Languages for modeling the RED active queue management algorithms: Modelica vs. Julia

Anna Maria Yu. Apreutesey[a], Anna V. Korolkova[a] and Dmitry S. Kulyabov[a,b]

[a]*Department of Applied Probability and Informatics, Peoples' Friendship University of Russia (RUDN University), 6, Miklukho-Maklaya St., Moscow, 117198, Russia*

[b]*Laboratory of Information Technologies, Joint Institute for Nuclear Research, 6, Joliot-Curie St., Dubna, Moscow region, 141980, Russia*

Abstract

This work is devoted to the study of the capabilities of the Modelica and Julia programming languages for the implementation of a continuously discrete paradigm in modeling hybrid systems that contain both continuous and discrete aspects of behavior. A system consisting of an incoming stream that is processed according to the Transmission Control Protocol (TCP) and a router that processes traffic using the Random Early Detection (RED) algorithm acts as a simulated threshold system.

Keywords

active queue management, simulation, Modelica, Julia, Random Early Detection

1. Introduction

The possible methods for investigating complex systems are the construction of a discrete event model, the construction of a continuous model, and hybrid modeling [1]. Such hybrid systems combine the operation of continuous and discrete elements, for example, systems with a discrete control device and a control object with a continuous nature of operation [2]. The system for the further research is the hybrid model of the interaction of the data transmission process via the Transmission Control Protocol (TCP) and the Random Early Detection (RED) algorithm as an algorithm of regulating the flow state. When modeling TCP-like traffic, the liquid (continuous) approach can be used, but it is especially important to consider discrete transitions between TCP states and a discrete packet drop function in RED-type algorithms. It is the hybrid approach that reflects these important features of the simulated system.

Workshop on information technology and scientific computing in the framework of the X International Conference Information and Telecommunication Technologies and Mathematical Modeling of High-Tech Systems (ITTMM-2020), Moscow, Russian, April 13-17, 2020

1032193049@pfur.ru (A. M. Yu. Apreutesey); korolkova-av@rudn.ru (A. V. Korolkova); kulyabov-ds@rudn.ru (D. S. Kulyabov)
https://yamadharma.github.io/ (D. S. Kulyabov)
0000-0001-6074-0201 (A. M. Yu. Apreutesey); 0000-0001-7141-7610 (A. V. Korolkova); 0000-0002-0877-7063 (D. S. Kulyabov)

1.1. The structure of the article

The structure of the article is as following. The 2 section describes the basic principles and the mathematical model of the operation of the congestion control mechanism in the TCP protocol with the RED queue management algorithm. The 3 section provides general information about the physical modeling language Modelica which implements, among other things, a hybrid paradigm, and this section also presents the implementation of the algorithm in the Modelica language. The 4 section provides information on the Julia programming language and the DifferentialEquations package for solving various differential equations. Also in this section some features of the Julia language are presented in the context of hybrid modeling by using the example of the RED module implementation for active traffic management.

1.2. Notations and conventions

In the work we will use the following variables and notation. The main parameters of the system are follows:

- $w(t)$ is the average TCP window size (in packets);
- $q(t)$ is the average queue length (in packets);
- $\hat{q}(t)$ is the exponentially weighted moving average of the queue length;
- $T(q,t)$ is the Round Trip Time (RTT, sec);
- $N(t)$ is the number of TCP sessions;
- C corresponds to the speed of procession of packets in the queue.

2. Random Early Detection as an active queue managment algorithm

The authors have repeatedly described the research problems, the features of the phenomenon under study, the construction of a mathematical model [3, 4].

In data transmission networks, there is a global synchronization, during which TCP sources send packets synchronously and also stop transmission synchronously. Such a stable self-oscillating mode of the system operation affects the quality of network service, throughput and latency negatively. The use of Active Queue Management algorithms, like RED [5], to control traffic reduces the probability of global synchronization, but it does not completely eliminate it. For some values of the initial parameters in the router settings, self-oscillations of the main parameters occur in the system.

An active queue management policy with a RED type algorithm is used to control and prevent congestions in the queues of routers. Algorithms for managing the state of traffic can be represented as control modules in network equipment. The advantage of this algorithm is its efficiency and relatively simple implementation on network equipment.

Let us describe a model for transmitting TCP-like traffic in which the dynamic flow rate is regulated by an RED type algorithm. The model consists of two elements - the packet-generating TCP source and the receiver, which is the queue of the router. It processes incoming packets in accordance with the control algorithm and notifies the source that the packets have reached their destination. The source and receiver interact through an intermediate link in accordance with the control algorithm, i.e. the value of the average queue length affects the source parameters, in particular, the size of the TCP congestion window.

A mathematical model of the interaction of an incoming TCP stream and a router that processes traffic using a RED-type control algorithm is the autonomous system of three differential equations [6, 7, 8, 9]:

$$\dot{w}(t) = \frac{1}{T(t)}\vartheta(w_{\max} - w) - \frac{w(t)}{2}\frac{w(t - T(t))}{T(t - T(t))}p(t - T(t)), \tag{1}$$

$$\dot{q}(t) = \begin{cases} (1 - p(t))\dfrac{N(t)w(t)}{T(t)} - C, & q(t) > 0, \\ \max\left[(1 - p(t))\dfrac{N(t)w(t)}{T(t)} - C, 0\right], & q(t) = 0, \end{cases} \tag{2}$$

$$\dot{\hat{q}}(t) = -w_q C\hat{q}(t) + w_q C q(t), \tag{3}$$

The (1) equation describes the dynamic change of the average TCP window size. The $1/T(q, t)$ element reflects the slow start TCP state, during which the window size increases per RTT. The $\vartheta(w_{\max} - w)$ component, which is a Heaviside function, limits the growth of the TCP window. The $(1 - p(t))w(t)N(t)/T(q, t)$ component in the (2) equation reflects the increase in the queue length when packets arrive which corresponds to the average packet arrival rate.

The (3) equation reflects the change of the $\hat{q}(t)$ — exponentially weighted moving average of the queue length function operating as a low-pass filter. It is introduced for some smoothing outliers of the queue length. As soon as the $\hat{q}(t)$ value exceeds some predetermined threshold value, the router finds out that the system has started overloading and reports this to the source.

The RED algorithm is controlled by the $p(\hat{q})$ — piecewise probabilistic packet drop function which can be written as a system of nonlinear equations.

$$p(\hat{q}) = \begin{cases} 0, & 0 \leqslant \hat{q}(t) < q_{\min}, \\ \dfrac{\hat{q}(t) - q_{\min}}{q_{\max} - q_{\min}} p_{\max}, & q_{\min} \leqslant \hat{q}(t) \leqslant q_{\max}, \\ 1, & \hat{q}(t) > q_{\max}. \end{cases} \tag{4}$$

This function depends on the threshold values of the queue size $q_{\min}$ and $q_{\max}$, as well as the $p_{\max}$ parameter that sets the part of packets that will be discarded if $\hat{q}$ reaches the maximum value.

3. Modeling in the Modelica language

The Modelica language was developed by the non-profit organization Modelica which also develops a freely distributed library based on it. This object-oriented language of physical modeling is used to solve a wide range of problems [10, 11]. Modelica is well suited for component-oriented modeling of complex systems consisting of various physical components, also having control components and elements oriented to individual processes. Let us demonstrate the application of this language to hybrid modeling of communication network algorithms [2].

The basis of the language is represented by classes that can be inherited. In Modelica classes contain methods and fields that can have types of variability such as constant, parameter and variable. In addition to methods and fields, the class contains functions and equations which are defined in the `equation` section. One of the mandatory requirements of the Modelica program is the coincident number of variables and equations.

Let us declare the variables and parameters used as initial ones.

```
Real p(start = 0.0) ”Drop probability”;
Real w(min = 1.0, max = wmax, start = 1.0, fixed = true) ”TCP window”;
Real q(max = R, start = 0.0, fixed = true) ”Instant queue length”;
Real q_avg(start = 0.0) ”EWMS queue length”;

parameter Real N(start = 60.0) ”Number of TCP sessions”;
parameter Real c(start = 10.0) ”Service intensity, Mbps”;
parameter Real packet_size(start = 500.0) ”Package size, bit”;
parameter Real T(start = 0.05) ”RTT”;
parameter Real thmin(start = 0.25) ”Normalized lower threshold”;
parameter Real thmax(start = 0.5) ”Normalized upper threshold”;
parameter Real R(start = 300.0) ”Queue size”;
parameter Real wq(start = 0.0004) EWMS parameter”;
parameter Real pmax(start = 0.1) ”Maximum drop probability”;
parameter Real wmax(start = 32.0) ”Max window size”;
parameter Real C = 125000.0 * c / packet_size ”Service intensity, packets”;
```

The congestion control algorithm in the queues of the RED router in Modelica is implemented as a `Red` class [12]. A discrete drop function is easily set in the main class using the standard `if` operator:

```
class Red
equation
p = if q_avg < thmin * R
    then 0.0
else if q_avg > thmax * R
    then 1.0
else (q_avg / R - thmin) * pmax / (thmax - thmin);
```

In this implementation, the discrete element of the system interacts well with continuous elements, the behaviour of which is also implemented in the main class of the program using three equations:

```
equation
der(w) = wAdd(w, wmax, T) + (-0.5) * w * delay(w, T, T) * delay(p, T, T) / delay(T, T, T);
der(q) = qAdd(pre(q), w, T, C, N, R);
der(q_avg) = wq * C * (q - q_avg);
```

In Modelica the der operator defines a time derivative. The delay is realized very simply with the help of the delay operator, which makes it possible to work with the delay of both continuous and discrete system elements.

The wAdd function limits the size of the TCP window:

```
function wAdd
input Real wIn, wmax, T;
output Real wOut;
algorithm
wOut := if noEvent(wIn >= wmax) then 0.0 else 1.0 / T;
end wAdd;
```

The qAdd function sets the change in the average queue length depending on the discrete packet drop function:

```
function qAdd
input Real q, w, T, C, N, R;
output Real qOut;
protected Real q1, q2;
algorithm
q1 := N * w / T - C;
q2 := q + q1;
qOut := if q2 > R then R - q else if q2 > 0.0 then q1 else -q;
end qAdd;
```

Additional restrictions on the size of $w(t)$ and $q(t)$ variables are set using the when operator in the Red class:

```
when w <= 1.0
  then reinit(w, 1.0);
end when;
when q >= R
  then reinit(q, R);
end when;
```

As a result of system modeling the dynamics of the change in $w(t)$, $q(t)$, $\hat{q}(t)$ parameters was obtained, presented in Figure 1. The graph shows that for some parameter values in the system there is a stable self-oscillating mode of operation.

4. Modeling in the Julia language

Julia is a high-level language designed for scientific and engineering calculations [2, 13, 14, 15].

Let us describe the implementation of an Active Queue Management algorithm with a RED control algorithm in Julia[1].

[1]We used the Julia-1.4.1.

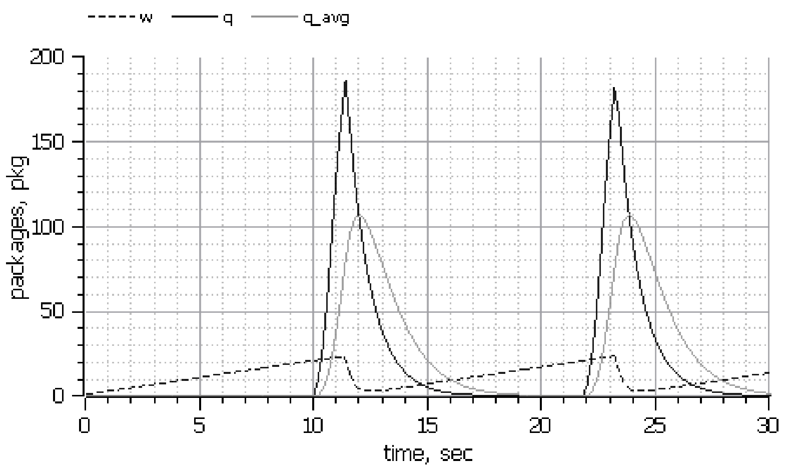

Figure 1: $w(t)$, $q(t)$, $\hat{q}(t)$ parameters behaviour based on Modelica modeling results

In this implementation, we used the DifferentialEquations package [16], designed for effective solution of differential equations of various types, such as ordinary differential equations, stochastic ordinary differential equations, differential algebraic and hybrid equations, as well as delay differential equations.

To install the package we will use the following command in Julia REPL:

```
using Pkg
Pkg.add("DifferentialEquations")
```

We will enable the package using the command:

```
using DifferentialEquations
```

We set the vector of the initial system parameters `p = (T, N, C, wq, q_min, q_max, R, p_max, w_max)`. The `pr` variable which is a function of the packet drop probability, acts as a global variable.

```
T = 0.5
N = 60.0
C = 10 * 1000000.0 / (8.0 * 500)
wq = 0.0004
q_min = 0.25
q_max = 0.50
R = 300.0
p_max = 0.1
w_max = 32.0
p = (T, N, C, wq, q_min, q_max,
   R, p_max, w_max)
pr = 0.0
```

Since the original system of differential equations contains delayed arguments, we define the history function `h(p, t)` which depends on the parameters vector `p` and time `t`. Next, we define the delay of the $w(t - T(t))$ variable in (1) equation:

```
h(p, t) = zeros(1)
tau = T
lags = [tau]
```

For our task, the `Red` dynamic function describing the behaviour of differential equations and setting constraints for w, q, $\hat{q}(t)$ parameters in DifferentialEquations will have the following form:

```
function Red(du, u, h, p, t)
w, q, q_avg = u
hist1 = h(p, t - T)[1]
du[1] = 1.0 / T -
        -(w * hist1 * pr / (2.0 * T))
du[2] = qAdd(q,w,T,C,N,R)
du[3] = -wq * C * q_avg + wq * C * q
end
```

In the `Red` function, we also set constraints for the w, q, $\hat{q}(t)$ parameters.

```
if (w <= 1.0)
    w = 1.0
end
if (q >= R)
    q = R
end
if (q_avg >= R)
    q_avg = R
end
```

Next we define the q parameter change function.

```
function qAdd(q,w,T,C,N,R)
    q1 = N * w / T - C
    q2 = q + q1
    if q2 > R
        return R - q
    elseif q2 > 0
         return q1
    else
        return -q
    end
end
```

One of the powerful tools of the DifferentialEquations package is callbacks, for which two functions are defined. A condition function is needed to check if an event has occurred. The affect function is executed if an event has occurred.

The discrete packet drop function is implemented as a controller which is updated every 0.01 seconds until the `tf` simulation time is reached. The `tstops` array defines the sampling intervals of the condition check, the function checks if `t` is one of the sample element.

```
tf = 30.0
tstops = collect(0:0.01:tf)
function condition_control_loop(u,t,integrator)
    (t in tstops)
end
```

Next, we determine the affect function, which is the controller. The `control_loop!` function at each step calculates the new value of the packet drop probability function p depending on the current values of the w, q, $\hat{q}(t)$ parameters in accordance with the formula (4).

```
function control_loop!(integrator)
    global pr
    w = integrator.u[1]
    q = integrator.u[2]
    q_avg = integrator.u[3]
    if (q_avg < q_min * R)
        pr = 0.0
    elseif (q_avg > q_max * R)
        pr = 1.0
    else pr = p_max * (q_avg/R - q_min) / (q_max - q_min)
    end
end
```

Then we define a discrete type callback:

```
cb = DiscreteCallback(condition_control_loop, control_loop!)
```

Finally, we define the vector of the system initial state, the simulation time and call the DDEProblem package solver. The `Red` function, vector of initial states, simulation time, and variables delay parameters are passed to its arguments:

```
u0 = [1.0, 0.0, 0.0]
tspan = (0.0, tf)
prob = DDEProblem(Red, u0, h, tspan, p, constant_lags=lags)
alg = MethodOfSteps(Tsit5())
sol = solve(prob, alg, callback = cb, tstops=tstops)
```

As a result of the simulation, we obtain a graph demonstrating the behaviour of the TCP Reno window size and the average queue length, i.e. reflecting the dynamics of a queue in a router (or gateway) with a queue management module using the RED algorithm (Figure 2).

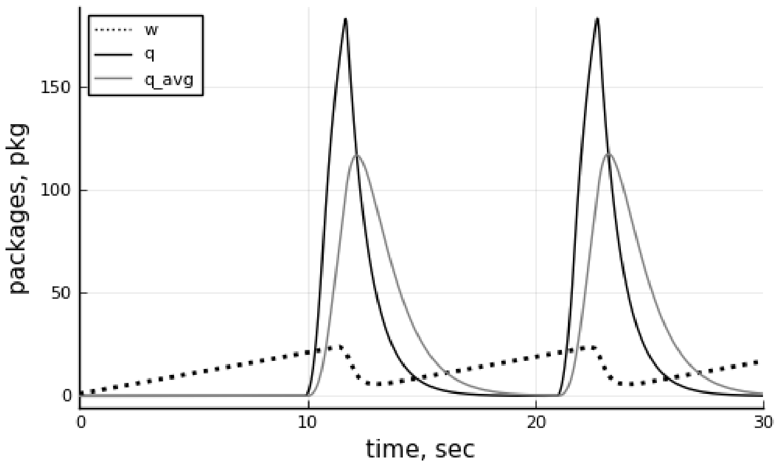

Figure 2: $w(t)$, $q(t)$, $\hat{q}(t)$ parameters behaviour based on Julia modeling results

5. Results

As a result of the study, two software complexes created in the programming languages Modelica and Julia were obtained. Both of these complexes implement the same mathematical model of a data transmission network with an active traffic control module operating according to the RED algorithm. A numerical experiment conducted within the framework of both software systems gives comparable results (see Figure 1 and Figure 2).

6. Discussion

Modelica and Julia languages are both domain-specific languages. However, Julia is considered as the common language of scientific calculations, Modelica is considered as a specialized language for modeling of dynamic, continuous and hybrid systems.

The mathematical model of the RED algorithm is formulated using a hybrid approach. Therefore, in the framework of the Modelica language, its implementation turned out to be quite simple. In this language ordinary differential equations are written with a delayed argument with the help of high-level tools quite easily. In addition, the use of discrete elements is implemented very clearly.

Julia language is aimed at solving a wider range of tasks. Therefore, it does not have as many syntax tools as the more specialized Modelica. In particular, the implementation of the hybrid paradigm in Julia requires a higher qualification of the programmer than when working with the Modelica language.

For specialized applications, Julia requires a greater level of knowledge than specialized languages such as Modelica. Julia is also a metalanguage and it can serve as the basis for the construction of other languages. For example, the Modia extension [17, 18] was made to migrate programs from Modelica to Julia (and possibly in the opposite direction).

7. Conclusion

The authors demonstrated the application of the continuous-discrete approach to the modeling of non-linear systems with control. The simulated system is the system consisting of an incoming stream processed according to the TCP protocol, as well as a router that processes traffic using a RED algorithm.

By comparing software implementations in the Modelica and Julia programming languages, the simplicity of hybrid systems modeling in Modelica is demonstrated, where continuous system elements, implemented by using a system of differential equations, interact with a discrete packet drop function quite well. Restrictions for some system parameters are set using the `when` operator. Implementation of the delay is also possible for system elements of any nature of functioning.

Julia also provides the ability to model systems according to a hybrid paradigm. The DifferentialEquations package allows to solve systems of differential equations of various kinds, including delay differential equations. The continuous probabilistic function has been successfully implemented using the callback option. A delayed argument to a continuous function is implemented using the `h(p, t)` history function.

Thus, the authors studied the capabilities of the Modelica and Julia programming languages in modeling of hybrid systems containing both continuous and discrete aspects of behaviour. The numerical simulation of the process of transmitting data via TCP and the process of regulating the flow state using the RED algorithm in the event of congestion has been performed, graphs demonstrating changes in the main parameters of the system have been obtained.

Acknowledgments

The reported study was funded by Russian Foundation for Basic Research (RFBR) according to the research project No 19-01-00645 (Dmitry S. Kulyabov and Anna V. Korolkova, mathematical model development). The publication has been prepared with the support of the “RUDN University Program 5-100” (Anna Maria Yu. Apreutesey, program development).

References

[1] A. V. Korolkova, T. R. Velieva, P. A. Abaev, L. A. Sevastianov, D. S. Kulyabov, Hybrid Simulation Of Active Traffic Management, Proceedings 30th European Conference on Modelling and Simulation (2016) 685–691. doi:10.7148/2016-0685.

[2] D. Färnqvist, K. Strandemar, K. H. Johansson, J. P. Hespanha, Hybrid Modeling of Communication Networks Using Modelica, in: The 2nd International Modelica Conference, 2002, pp. 209–213.

[3] A. V. Korolkova, D. S. Kulyabov, T. R. Velieva, I. S. Zaryadov, Essay on the study of the self-oscillating regime in the control system, in: M. Iacono, F. Palmieri, M. Gribaudo, M. Ficco (Eds.), 33 European Conference on Modelling and Simulation, ECMS 2019, volume 33 of *Communications of the ECMS*, European Council for Modelling and Simulation, Caserta, 2019, pp. 473–480. doi:10.7148/2019-0473.

[4] D. S. Kulyabov, A. V. Korolkova, T. R. Velieva, E. G. Eferina, L. A. Sevastianov, The Methodology of Studying of Active Traffic Management Module Self-oscillation Regime,

in: W. Zamojski, J. Mazurkiewicz, J. Sugier, T. Walkowiak, J. Kacprzyk (Eds.), DepCoS-RELCOMEX 2017. Advances in Intelligent Systems and Computing, volume 582 of *Advances in Intelligent Systems and Computing*, Springer International Publishing, Cham, 2018, pp. 215–224. doi:10.1007/978-3-319-59415-6_21.

[5] S. Floyd, V. Jacobson, Random Early Detection Gateways for Congestion Avoidance, IEEE/ACM Transactions on Networking 1 (1993) 397–413. doi:10.1109/90.251892.

[6] V. Misra, W.-B. Gong, D. Towsley, Fluid-Based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED, ACM SIGCOMM Computer Communication Review 30 (2000) 151–160. doi:10.1145/347057.347421.

[7] V. Misra, W.-B. Gong, D. Towsley, Stochastic Differential Equation Modeling and Analysis of TCP-Windowsize Behavior, Proceedings of PERFORMANCE 99 (1999).

[8] A. V. Korolkova, D. S. Kulyabov, L. A. Sevastianov, Combinatorial and Operator Approaches to RED Modeling, Mathematical Modelling and Geometry 3 (2015) 1–18.

[9] M. Hnatič, E. G. Eferina, A. V. Korolkova, D. S. Kulyabov, L. A. Sevastyanov, Operator Approach to the Master Equation for the One-Step Process, EPJ Web of Conferences 108 (2016) 02027. doi:10.1051/epjconf/201610802027. arXiv:1603.02205.

[10] P. Fritzson, Principles of Object-Oriented Modeling and Simulation with Modelica 2.1, Wiley-IEEE Press, 2003.

[11] P. Fritzson, Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica, John Wiley & Sons, Inc., Hoboken, NJ, USA, 2011. doi:10.1002/9781118094259.

[12] A.-M. Y. Apreutesey, A. V. Korolkova, D. S. Kulyabov, Modeling RED algorithm modifications in the OpenModelica, in: D. S. Kulyabov, K. E. Samouylov, L. A. Sevastianov (Eds.), Proceedings of the Selected Papers of the 9th International Conference ”Information and Telecommunication Technologies and Mathematical Modeling of High-Tech Systems” (ITTMM-2019), Moscow, Russia, April 15-19, 2019, volume 2407 of *CEUR Workshop Proceedings*, Moscow, 2019, pp. 5–14.

[13] J. Bezanson, S. Karpinski, V. B. Shah, A. Edelman, Julia: A Fast Dynamic Language for Technical Computing (2012) 1–27. arXiv:1209.5145.

[14] J. Bezanson, A. Edelman, S. Karpinski, V. B. Shah, Julia: A fresh approach to numerical computing, SIAM Review 59 (2017) 65–98. doi:10.1137/141000671. arXiv:1411.1607.

[15] A. Joshi, R. Lakhanpal, Learning Julia, Packt Publishing, 2017.

[16] C. Rackauckas, Q. Nie, DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia, Journal of Open Research Software 5 (2017). doi:10.5334/jors.151.

[17] H. Elmqvist, T. Henningsson, M. Otter, Systems Modeling and Programming in a Unified Environment Based on Julia, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications. ISoLA 2016, volume 9953 of *Lecture Notes in Computer Science*, Springer, Cham, 2016, pp. 198–217. doi:10.1007/978-3-319-47169-3_15.

[18] M. Otter, H. Elmqvist, Transformation of Differential Algebraic Array Equations to Index One Form, in: Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017, volume 132, 2017, pp. 565–579. doi:10.3384/ecp17132565.