the process-based modeling approach. It is based internally on the simulation language SIMAN [16] and outputs a SIMAN program corresponding to the graphically constructed model. This can be useful for verification purposes, since the specification of the lower-level SIMAN constructs is much easier to understand than the complex blocks used on the upper level in Arena. And one can go down further: In [17] an essential part of the SIMAN language has been implemented using the PDEVS formalism, thereby providing a sound mathematical formulation of important parts of Arena.

6 Conclusions

Using a clear mathematical description of our example problem we arrived at a precise specification that could be implemented in Simulink in a completely routine way and is structurally simpler than the previous “straightforward implementation”. But this idea only works, if the underlying simulation tools are grounded on sound mathematical models themselves. Though some efforts have been made in this direction, a lot needs to be done to reach a satisfying, well defined environment for our models.

The practitioner often works in the context of a given very large model that changes in the course of further development. He can’t be an expert of every intricacy of the complex simulation tool he works with, and very often has not enough time to go to the bottom of every problem. This can lead to ad-hoc implementations that are potentially dangerous in the highly dynamic larger context. The only way out is to use a precise mathematical model - for the own problem as well as for the relevant features of the simulation tool.

References

[1] Lunze J. *Ereignisdiskrete Systeme*. Berlin: de Gruyter, 3rd ed. 2017.

[2] Zeigler BP, Praehofer H, Kim TG. *Theory of Modeling and Simulation*. San Diego: Academic Press, 2nd ed. 2000.

[3] Cassandras CG, Lafortune S. *Introduction to Discrete Event Systems*. New York: Springer, 2nd ed. 2008.

[4] Junglas P. *Praxis der Simulationstechnik*. Haan-Gruiten: Verlag Europa-Lehrmittel. 2014.

[5] The Mathworks. *Handle Rate Transitions (Simulink Coder)*. `https://de.mathworks.com/help/releases/R2018a/rtw/ug/handle-rate-transitions.html`.

[6] Ljung L, Glad T. *Modeling and Identification of Dynamic Systems*. Lund: Studentlitteratur AB. 2016.

[7] Breitenecker F, Zauner G, Popper N, Judex F, Troch I. Structure of Simulators for Hybrid Systems - Development and New Concept of External and Internal State Events. *SNE Simulation News Europe*. 2007; 17(2):39–48.

[8] The MathWorks. *Stateflow: Model and simulate decision logic using state machines and flow charts*. `http://www.mathworks.com/products/stateflow/`.

[9] Hamon G. A denotational semantics for Stateflow. In: *Proceedings of the 5th ACM international conference on Embedded software*. ACM. 2005; pp. 164–172.

[10] Hamon G, Rushby J. An operational semantics for Stateflow. *International Journal on Software Tools for Technology Transfer*. 2007;9(5-6):447–456.

[11] Zhan N, Wang S, Zhao H. *Formal Verification of Simulink/Stateflow Diagrams: A Deductive Approach*. New York: Springer. 2017.

[12] Clune MI, Mosterman PJ, Cassandras CG. Discrete Event and Hybrid System Simulation with SimEvents. In: *8th International Workshop on Discrete Event Systems*. Ann Arbor. 2006; p. 386–387.

[13] Austermann L, Junglas P, Schmidt J, Tiekmann C. Conceptional problems of transaction-based modeling and its implementation in SimEvents 4.4. *SNE Simulation News Europe*. 2017;27(3):137–142.

[14] Li W, Mani R, Mosterman PJ. Extensible discrete-event simulation framework in SimEvents. In: *Proc. 2016 Winter Simulation Conference*. Arlington: IEEE. 2016; pp. 943–954.

[15] W David Kelton NBZ Randall P Sadowski. *Simulation with Arena*. New York: McGraw-Hill, 6th ed. 2015.

[16] Pegden CD, Shannon RE, Sadowski RP. *Introduction to Simulation using SIMAN*. New York: McGraw-Hill, 2nd ed. 1995.

[17] Sanz V. Hybrid System Modeling using the Parallel DEVS Formalism and the Modelica Language. Ph.D. thesis, UNED Madrid, E.T.S.I. Informatica. 2010.