Using Artificial States in Modeling Dynamic Systems: Turning Malpractice into Good Practice

Dirk Zimmer

German Aerospace Center (DLR), Institute of System Dynamics and Control, Germany
dirk.zimmer@dlr.de

Abstract

This paper analyzes the current use of artificial states in modeling practice and proposes a new form of equations for the purpose of modeling dynamic systems. These balance dynamics equations are used to formulate dynamic processes that help to find the solution of non-linear systems of equations.

Keywords: artificial states, continuation methods, language design.

1. Introduction

Any kind of formal modeling involves abstraction. The modeler has to study the given system and decide which parts are relevant and which are not. Typically a system contains many dynamic processes where only a small subset is of interest. For instance, in rigid body dynamics, the modeler chooses to ignore the elasticity of the applied material. In power-electronics with ideal switches, the modeler chooses to ignore the complicated switching behavior.

In an equation-based modeling language, the modeler will then provide equations for both parts. The dynamic processes that are regarded as relevant will be represented by differential equations. For other processes idealizations are provided in form of algebraic equation systems. Optimally, the resulting set of differential-algebraic equations has a set of state variables that precisely matches the dynamics of interest. In real modeling practice, this is infeasible for many cases.

In many applications, the modeler is forced to extend the dynamics of the system significantly beyond his area of interest. The reason for this aggravation is that otherwise the systems of non-linear algebraic equations resulting from the idealization of dynamic processes get too complex to be reliably solved by a general simulation engine. In order to avoid this, the modeler counteracts by including more state-variables in his system than he actually intends and thereby breaking the algebraic equation systems down. Consequently, these state variables are denoted as artificial since the dynamics of them are actually of no interest. They have been artificially introduced in order to enable a better computational realization of the simulation code.

This method of artificial states represents common modeling practice. It is applied in many different ways and comes along in many disguises. In mechanics, rigid detents get replaced by stiff spring-damper constructs. In electrics, micro capacitances or leakage currents are used without original intent. In bondgraphs, small-valued C or I elements are being added. And in this paper, we present two further examples that belong to the domains of thermodynamics and microeconomics.

Although the use of artificial states is common practice, it is not regarded as good practice. Instead it is often denounced as malpractice or as method of last resort that shall only be applied if all other potential remedies have failed. This is because of the significant disadvantages this method typically incorporates.

Since the artificial states mostly express dynamic processes whose time scale is orders of magnitudes lower than the time scale of actual interest, the system becomes very stiff. This requires the use of complex ODE-solvers for stiff systems, reduces simulation speed, and often prevents real-time capability of the simulation code. Furthermore, modeling the processes attached to artificial states requires parameters that are mostly of no interest or that cannot be retrieved in a meaningful way. This results in so-called fudge parameters whose values are arbitrarily stipulated but not based on any real data. Instead, the determination of these parameter values represents mostly a trade-off between the unwanted degree of stiffness and the unwanted loss of precision: a true choice between the devil and the deep blue sea.

Hence it is easy to understand why the use of artificial states seems strongly objectionable. The more rewarding question is to ask why this method is still being so frequently applied and why the recent progress in general M&S frameworks has not eradicated the need for this method. Why do modelers use a method from that they know it is bad? What forces them to use a method of last resort? And what is to say about all the other resorts?

This paper examines these questions and it will show that the method of artificial states is not bad per se. It is actually quite clever, a smart thing to do in many occasions, and, when conducted carefully, provides valuable insight into the modeled system. What is wrong

5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, 19 April, 2013, University of Nottingham, UK.

EOOLT 2013 website:
http://www.eoolt.org/2013/

about it is the way modelers are forced to apply this method in today´s M&S frameworks. Hence, we will suggest new constructs for modeling languages and new computational processing schemes for simulation engines. With these new tools at hand, the malpractice of artificial states will be turned into good practice.

But first let us look at some examples to expose the current dilemma.

2. Using Artificial States in Modeling Practice

In this section, we demonstrate the practical use of artificial states by the means of two examples. Both are realistic examples in the sense that they demonstrate the kind of problems that a modeler is typically confronted with in equation-based modeling languages. Both examples demonstrate the problems that forced the modeler to use artificial states although being initially reluctant.

2.1 Example 1: Energy Market Model

In the first example, principles from microeconomics are used for the management of energy flows [10]. The idea is the following: based on a market price each generator produces a certain amount of power and each load consumes a certain amount of power. The corresponding cost curves of generators and consumers are continuous monotonic increasing (Figure 1). The market price is then simple determined as the intersection between the two cost curves (Figure 2) for generators and consumers. In this way, a market model can be used to compute the power flow in an energy network.

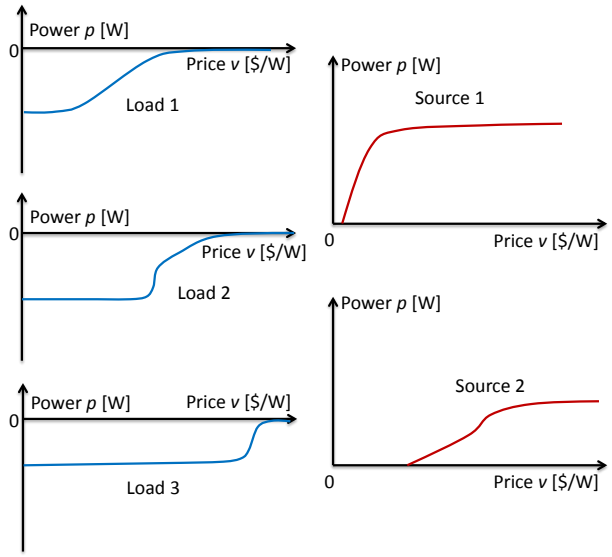

Figure 1: Cost curves

So far, so simple – but when we approach more sophisticated applications, things become a bit more difficult. Figure 3 presents the model diagram of a combined power generator whose outputs are electric and thermal energy. Up to 60% of the thermal energy can be converted into electricity.

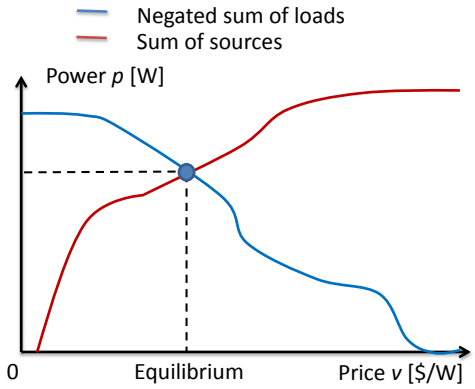

Figure 2: Equilibrium price

To this end, the power is split from the source (red component) into two sub-markets by the fixed split: 40% - 60%. Connected to this market are the consumers (blue). The thermal market, however, can take energy from the electric market but not vice versa. This is modeled by a one-way component that acts like an electric diode. The energy needs to be converted before reaching its consumer, hence the conversion element. The thermal energy can be wasted if inevitable, hence the waste element.

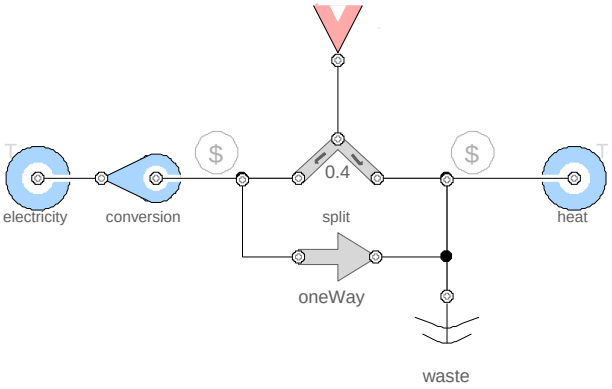

Figure 3: Model diagram of a combined power generator

The problem we get here is that we have two different market prices: one for electric energy and one for thermal energy. However these markets are not independent but coupled by algebraic equations. For instance, the model of the split component states that price at the generator is the weighted mean of the two consumer prices.

$$v_{in} = v_{out1} \cdot R + v_{out2} \cdot (1\text{-}R)$$

Hence we have a non-linear system with two iteration variables, namely the two prices of the electric and thermal market. This is certainly not exceptional and poses often no problems at all. However, in this particular case, it does. The cost-curves for the generator, the consumers, and the one-way limiter as well as the waste element all contain very flat and very steep gradients. This makes iterative, gradient-based solvers (such as Newton's method) difficult to apply since the convergence area is often very small. Finding the initial

solution requires a very good guess and steps of time-integration have to be small in order to stay within the area of convergence.

In order to approach a market solution in a more robust way, we provide a price controller. With this element, it is possible to find the solution in robust way by approaching steady state. Instead of having to determine the market price v directly such that the balance equation of power

$$p_1 + p_2 + p_3 = 0$$

holds, we make the more relaxed statement:

$$p_1 + p_2 + p_3 + p_c = 0$$

and control the price v by the lack or excess of power represented in p_c.

The corresponding controller is a very simple model that introduces an artificial state. It may compensate for any lack or excess of power p_c. The controller increases the market price in case of a power outflow ($p_c > 0$) due to a lack of power and decreases the price in case of a power inflow ($p_c < 0$) due to excess of power.

$$dv/dt \cdot T = p_c$$

where T is an arbitrary time constant. In the diagram of Figure 3, it is depicted as grey "$" placed in a circle. We can use such a price controller, because we know that the cost-functions are monotonic increasing. Any price advance will lower the demand and increase the provision of power and vice versa. This knowledge is not available to a non-linear solver but can be incorporated into the model in this way.

The incorporated disadvantages are a stiff system and that the simulation results are polluted by the dynamics of the price controller.

2.2 Exampe 2: Environmental Control System

The second example represents the modeling of a three-wheel bootstrap circuit from the environmental control system of classic aircraft architectures [7]. Here, air that is tapped from the aircrafts turbine (bleed air) is used to pressurize the cabin. Since the bleed air is hot (ca. 220°C) and at high pressure (ca. 2.5bar) [6], it needs to be cooled down and expanded before it enters the cabin. The idea is to use the energy gained in this expansion process to power a compressor and a fan for the ram air channel that is being used as cooling element. With those two devices joining the drive shaft of the expansion turbine, a more efficient cooling device can be designed.

Let us trace the path of the bleed air in the corresponding model diagram of Figure 4. The bleed air first passes the primary heat exchanger (PHX) for cooling and is then compressed before passing the main heat exchanger (MHX). Before entering the turbine for expansion, the water content needs to be extracted. Hence the bleed air passes a condenser and later on a reheater. These are both heat exchangers where the bleed air is actually interacting with itself at different stages in the circuit. Finally, after expansion, the air is sent to the mixer where it is being used to pressurize the cabin.

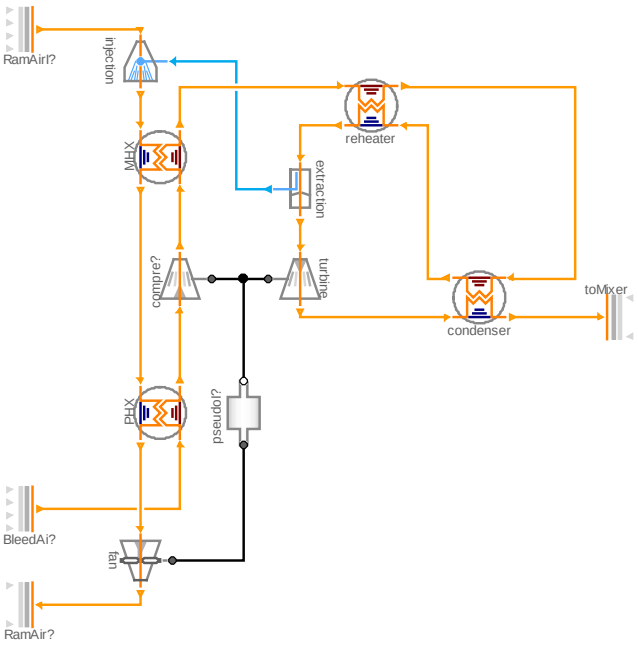

Figure 4: Model diagram of an environmental control system

In this model, we are only interested in the equilibrium point and not in any dynamics of the system at all. In the equilibrium point, the energy consumed by compressor and fan will balance the energy gain of the turbine. Furthermore all losses and gains of thermal energy in the heat exchangers cancel each other out. In the model, this equilibrium point is described by a set of pure algebraic equations. Due to the nature of thermal processes many of these equations are non-linear. The connections between the components in Figure 4 form many loops. This indicates that many of these algebraic equations are tightly coupled with each other[1]. And indeed when we have implemented the model in the modeling language Modelica, there results a very difficult non-linear system with more than 200 equations. Corresponding M&S frameworks like Dymola [3] are able to compress the system but even then a non-linear system remains with more than 40 iteration variables.

Solving such a complex system of equations in a robust manner is a very difficult task. But even when possible, a large system with more than 40 iteration variables significantly slows down the simulation engine as soon as the ECS becomes part of other dynamic processes.

For these reasons, artificial states have been used to tear the algebraic equations system apart. In total 5 state variables were sufficient to break down the non-linear equation system into individual non-linear equations that can be solved one after another.

[1] more technically: they represent a large block in the block lower triangular form of the equation system.

One of the state variables represents the velocity of the drive-shaft. A small inertia has been assigned to this shaft and hence any difference between turbine and compressor power does not need to be immediately balanced. Instead the difference can be used to accelerate or decelerate the drive shaft, as this happens in reality too. The precise value of the inertia I is not important here since we are not interested in the corresponding dynamics.

The inertia of the drive shaft introduces the following differential equation:

$$\tau = der(\omega) \cdot I$$

The variable ω represents the angular velocity of the drive shaft and is now an artificial state of the system. Its product with the torque τ determines the lack or excess of power that is (de-)accelerating the drive shaft.

The other four states are not mechanical inertias but thermal inertias. Although the physical domain is different, the applied methodology is identical.

By using artificial states, the model can be solved robustly and is open for further extension as for instance its inclusion into a complete aircraft energy system model. The amount of stiffness that is added to the system depends on the fudge parameters. However, for many practical applications, solving the stiff system is still faster than solving the original system simply because there is no complicated non-linear system with over 40 iteration variables to be solved.

3. Review of the method of artificial states

Let us review the methodology that can be extracted from the two examples. In both cases, the modeler generated a non-linear system of equations that turned out to be very difficult to solve. It is inappropriate to blame the numerical solvers for this. Without any further information no one can guarantee that any potential solver will find the correct solution[2]. Demanding for a better solver method to solve all of your problems is a pie in the sky.

It is important to understand that these non-linear system of equations result from a process of idealization. In example 1, we requested for a balance between power consumption and generation. The price had to be determined in such a way that the balance is met. Closer to reality is to regard the price determined by continuously ongoing negotiation. A lack of power leads to a higher prices and an excess of demand leads to lower prices. The balance equation simply idealizes this negotiation process by reducing it to an instant and letting it take immediate effect.

Also in example 2, balance equations are a source of idealization. The balance of power along the drive shaft ignores the inertia of the shaft and that it takes time to establish this balance. In many, many cases non-linear systems of equations result from the idealization of such balance dynamics.

[2] Presuming that there is exactly one solution or that there are multiple solutions of which any of them can be regarded as correct.

What happens now is particularly interesting. After the modeler has realized that he has gone too far and that his idealizations have created non-linear systems too difficult to solve, he reverts some of his idealization against his original intent. In example 1, the continuous process of negotiation has been reintroduced by a price controller. In example 2, mechanic and thermal inertia have been added to the system although the corresponding dynamics are of no interest.

The modeler understands that the system cannot be solved without some background knowledge that is inaccessible to the solver. It is inaccessible because it got lost in the process of idealization. For instance, the modeler knows the effects of price advance and price reduction and how to use that knowledge to derive a market solution. He also knows that inertias in physical systems help to balance the system.

But how can a modeler convey such valuable background knowledge into a general M&S framework? He sees no other way than to introduce artificial dynamics in his system and hence the method of artificial states becomes the weapon of choice. In this way, he abuses the time-integration of the simulator as a solver for his non-linear systems of equations.

When using artificial states, the modeler evidently makes a distinction between

- Dynamic processes that are relevant of the system under study.
- Dynamic processes that describe how to solve a non-linear system of equations.

Once we have become aware of this distinction, the problematic point about the use of artificial states becomes evident: The modeler makes this distinction but the M&S framework does not. It is not the modeler who wants to mix up things. He is forced to mix up things because the M&S frameworks do not provide adequate means to make a proper distinction between these two descriptions of dynamic processes.

The aim of a good modeling language should be to grasp the modeler's knowledge in a formal, clear and unambiguous way. So when the modeler knows which dynamics lead to the solution of a non-linear system of equations, any modeling language should encourage him to include this knowledge into his models in a proper form. After all, this represents valuable knowledge that can only be beneficial for the subsequent processes of code generation and simulation.

Hence the next chapter suggests a way, how such knowledge can be conveniently incorporated in a modeling language. It turns out to be surprisingly simple and intuitive.

4. Balance dynamics equations: Turning implicit idealization into explicit idealization

In the previous section, we stated that the idealization of balance dynamics is a very frequent source of non-linear system of equations. Let us therefore review the equations of the price controller from Example 1 that represent exactly one such example. First we had the desired ideal form for the balance of power flows:

$$p_1 + p_2 + p_3 = 0$$

In this case, the market price v has to be determined by a non-linear system of equations. Because of this, we relaxed the balance equation by introducing p_c and determined (or controlled) the price by means of a differential equation:

$$p_1 + p_2 + p_3 + p_c = 0$$
$$dv/dt \cdot T = p_c$$

We can derive the ideal form out of the balance dynamics by assuming a steady-state scenario and setting the derivative to zero. Furthermore, we assume that this steady state is continuously maintained and hence that the corresponding balance dynamics take instantaneous effect. This is like stating that the time constant is approaching zero.

This helps to understand the process of idealization and we can see that there are two major implications behind this idealization process:

1. The balance dynamics take place in no time; so they are regarded as infinitely fast.
2. The balance dynamics finally reach a stable steady state.

Since this pattern of idealization is so common in so many applications, it seems meaningful to enable its explicit formulation in a modeling language. To this end, a simple operator suffices.

Most modeling languages feature an operator for the time-derivative such as:

$$der(v) \cdot T = p_c$$

In strong resemblance to this operator, we can define and use a balance operator instead:

$$balance(v) = p_c$$

The operator balance(v) simply replaces the term der(v) · T and implies the two idealizations der(v) = 0 and T → 0. The fudge parameter T has consequently gone lost.

With this operator we have introduced a new kind of equation. We call them balance dynamics equations. They enable us to state the implicit assumption of the idealized algebraic equation in explicit form. In this way, you get the best of both worlds: you can interpret them as algebraic constraints in the simulation context but you can also interpret them as dynamic process in the solver context. How to precisely do that is content of the next section.

5. Handling balance dynamics equations in a simulation environment

In the following small example, we find an algebraic equation, a differential equation, and a balance dynamics equation:

$$der(x) = z$$
$$balance(y) = p(y)-x$$
$$z = \sin(y)$$

The balance equation now states two things: a non-linear equations system (0 = p(y)-x) and a dynamic process how to solve this system (der(y) = p(y)-x) based on the modeler's knowledge that p is a monotonic increasing function.

Consequently, these model equations can now be transformed in two different ways:

$der(x) = z$	$x = const$
$0 = p(y)-x$	$der(y) = p(y)-x$
$z = \sin(y)$	$z = \sin(y)$

The left version represents the *simulation dynamics*, the dynamics of relevance for simulation; the right version represents *the solver dynamics*, the dynamics needed to solve the non-linear system of equations. This solver dynamics is formulated as sub-simulation (a simulation nested within the main simulation) hence the states of the actual simulation (here: x) are held constant. There remains the question how to take use of such a sub-simulation.

The idea is of course that in case we fail to solve the non-linear equation (here: 0 = p(y)-x) directly, we use the sub-simulation on the differential equation (here: der(y) = p(y) − x) to get to the area of local convergence. But before we pursue this idea any further, let us highlight another benefit of balance dynamics equations. Balance dynamics equations do not only help solving non-linear equations by getting to the area of local convergence but they also provide information that helps to solve the actual system more efficiently once you are in this area.

Whenever an iterative numerical solver is applied to a non-linear system of equations, we need to determine a set of suitable iteration variables. These iteration variables are also often denoted as tearing variables since they are used to tear the algebraic loops apart and generate residual values instead. Choosing such iteration variables is a difficult task where many constraints have to be regarded [11]. This led to the choice of over 40 iteration variables for the ECS system in example 2.

Balance dynamics equations provide an excellent indication which variables to choose as iteration variables. Since they are assigned to a state-variable in the corresponding sub-simulation, this state variable must also be a suitable iteration variable. In this way, the number of iteration variables (and thereby the size) can be significantly reduced. For instance, in Example 2, the number of iteration variables can be reduced from over 40 to 5, leading to a much more efficient simulation.

This coincidence of iteration variables for the direct solution with the state variables for sub-simulation

indicates that these two tasks are actually closely related. Remember, the balance equation

$$\text{balance}(x) = f(x)$$

offers us two different ways to get to a solution. Either we solve the equation $0 = f(x)$ directly or we approach the steady state by a sub-simulation on the differential equation $\text{der}(x) = f(x)$. On the first look, this looks like two separate tasks. However, let us analyze how we would perform such a sub-simulation in practice.

After all, this is a special case: we do not perform a usual simulation; we want to perform a simulation for the sole purpose to approach the steady state with the time[3] $t \to \infty$. Which integration method would we choose for that?

Since the differential equation $\text{der}(x) = f(x)$ is supposed to describe a stable system, an implicit method is a strong favorite. Any explicit integration method would be limited in its step-size in order to maintain stability and approaching infinity with steps of finite width is an unpromising endeavor.

Since we do not care about the precise trajectory leading to the steady state and since the steady state solution itself is insensitive to the local integration error, there is no reason to choose any higher-order method. Order 1 is completely sufficient.

So our method of choice would be to perform Backward Euler with as large steps as possible. Hence, let us look at one integration step of this method going from t to $t+h$:

$$x_{t+h} = x_t + h \cdot \text{der}(x_{t+h})$$

or in our current example:

$$x_{t+h} = x_t + h \cdot f(x_{t+h})$$

Being an implicit method, we have to solve the system of equations: $0 = g(x_{t+h})$ with $g(x_{t+h})$ being defined as:

$$g(x_{t+h}) = x_t - x_{t+h} + h \cdot f(x_{t+h})$$

Evidently for $h \to \infty$, solving $g(x_{t+h})$ becomes equivalent to solving $f(x)$ directly. Now it becomes clear how the solver dynamics can support us to find the solution of $f(x)$. Instead of solving the system $f(x)$, we can solve $g(x)$ and in this way, we have won one important degree of freedom: we can choose h.

In this way, we have transformed the problem into a numerical continuation problem [1]. In general, a continuation problem results from transforming a function $F(x)$ to $F'(x,\lambda)$ with $\lambda \in [0,1]$ where $F'(x,1) = F(x)$ and $F'(x,0)$ is easy to solve. Many solutions methods have been developed for this kind of problem and they are already applied by many M&S Frameworks, mostly to solve initialization problems in a more robust way [9] for instance by using homotopy [8].

[3] Please note, the time t does not represent the main simulation time here but the time of the nested sub-simulation. The complete time-span of the sub-simulation represents only one instant in the main simulation.

To use numerical continuation solvers not only for initialization problems but also during simulation is also not a completely new idea. Artificial time integration is not uncommon to find solutions for PDEs [2]. The main difference to classic continuation problem in our case is that is not bounded by 1 but is free to go to infinity. Hence we have to adapt the continuation solver. The following paragraph sketches an algorithm that is a variant of the simplest kind of numerical continuation: the natural parameter continuation where h is our continuation parameter.

In case h is too large and our guess value for x_{t+h} is outside the convergence area, we can choose h small enough to be located in the convergence area again. And with each solution of $g(x)$, we step a little closer to the final solution of $f(x)$. In this way, we have found a robust way to solve our non-linear system of equations. Figure 5 depicts the corresponding algorithm of the balance dynamics solver.

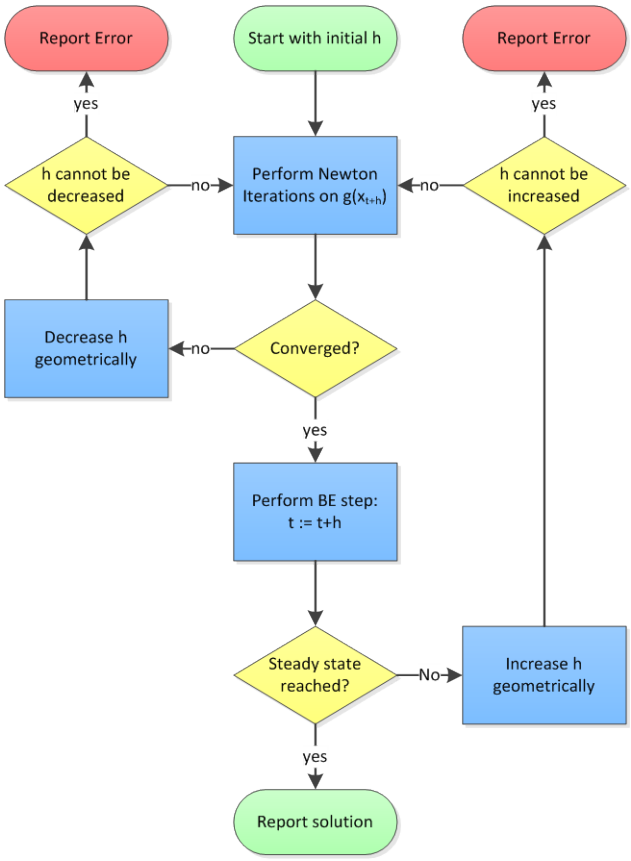

Figure 5: Algorithm for the balance dynamics solver

This algorithm becomes part of the main simulation loop and replaces the former direct solver for $0 = f(x)$. It is hence performed at each integration step of the main simulation task. It is not necessarily slower than the direct solver for $f(x)$. Having a high initial value for the sub-simulation step-size h and a good guess value for x_{t+h}, not many more iterations would be required than for a direct solution of $0 = f(x)$. A call to the direct solver is thus not required.

The difference occurs when good guess values for x_{t+h} are not available. In a normal setup, the integration step-size of the main simulation loop would be reduced in order to reobtain a good guess. Using our balance dynamics solver, this is unlikely to be necessary. More iterations would be needed in the solver to get the solution but the step-size of the main simulation loop can be maintained. And of course, finding the initial solution is also much simpler.

Without balance dynamics equations, the modeler has the choice of either creating a stiff system or a difficult non-linear system of equations. In both cases, he imposes severe limitations on the main integration step size and thereby creates a global damage even when only a small subsystem is actually concerned. With balance dynamics equations and a corresponding solver, the damage is kept local.

A final remark with respect to the algorithm in Figure 5: please note that the step-size control of h is not equivalent to classic step-size control in ODE solvers. It is based solely on the matter of convergence not on the matter of local integration error and hence can be performed much more aggressively.

6. Small application example

To prove the feasibility of this approach, we provide a small example. The following DAE

$$\begin{aligned} dx/dt &= y \\ dy/dt &= -0.1 \cdot a - 0.4 \cdot y \\ s(a) &= 10 \cdot x \end{aligned}$$

requires the solution of an expression containing the non-linear function s(a) displayed in Figure 6:

$$s(a) = \begin{cases} a/4 - 3/4 & \text{if } a < -1 \text{ then} \\ a/4 + 3/4 & \text{else if } a > 1 \text{ then} \\ a & \text{else} \end{cases}$$

with its derivative to be defined as

$$s(a) = \begin{cases} 1/4 & \text{if } a < -1 \text{ then} \\ 1/4 & \text{else if } a > 1 \text{ then} \\ 1 & \text{else} \end{cases}$$

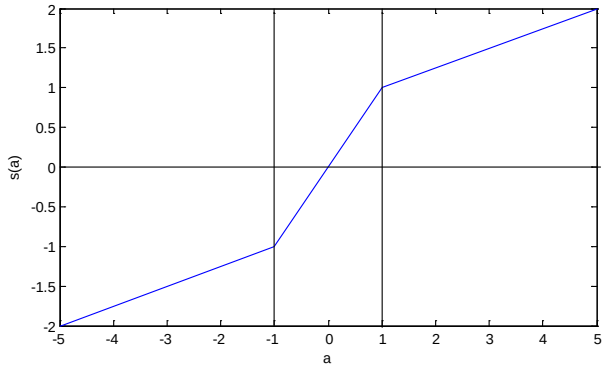

Figure 6: The piecewise linear function s(a)

The convergence area of solving s(a)=0 with respect to Newton's method is exactly [-1,1]. Although the convergence area is strictly limited, the solution can easily be found if one knows that s(a) is strictly monotonic increasing. We can incorporate this knowledge in form of a balance dynamics equation:

$$\begin{aligned} dx/dt &= y \\ dy/dt &= -0.1 \cdot a - 0.4 \cdot y \\ balance(a) &= 10 \cdot x - s(a) \end{aligned}$$

This DAE is now transformed into two forms for numerical ODE solvers.

- For the main ODE solver:

$$\begin{aligned} dx/dt &= y \\ dy/dt &= -0.1 \cdot a - 0.4 \cdot y \\ 0 &= 10 \cdot x - s(a) \end{aligned}$$

- For the continuation solver:

$$\begin{aligned} x &= const \\ der(a) &= 10 \cdot x - s(a) \end{aligned}$$

The main simulation is performed with Forward Euler and a step width of 0.1s for 100s. Without the continuation solver, the non-linear system of equations cannot be solved when the state variable x enters the range of [-0.1,0.1]. A step-size of smaller than 0.01s has to be taken in order to practically ensure the solvability of the system.

The continuation solver has been realized according to the algorithm sketched in Figure 5 and can robustly solve this system of equations. Figure 7 shows the simulation result. The step-size of the main-simulation loop is not impaired by the non-linear system anymore.

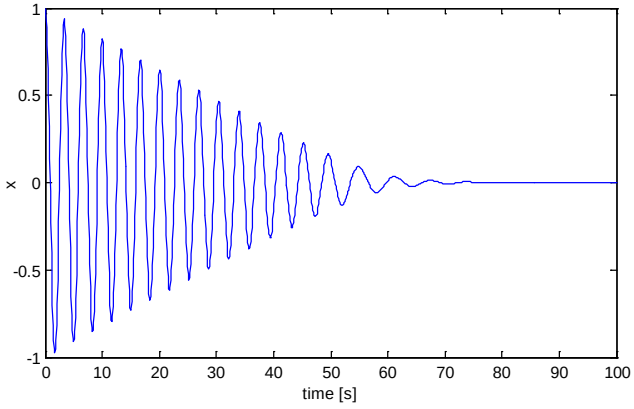

Figure 7: Simulation result showing the state x

Of course, this is a very small and simple example but it demonstrates that the basic idea works. For more mature implementation, we need to examine a number of interesting questions:

- How to optimally control the step-width h of the sub-simulation?
- How to provide suitable initial guesses of this step-width?
- When to stop sub-simulation when convergence is not reached?
- How should the step-size control of the main-simulation loop be controlled w.r.t to the convergence speed of the continuation solver?
- etc…

In this test-implementation, for instance, the initial step width h was chosen to be three times the minimum step-width that was required for the solution of the last step from the main simulation. This ensures that the initial value of h is relatively close to a prior successful continuation step while enabling a geometric increase of h for the case the continuation solver is actually not needed.

Figure 8 plots the number of times the function s(a) is being evaluated by the continuation solver in order to compute a new step or to check for convergence. Each value in the plot represents one time-step of the main simulation loop.

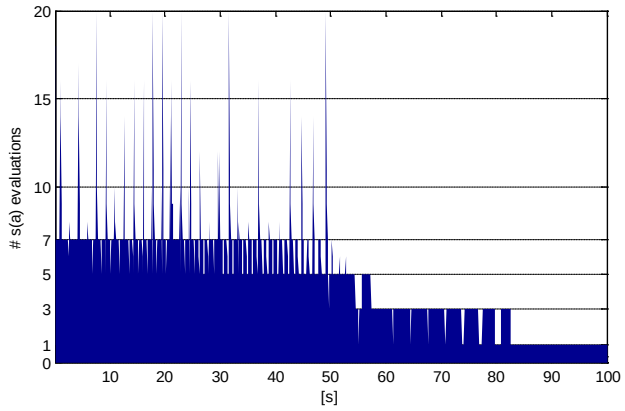

Figure 8: Number of function evaluations

During the first half of the simulation, the system oscillates with high amplitude. There are several peaks caused by the continuation method when several sub-steps had to be taken before convergence could be reached. This is where a direct solution with a gradient-based solver would have failed.

In the second half, there is no need for the continuation method anymore and a large initial step-size and an increasing quality of the guess value significantly reduce the computational effort. With the exception for one extra evaluation to check for convergence, the continuation solver hardly generates any additional burden anymore.

Even this rudimentary test implementation shows that a continuation solver is affordable for each time-step of the main simulation while the robustness of the solution can be improved. Solving at certain points might be expensive but when not needed the overhead is small. Because of the robust solution method, a large step-size of the main simulation can be afforded.

7. Prospective Limits of this Solution Method

The proposed variant for the natural parameter continuation is of course very simple and may not be able to solve all forms of balance dynamics. More elaborated continuation solvers support to deal with more complex continuation paths such as bifurcations of turning points. (There are complete libraries for continuation solvers such PyCont [5]). However, the need for such complex solvers is a warning and we shall rather question ourselves about the origin of this need.

After all, balance dynamics should be simple but practically oriented modelers will tell us that they can become pretty complex even stating self-contradictory sentences such as: “the value of fudge parameters is very significant”. Evidently, “fudge parameter tuning” can become an obsession. Why does this happen?

One important point is that in many complex applications, balance dynamics are layered. For instance, in the model of an environmental control system, there are balance dynamics resulting from physical inertia. There might also be balance dynamics resulting from sub-controllers. The modeler is actually interested in none of these processes but for the working of the sub-controller, it is important that the physical balance dynamics take place in a shorter time span. The resulting layering is illustrated in Figure 9:

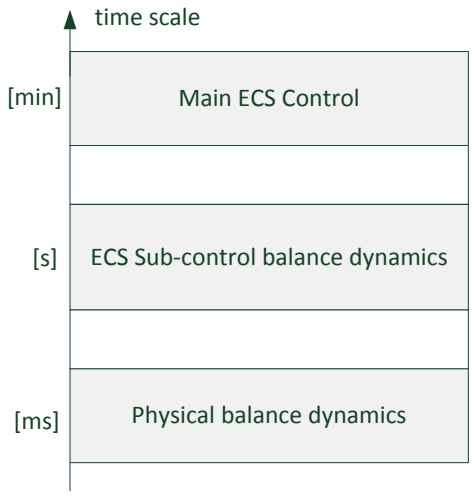

Figure 9: Layering of different balance dynamics

In classic modeling, the modeler is now using the fudge parameters to impose this layering. During this process, he typically makes a trade-off: on the one hand, he wants to separate the different balance dynamics and keep them in right order. On the other hand, he wants to reduce the stiffness of the overall system. Optimizing this trade-off is what is typically described as “fudge parameter tuning”.

Having this idea in mind, it seems now smart that instead of having one continuation solver to solve complex balance dynamics, we prefer nested continuation solvers, each of them solving one layer of simple balance dynamics. This requires that the modeler has means to separate different balance dynamics and to layer them. The currently proposed operator does not offer a sufficient solution for this.

8. Realization within a Modeling Language

The proposal as presented here is of course very easy to realize in most equation-based modeling languages. It is sufficient to add one single operator for the formulation of balance equations just as this had been done for the homotopy operator [8] in Modelica [4].

However, as temptingly simple as this seems, the balance operator as presented here will prove to be only partly sufficient. There are two major flaws involved with this solution:

1. The last section outlined the need to layer balance dynamics and to nest the corresponding continuation solvers. This is not possible with this operator notation.
2. Having available only this operator, it is not possible to reuse existing models (or components) of dynamic processes formulated with derivatives for the balance dynamics. The modeler is forced to remodel all relevant equations using the balance operator instead.

The second point of critique is valid also for the homotopy operator in Modelica. Typically, a modeler first builds a stiff system that includes the balance (or initialization) dynamics and then, in a second stage, he separates the two dynamics from each other. It would be favorable if the modeling of the second stage could reuse the components of the first stage.

For these reasons, we will finally need a better solution than the proposed operator but we can regard this proposal as intermediate solution in order to conduct the heavily needed research on this topic.

9. Conclusions

Since decades modelers use artificial states. Since decades they are being told that this is bad. Since decades they do it anyway. This is because formulating the dynamics that lead to the balance point of a sub-system is often the only way a modeler can explain how to solve his non-linear system of equations. It is unfortunate that M&S frameworks have not recognized this and provided better means for the modeler that enable him to distinguish between simulation dynamics and solver dynamics. This would prevent the rightfully criticized abuse of simulation dynamics to solve non-linear systems of equations.

For this purpose, we have proposed the concept of balance dynamics equations. It turns out that adding a simple operator is at least partly sufficient and a first step to explore the concept further. Balance dynamics equations can then be used to create code for a corresponding solver that is much more robust. Furthermore, the information contained in them can be used to make a better choice of iteration (or tearing) variables. Unwanted stiffness can be avoided and the integration step size of the main-integration loop is not needlessly limited. A difficult non-linear system of equations in a subcomponent will still increase the computational burden but the damage can be kept local.

This work so far is essentially based on theoretical thoughts and analysis of modeling experience. It needs to be put into practice and properly tested. Also balance dynamics equations do not provide never-ending salvation. They won't solve all modeling problems, but they have the potential to solve a big chunk of them. We hope for the future that this or similar methodologies are adapted by M&S Frameworks of industrial maturity.

Acknowledgements

I would like to thank Andreas Pfeiffer, Martin Otter and Michael Sielemann from DLR for suggesting several improvements.

References

[1] Eugene L. Allgower and Kurt Georg. *Introduction to Numerical Continuation Methods*, SIAM Classics in Applied Mathematics 45. 2003.

[2] U. Ascher, H. Huang, and K. van den Doel. Artificial Time Integration. *BIT Numerical Mathematics*, 47(1): 3-25, 2007.

[3] Dymola: available at www.dymola.com

[4] The Modelica Association. *Modelica® A Unified Object-Oriented Language for Systems Modeling - Language Specification Version 3.3*, Available at www.modelica.org, 2012

[5] PyCont available at: www2.gsu.edu/~matrhc/PyCont.html

[6] Rolls Royce. *The Jet Engine*. Rolls Royce Plc. Derby England. 278p. 1996.

[7] M. Sielemann, T. Giese, B. Oehler, M. Gräber, Optimization of an Unconventional Environmental Control System Architecture. In: *SAE International Journal of Aerospace*, 4(2):1263-1275. 2011

[8] M. Sielemann et. al., Robust Initialization of Differential-Algebraic Equations Using Homotopy. In: *Proceedings of 8th International Modelica Conference*. Dresden, Germany, 2011

[9] M. Sielemann and G. Schmitz, A quantitative metric for robustness of nonlinear algebraic equation solvers. In: *Mathematics and Computers in Simulation*, 81 (12), pp 2673-2687. Elsevier, 2011.

[10] D. Zimmer and D. Schlabe, Implementation of a Modelica Library for Energy Management based on Economic Models. *Proceedings of the 9th International Modelica Conference* , Munich, Germany (2012)

[11] D. Zimmer, *Equation-Based Modeling of Variable Structure Systems*. PhD Thesis, ETH Zürich, 219 p. 2010

Biography

Dr. Dirk Zimmer received his PhD degree from the Department of Computer Science at the Swiss Federal Institute of Technology (ETH Zurich). He is currently pursuing his research work at the Institute of System Dynamics and Control belonging to the German Aerospace Center (DLR). Also, he is lecturer at the Institute of Computer Science at the Technical University of Munich (TUM).