In *Computer Aided Conceptual Design '97*, Proc. 1997 Lancaster International Workshop on Engineering Design CACD'97, A. Bradshaw and J. Counsell (Eds.), Lancaster University Engineering Design Centre, Lancaster, U.K., ISBN 1-86220-026-2, 1997

Improving Dynamic System Model Building Through Constraints

Theo J.A. de Vries, Paul B.T. Weustink and Johannes A. Cremer

EL–BSC–RT, University of Twente, P.O.Box 217, 7500 AE Enschede, The Netherlands
phone +31–53–489 28 17; fax +31–53–489 22 23;
e–mail vri@rt.el.utwente.nl; WWW http://www.rt.el.utwente.nl/~vri

Abstract

An important step in model building that has been given little attention is the formulation of consistent initial state– and parameter vectors. In this paper we motivate that indeed this step is error prone. Improvement is obtained by incorporating in the model 'static relations' that limit the freedom in choosing values for the initial state– and parameter vectors. Such relations should be implemented by means of constraints. A prototype realisation in 20-sim is described.

Keywords: modelling, model building, object orientation, constraints, CAE systems

1. Introduction

A parameterised dynamic system model is a set of relations between state variables $\boldsymbol{x}$, parameters $\mathbf{q}$, input variables $\boldsymbol{u}$ and output variables $\boldsymbol{y}$.

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= \boldsymbol{f}\left(\boldsymbol{x}(t),\boldsymbol{u}(t),\mathbf{q}\right) \\ \boldsymbol{y}(t) &= \boldsymbol{g}\left(\boldsymbol{x}(t),\boldsymbol{u}(t),\mathbf{q}\right) \end{aligned} \tag{1}$$

During model building, the task is to determine the states, parameters, inputs, outputs and relations that together describe the system being studied for the given problem context. Our

main concern during this task is to obtain a *competent* description, that is, to properly answer the question what is actually relevant behaviour of the system being studied. Tools, techniques and methods have been (and are) developed to support this process, which are nowadays commonly gathered under the label object oriented modelling. In these developments, emphasis has been given to the formulation of the relations describing the dynamics, i.e., what is reflected by $\boldsymbol{f}$ and $\boldsymbol{g}$ in (1). Much progress has been achieved here, and in contemporary model building environments the proper formulation of the dynamics of a system is supported well. Examples of such systems are 20-sim (Controllab Products), Working Model (Knowledge Revolution), Schemebuilder [Sharpe et al., 1994] and Omola [Anderssen, 1995].

An important step in model building that has been given less attention is the formulation of consistent initial state– and parameter vectors. The goal of this paper is firstly to motivate that indeed this step is error prone and requires support, and secondly to propose a way to do so. To set the scope for the discussion, we start with characterising object oriented modelling in section 2. Next, we formulate the problem signalled above. In section 4 we argue that model building is facilitated if the representation of the model allows to observe and manipulate the initial state– and parameter vectors in an explorational manner. We discuss shortly how support for the formulation of consistent initial state– and parameter vectors is being realised in 20-sim and show an example of the tool in section 5. We finish with conclusions in section 6.

2. Object oriented modelling

For non-trivial models, a description of the form (1) is hard to interpret. To improve this, *object orientation* can be applied when describing models. Consider the system depicted in figure 1, which can be thought of as representative for a large class of mechatronic systems.

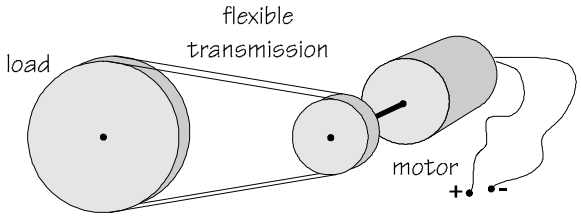

Figure 1: The Linix laboratory setup; the amplifier and computer–based controller are not shown.

If we use a modelling system that allows connections between subsystems that denote power exchange, we would probably model this system as shown in figure 2.

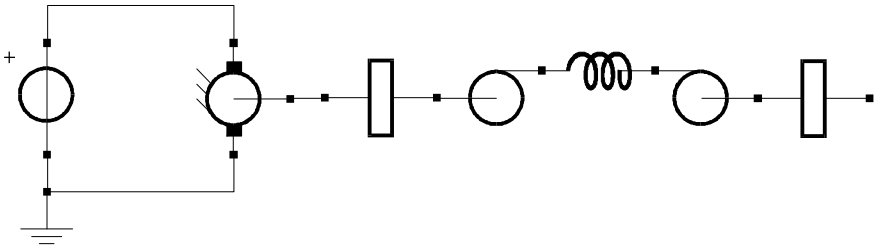

Figure 2a: Top level model of the Linix laboratory setup

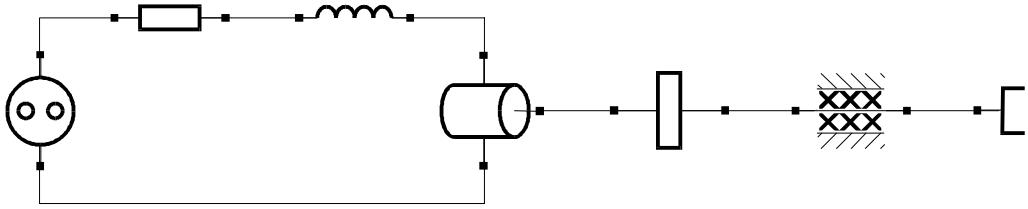

Figure 2b: Composite submodel of the motor

$$T = k_m \cdot I$$
$$V = k_m \cdot w$$

Figure 3: Elementary submodel of the electromechanical transducer

We would call such a model object oriented, because it features the following characteristics:

- *Encapsulation*; states, parameters, inputs and outputs are incorporated ('hidden') in submodels. In addition to these variables, submodels posses interaction variables that are encapsulated in ports.

- *Interconnection*; dynamic behaviour is obtained through the exchange of interaction variables between the submodels through connections between ports. Hence, the system is seen as a composition of interconnected (interacting) submodels.

- *Part–of hierarchy*; a submodel may be either a composition of lower level submodels or a set of relations between its interaction variables and its encapsulated states, parameters, system inputs and system outputs.

- *Abstraction*; the instantiation of a submodel is separated from the definition of its properties, thus enhancing model reuse.

Of these characteristics, the first three have immediate effect on the structure of a model. Hence, the general structure of an object oriented model is shown in figure 4.

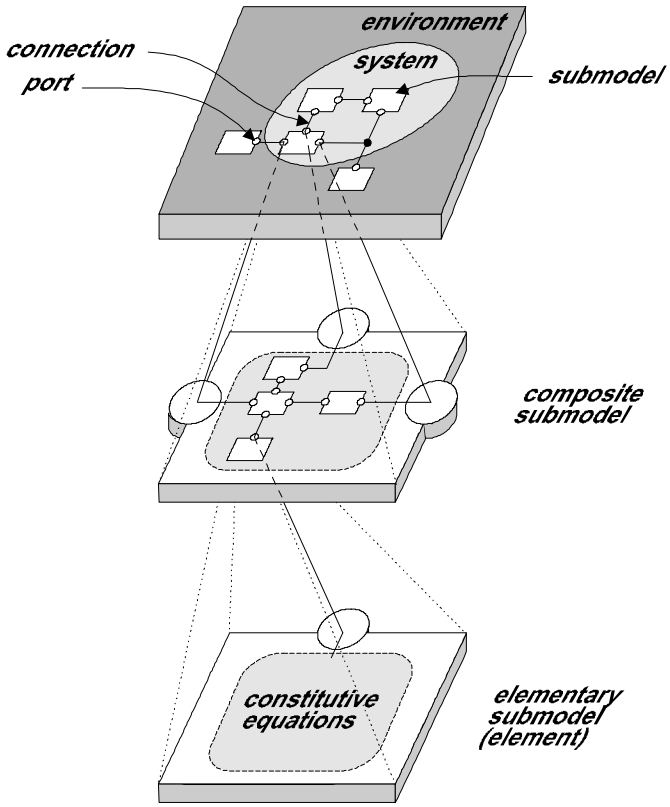

Figure 4: Object oriented model in general terms

When using an object oriented approach, modelling becomes the selection or creation of relevant submodels and the determination of their interconnections [De Vries, 1994].

3. Model consistency

Consider the Linix system of figure 1. Suppose we decide to increase the radius of pulley at motor side, as this will give us a more attractive transmission ratio. Hence, in our model we have to change this radius, which is encapsulated in the left-most wheel in figure 2a. However, if the radius of a pulley increases, also its moment of inertia will get larger. This parameter is encapsulated in another submodel, and the model building system will not alert us in any way if we forget this coupling of parameters. Moreover, if the flexible transmission has a non-linear spring characteristic, its stiffness will also vary if the radius of the pulley is enlarged. Both of these 'secondary' parameter adaptations are left at the responsibility of the modeller, and inconsistencies will arise easily.

A second example of the same effect is the following. For the purpose of controller design, one would like to simplify the model to a 4^{th} order one with as little elements as possible [Coelingh et al. 1997]. To this end, the inertia of the motor and of the pulley at the motor side will be aggregated into one mass parameter. When we apply a mechatronic design approach, we might decide during controller design to modify the value of this aggregated mass parameter. It would be helpful if the system supports this modification by adapting the original submodel parameters accordingly. When we test the controller on the non-simplified system, we do not have to worry about consistency of this extended model with knowledge upon which we based our controller design.

From these examples we may conclude that dynamic system models are generally under-constrained; values for parameters and initial states can be chosen with more freedom than is the case in reality. The parameters and initial states the system model are interdependent in an implicit way. As some of the parameters and/or initial states will change value during model building, this situation can easily lead to inconsistent models. This specifically holds for parameters and states that are encapsulated in different submodels.

To resolve this, the dependencies should be incorporated explicitly in the model. I.e., in addition to relations describing interaction, it is proposed to explicitly incorporate relations between initial state values and relations between parameters in dynamic models. These additional relations are not part of the dynamic system model formulated above, and hence are called *static relations* hereafter.

4. Manipulating static relations

4.1. Problem statement

The specification of static relations can be done in a kind of equation editor. It is however impossible to determine the causality of such relations during specification, as it is unpredictable which one of the involved parameters will change during model building, and what other parameter(s) should be adapted in order to satisfy a relation again. Therefore, static relations should be implemented by means of constraints, as constraint satisfaction is the appropriate technique to deal effectively and efficiently with such situations. A constraint system can guarantee consistency of the static part of the model and at the same time allows modification of values in any desired way.

4.2. Constraint satisfaction

The general properties a constraint system must fulfill are [Cremer, 1996]:

1. *Constraint specification*; the constraint system must be able to create the appropriate relations. Since both the element description as its representation are objects, these relations can be summarized as relations between attributes of one or more objects.

2. *Multi-way propagation*; the system must allow relations to be solved in more than one direction.

3. *Immediate propagation*; a change should not result in an unsatisfied relation for more than an instant.

4. *Efficient constraint satisfaction*; only the relations which contain changed attributes must be satisfied again, preferable in a way such that a minimum number of value changes takes place.

There are two categories of propagation techniques for constraint satisfaction, namely the *distributed technique* and the *centralized technique*

4.3 Distributed propagation

The distributed propagation technique relies on a dependency mechanism between the constraint variables. Constraints are registered as dependents of these variables. An example of such a system is given in figure 5.

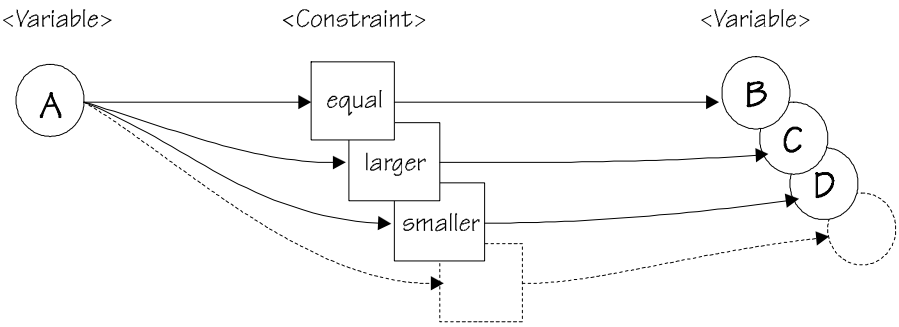

Figure 5: Distributed propagation mechanism

When variable *A* is changed, this change is signaled to each dependent constraint, resulting in a modification of variables *B*, *C*, and *D*.

This technique has some disadvantages, the most important ones being:

1. Solving the constraints can be *inefficient*. An object can change in a way that is relevant to just one of its dependent constraints, but yet each of them will be updated through the constraint object.

2. Two-way constraints are hard to implement, since circular dependencies may easy lead to *infinite recursion*.

4.4. Centralized propagation

A centralized approach will solve these disadvantages. The centralized propagation technique relies on one constraint system that includes all constraints. The values of variables are now *references* to the actual numerical values. The numerical values are maintained by the constraint systems. Figure 6 shows an example.

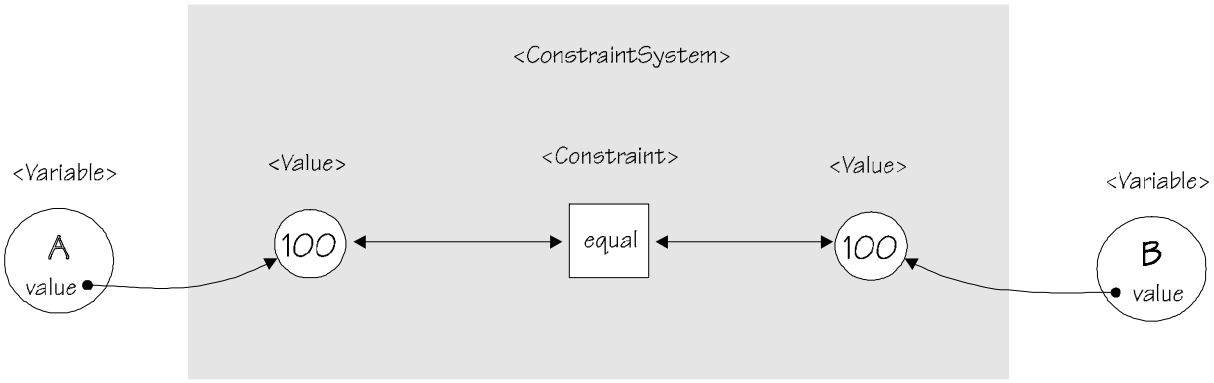

Figure 6: Centralized propagation mechanism

Instead of the variables in the distributed technique, the constraint system now controls when the values are set and checks whether a request to set a value is valid before the change is performed. This method is more efficient than the one used in the distributed technique, since only values that really need to be changed are set. Note that the variables cannot set or access their values explicitly, but make a request to the constraint system to set or retrieve its value.

The constraint system contains the constraints between attributes and is responsible for the addition and removal of these constraints. Multi-way constraints can now easily be created. The constraint system can determine a direction of the constraint, dependent on which of the relating variables undergo a change. In this way the possibility of infinite recursion, present in the distributed technique, can be eliminated.

4.5. Constraint invocation

The model building process is greatly facilitated if the *representation* of the model allows to observe and manipulate the static relations. That is, if a dynamic system model is represented by means of an iconic diagram, it should be possible to adapt parameters or state values of the submodels by changing the icons that represent these submodels. In order for this to work naturally, the causality of the static relations should not be fixed and multi-way constraints are necessary. Suppose that the parameter of a spring, the spring stiffness, is represented by the width of its representation (as shown in figure 9). A change in this parameter value must result in a change in the width of the spring figure, but a change in the width must also propagate to the parameter value to maintain consistency.

5. Realisation

5.1. Sky Blue constraint solver

For the implementation in 20-sim, the SkyBlue constraint solver developed by Sanella [1993] has been chosen. This solver is part of the HotDraw package [Johnson, 1992] which is the basis for all graphical editors of 20-sim.

A SkyBlue constraint contains input and output variables and one or more methods. A method is a mathematical procedure that takes the values of the input variables and calculates the values for the output variables. For example, the constraint $A = B + C$ can consist of three methods as shown in figure 7 [Cremer, 1996[.

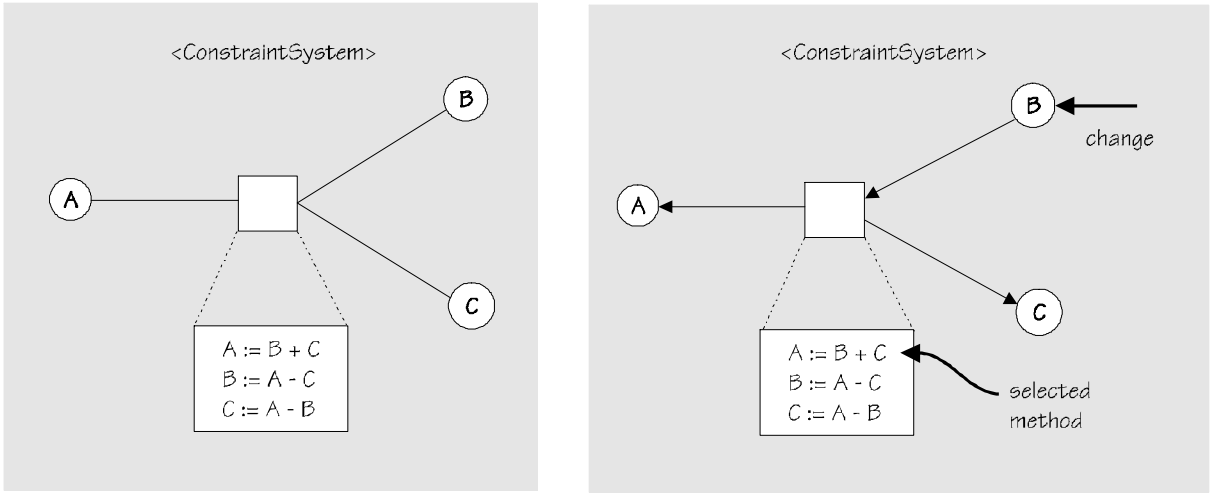

Figure 7: The constraint graph results is a directed graph when a method is selected to solve the constraint

If the value of B or C is changed, SkyBlue can maintain the constraint by executing the method $A := B + C$. This method is called the *selected method* of the constraint. The task of a constraint solver like SkyBlue is the following. Given is a *constraint graph*, which is a set of constraints on a set of variables. Suppose one or some of the variables get new values. SkyBlue will select and execute the methods that will adapt the values of the other variables, such that all constraints are satisfied (resulting in a directed graph as shown in figure 7).

The SkyBlue constraint solver caches so-called 'plans' that specify a method sequence such that all constraints are satisfied again when a particular constraint variable changes. Additionally, SkyBlue allows constraints to have associated weights. These weights can be

used to indicate preferences for propagation, for instance to let the method *C := A - B* in figure 7 be the first selected method. Another important advantage of a constraint solver like SkyBlue is that many types of constraints can be applied. This makes it perfectly suitable for creating the static relations. As constraint graphs become very large, the performance of the constraint solver deteriorates, but SkyBlue offers several techniques to improve its performance.

5.2. Short example

To give an impression of the created tool, consider the following. The icon attributes of a mass and spring can be related to their parameters. For a mass, this is shown in detail in figure 8. Standard value editors are also available to manipulate parameter values.

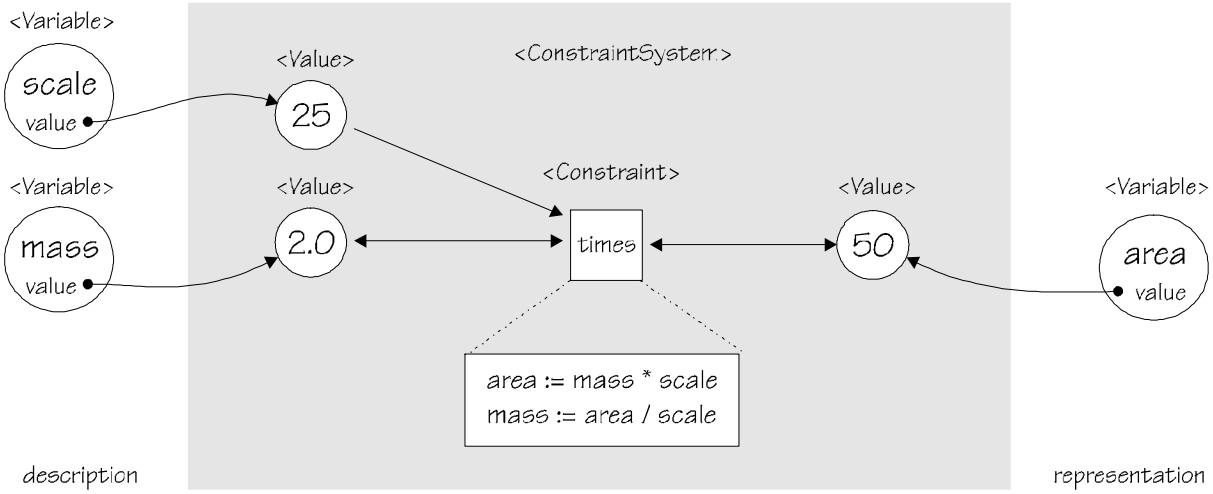

Figure 8: Constraint to represent the mass parameter as the area of a figure

The model of section 3 is simplified and transformed such that a 4th order model is obtained that is represented by two mass icons and a spring icon, see figure 9. Note that mass m2 equals the addition of the motor inertia and the inertia of the pulley at the motor side. The value of a mass can be increased by means of a value editor, and will result in an increase of the area of its icon, see figure 9. Alternatively, the user may drag on the icon, resulting in an update of the value editor. In an analogous way, the initial compression and the stiffness of the spring can be manipulated.

The representation of the model allows immediate interpretation of the relative sizes of m1 and m2. This is powerful, as an expectation of the dynamic behaviour of the system is thus

formed. However, absolute values of the masses cannot be seen immediately, as they depend on a scale factor. If the value of m2 is changed, the constraint system will instantaneously adapt any or both of the values of the inertia's of the motor and pulley, such that parameter values are consistent at all times.

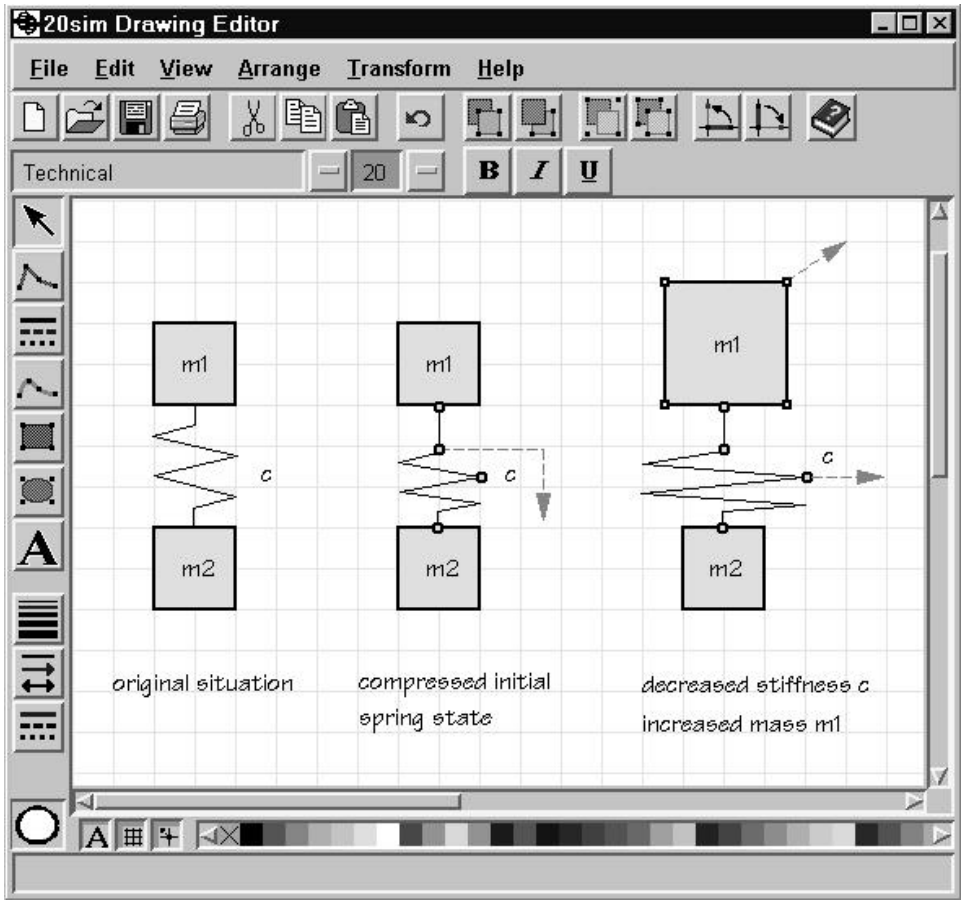

Figure 9: Manipulation of parameters and initial states via icon adaptation

In a companion paper [Coelingh et al., 1997], a more elaborate example of the use of constraints for dynamic system model building is discussed.

5.3. Constraints are no good for simulation

Constraints might be used to implement dynamic relations as well. However, these dynamic relations appear in larger numbers usually and are generally not solved during model building, but only during simulation. In this phase, they need to be solved many times, often in an iteration loop. This makes the use of constraints for this purpose less attractive. When the

causality of the dynamic relations are determined explicitly before calculation is started, simulation can be done much more efficiently.

6. Conclusions

Dynamic system models are generally under-constrained in the sense that values for parameters and initial states can be chosen with more freedom than in reality is the case; they are interdependent in an implicit way. As some of the parameters and/or initial states will change value during model building, this situation can easily lead to inconsistent models. This specifically holds for parameters and states that are encapsulated in different submodels. To resolve this, the dependencies should be incorporated explicitly in the model in the form of so-called static relations. The specification of static relations can be done in a kind of equation editor. It is however impossible to determine the causality of such relations during specification, as it is unpredictable which one of the involved parameters will change during model building, and what other parameter(s) should be adapted in order to satisfy a relation again. Therefore, static relations should be implemented by means of constraints, as constraint satisfaction is the appropriate technique to deal effectively and efficiently with such situations.

To allow for proper manipulation of parameters and initial states, the representation of a model should be able to reflect the particular values of states and parameters, but not necessarily the underlying constraints. The reflection of values can always be done alpha-numerically. The interpretation of a model may be improved greatly if the icons representing the submodels also reflect the values of (some) parameters and initial states. From our prototype realisation we have learned however that this is not generally the case.

References

Anderssen, M. [1995], *OmSim and Omola Tutorial and User's Manual*, version 3.4, Dept. of Automatic Control, Lund Institute of Technology, Sweden

Coelingh, H.J., T.J.A. de Vries, J. van Amerongen [1997], *Automated Conceptual Design of Controllers for Mechatronic Systems*, submitted to CACD'97 Lancaster International Workshop on Engineering Design, Lancaster, U.K.

Controllab Products [1997], *20-sim Reference Manual*, Enschede, The Netherlands, http://www.rt.el.utwente.nl/20sim/

Cremer, J.A. [1996], *Dynamic model representations in MAX*, MSc thesis, Control Laboratory, Dept. Electrical Engineering, University of Twente, Enschede, The Netherlands

Johnson, R.E. [1992], *Documenting Frameworks using Patterns*, Proceedings of Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA) 1992, Vancouver, British Columbia, Canada

Knowledge Revolution [1994], *Working Model Reference Manual*, San Mateo, California, http://www.workingmodel.com/

Sanella, M. [1994], *The SkyBlue Constraint Solver and Its Applications*, Proceedings of the 1993 Workshop on Principles and Practice of Constraint Programming, MIT Press, Cambridge

Sharpe, J.E.E., and R.H. Bracewell [1994], *A Computer Aided Methodology for the Development of Conceptual Schemes for Mixed Energy-Transforming and Real-Time Information Systems*, Proceedings of the 1994 Lancaster International Workshop on Engineering Design, Lancaster, U.K.

Vries, T.J.A. de [1994], *Conceptual design of controlled electro-mechanical systems*, PhD thesis, University of Twente, Enschede, The Netherlands