Modeling of Multibody Systems with the Object-Oriented Modeling Language Dymola

M. Otter
Institute for Robotics and System Dynamics, German Aerospace Research Establishment Oberpfaffenhofen (DLR), Postfach 1116, D-82230 Wessling, Germany, e-mail: df43@master.df.op.dlr.de

H. Elmqvist
Dynasim AB, Research Park Ideon, S-223 70 Lund, Sweden, e-mail: Elmqvist@Dynasim.se

F. E. Cellier
Department of Electrical and Computer Engineering, The University of Arizona, Tucson, Arizona 85721, U.S.A., e-mail: Cellier@ece.arizona.edu

Abstract. The object-oriented modeling language Dymola allows the physical modeling of large interconnected systems based on model components from *different engineering domains*. It generates *symbolic* code for different target simulators. In this paper, a Dymola class library for the efficient generation of the equations of motion for multibody systems is presented. The library is based on an O(n) algorithm which is reformulated in an object-oriented way. This feature can also be interpreted as a bond graph oriented modeling of multibody systems. Furthermore a new algorithm for a certain class of *variable structure* multibody systems, such as systems with Coulomb friction, is presented, which allows the generation of efficient symbolic code.

Key words: Object-oriented modeling, multidisciplinary simulation, multibody systems, variable structure

1. Introduction

Dymola [6, 8, 4, 5] is an object-oriented modeling language for modeling of large dynamical systems. Models are hierarchically decomposed into submodels which are connected in accordance with the *physical coupling* of the components. The features of Dymola allow the development of domain specific class-libraries for e.g. electronic circuits, control systems, hydraulic systems, thermodynamics systems, bond graphs and others, which can all be used in conjunction for generating a specific multi-domain application model. Dymola generates efficient *symbolic* code for several target simulators[1]. It can handle ordinary differential equation models as well as differential-algebraic equation (DAE) models. If a DAE is of higher index, certain parts of the equations are symbolically differentiated according to the algorithm of Pantelides [27].

In this paper it is shown how variable structure multibody systems can be modeled by Dymola. For this purpose a Dymola library based on the recursive O(n) algorithm of [1] is explained in detail. The algorithm was modified in order to arrive at an object-oriented formulation as encouraged by Dymola. This means that physical objects of the multibody system are mapped to corresponding Dymola objects, which are connected in accordance with the physical coupling of the system (this is *no* block-diagram representation of input/output blocks). The class description of an object contains the equations that describe the object, and the *cut*-definitions, i.e., definitions of the interfaces of that object to other objects. Due to this procedure, the equations of motion of a multibody system are determined by Dymola using only the connection structure of objects and the *local* information about objects. It turns out that this description form has close relations to the bond

[1]Presently, Dymola supports ACSL[22], DESIRE[17], SIMNON[7], SIMULINK[20] and Fortran in either Simnon- or DSblock-format[24]

graph methodology [16, 4]. In fact, this object-oriented definition of a multibody system can be interpreted as a multibody bond graph[2].

Besides a reformulation of an O(n) algorithm, a new enhancement of this algorithm is presented that allows the efficient treatment of a certain class of variable structure systems, i.e., systems with varying degrees of freedom. In the last years, methods to handle multibody systems with variable structure have been considerably improved [18, 28, 13, 12]. Usually, such systems are modeled by *numerical* multibody programs. This is due to the fact that, if n independent switches are present to remove or add one degree of freedom, 2^n different configurations are possible. For example, if dry friction is present in the joints of a 6 degree-of-freedom (dof) robot, $2^6 = 64$ configurations are possible, since every joint can either be in sliding (= 1 dof) or in sticking (= 0 dof) mode. *Numerical* multibody programs are designed to handle a large class of multibody systems with the same program. If such a feature is present, it is not principally difficult to change the configuration (i.e., the multibody system) during integration. On the other hand, a *symbolic* multibody program generates code for a *specific* multibody system only. Therefore, 2^n different codes have in general to be generated, if there are 2^n possible configurations (= multibody systems). Even for a modestly sized multibody system, such as a 6 dof robot with dry friction, this would be quite impractical. It will be shown that a simple modification of the recursive O(n) algorithm circumvents this difficulty and will allow the *symbolic* generation of compact and efficient code for variable structure systems.

Dymola together with its multibody library is comparable to commercially available multibody programs both in terms of efficiency and ease-of-use. However as already noted, Dymola can easily model components from other engineering domains in conjunction with the multibody components by invoking them from other already available class libraries. For example, a sophisticated library for electronic components corresponding to the SPICE electronics program [23] (diodes, Zener diodes, tunnel diodes, BJTs, etc.) is being developed at the University of Arizona [14]. In contrast to other multibody programs, Dymola supports *multidisciplinary* modeling within *one* environment — the multibody part being just one model component among other equally important engineering disciplines.

2. Detailed robot model

Dymola is introduced by means of a detailed dynamic model of the industrial robot Manutec r3 described in [11]. The structure of this "multidisciplinary" model is shown in Figure 1. The 6 degree-of-freedom robot consists of a system of rigid bodies connected by ideal revolute joints. Every joint is driven by a torque, produced by the electro-magnetic field of a current-controlled DC-motor and transformed by gear-boxes. The motors are controlled by decentralized cascade controllers. The block "rotor+gear" in Figure 1 contains the mechanical part of a motor and of its gear-box.

Dymola supports a *hierarchical decomposition* of models. The Dymola model can therefore be designed such that it directly reflects the model structure as shown by the definition of the robot in the Dymola model language in Figure 1 ("..." indicates similar items that have been omitted in order to shorten the text).

According to the different block component types of Figure 1, model classes are defined in a library that is made available to the model (object) definition through the command *@r3.lib* (the *@* operator tells Dymola to *include* the file *r3.lib*). New models (or objects) of these class definitions are instantiated by the command:

submodel *(class-name) object-name (parameters) ...*

For example the statement "**submodel** *(R3control) c1, c2, c3, c4, c5, c6*" instantiates six identical objects of the same class *R3control*. This is meaningful since, in the r3 robot, six identical cascade

[2]Someone not familiar with the bond graph methodology can just skip related paragraphs in this paper.

```
@r3.lib     {use library r3.lib}
model R3robot
  submodel (R3control) c1, c2, c3, c4, c5, c6
  submodel (R3motor)  m1 (Ci=5.22E-7,...), ...
  submodel (R3drive1) d1 (i=i1,...), ..., d3(...)
  submodel (R3drive2) d4 (i=i4,...), ..., d6(...)
  submodel (R3mbs)    rob {multibody system}

  constant i1=−105, i2=210, ..., i6=−99
  input rq1, rq2, ..., rq6  {required angles}
  input rw1, rw2, ..., rw6 {required angular vel.}

  connect c1 to m1 to d1 to rob:j1
  connect c2 to m2 to d2 to rob:j2
  connect c3 to m3 to d3 to rob:j3
  connect c4 to m4 to d4 to rob:j4
  connect c5 to m5 to d5 to rob:j5
  connect c6 to m6 to d6 to rob:j6

  {Controller input = gear-ratio · model-input}
  c1.rq  =  i1*rq1
  c2.rq  =  i2*rq2
        ...
  c6.rw  =  i6*rw6
end
```

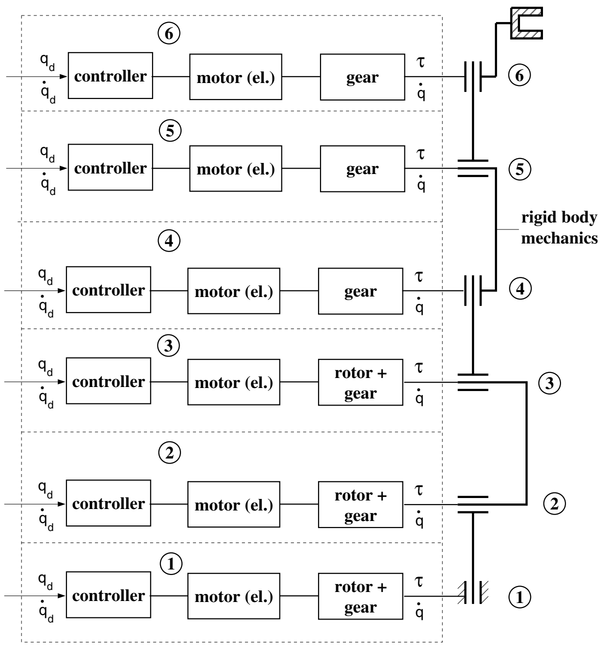

Figure 1: Model structure of Manutec r3 robot

controllers are used. In contrast, the six motors of the robot have the same *structure* but different parameters. Therefore the class *R3motor* is used to describe the motor structure, whereas a specific motor is defined by supplying appropriate values for the motor parameters. Since the robot employs gear-boxes of two structurally different kinds, two different classes (*R3drive1, R3drive2*) are provided. Finally an object of the multibody description (i.e., the robot itself without the motors, gear-boxes and controllers) is instantiated from class *R3mbs.*

With the statement "**constant** *i1=−105,...*" the gear ratios i_i are defined as constants. They are used in two different places (gear-box and input of controller). With the statements "**input** *rq1,...*" the input signals of the overall model are defined. These are the required angles and angular velocities of the joints as input to the controllers.

All the objects are assembled by the **connect** statements. The meaning of a connect statement such as "**connect** *c1* to *m1*" is defined in the corresponding class description, as will be explained below. Note, that there is no signal *direction* associated with a **connect** statement. It is therefore *not* an input/output block-diagram description! Instead, the **connect** statement reflects the *physical* coupling of components. If an object has several different interfaces, also called **cuts**, the notation *object:cut* is used. For example, "**connect** *d1* to *robot:j1*" states that gear-box 1 is attached at **cut** *j1* (= joint 1) of object *robot.* At the end of the model description, the connections between the global input signals and the model are specified. For example, "*c1.rq = i1*rq1*" states that the input variable *rq* of controller *c1* is the required angle *rq1* multiplied by the gear-ratio *i1.*

To this point, the topmost model components have been assembled. We shall now look at the component models themselves. Motors are described by objects of class *R3motor*, which makes use of a basic Dymola library for electrical components such as resistors, capacitors, and operational amplifiers. Let us first have a look at one of the classes of this library in order to familiarize ourselves with some important concepts of Dymola. In Figure 2 the definition of a *capacitor* from the aforementioned library is shown.

A capacitor is an element with two **cuts** *A* and *B*, through which it can be connected to other elements. With each cut two variables are associated, the potential at the wire *(Va, Vb)* and the

```
model class capacitor
      parameter C
      cut A (Va / i), B (Vb / −i)
      main cut   cAB [ A, B ]
      main path pAB <A − B>
      local u

      Va − Vb  = u
      C*der(u) = i
end
```

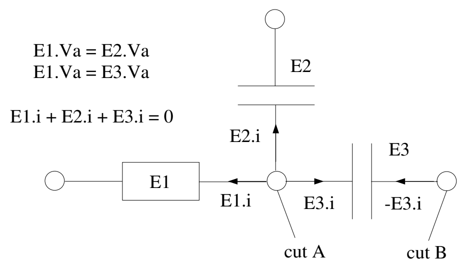

Figure 2: Capacitor definition and Dymola connection rules

current flowing into the component through the wire *(i, −i)*. Dymola distinguishes between two types of variables: *across* and *through* variables. In a **cut** declaration, all variables to the left of the slash operator ("/") are defined to be *across* variables, whereas all variables to the right of the slash operator are defined to be *through* variables. Accordingly, the potentials *Va, Vb* are considered to be *across* variables, whereas the current *i* is a *through* variable. The difference between these two variable types becomes apparent in a connection only. Assume for example, that three electric elements E1, E2, E3 are connected at one node as indicated in Figure 2. The Dymola built-in rules will generate the shown equations, i.e., the *across* variables at a node are set equal, whereas the *through* variables at a node are summed up to zero. For electrical circuits, these rules correspond to Kirchhoff's voltage law and Kirchhoff's current law, respectively. The statements "**main cut ...**" and "**main path ...**" in the capacitor definition state the default connections that are used if no cut-names are explicitly specified in the connect statement. For example, the statement "**connect** *C* **at** *(n1,n2)*" is equivalent to the longer specification "**connect** *C:A* **at** *n1, C:B* **at** *n2*.

The physical laws of the capacitor are programmed in the last two statements using the cut-variables and the **der** operator that characterizes a *derivative*. An important feature of Dymola is that no *direction* is associated with variables. Based on the connection structure of the overall model and the problem description, Dymola will determine on its own for which variable each of the equations needs to be solved, and apply symbolic formula manipulation to transform the equations to the desired form, if necessary. Due to this feature, Dymola supports the use of true *equations* rather than simple assignment statements.

Let us now return to the robot example. Using the aforementioned library of basic electrical components, the motor shown in Figure 3 can be defined in the following way:

```
model class R3motor
    submodel (Resistor)       Rd1(R = 100), Rd2(R = 100), Rd3(R=100), Rd4(R = 100)
    submodel (Resistor)       Ri  (R = 10)  , Rp1(R = 200), Rp2(R= 50), Ra  (R = 250)
    submodel (Capacitor)      Ci  (C = Ci)
    submodel (Inductor)       La  (L = La)
    submodel (Emf)            emf(k=k)          {elektro-motoric force}
    submodel (Hall)           hall              {Hall sensor}
    submodel (OpAmpIdeal) diff, I, power        {ideal operational amplifier}
    submodel (Ground)         g
    parameter Ci, La, k
            cut   control (q, qd, Vd)
            cut   rotor   (q, qd, qdd / f)
    main cut      mc      [control, rotor]
    main path     mp      < control − rotor >

    node    n0, n1, n2, n3, n4, n5, n6, n7, n8, n9
    connect Rd1 at (n2,n5),   Rd2 at (n1,n2),         Rd3 at (n4,n3),        Rd4   at (n3,n0),
            Ri  at (n5,n6),   Rp1 at (n8,n9),         Rp2 at (n8,n0),        Ci    at (n6,n7),
            g   at n0,        diff at (n3,n2,n5),     I   at (n0,n6,n7),  power at (n7,n8,n9),
            n9 to Ra to La to emf to hall to n0,   hall:m at n4,   rotor at emf:mech
    Vd = Rd2:a
end
```

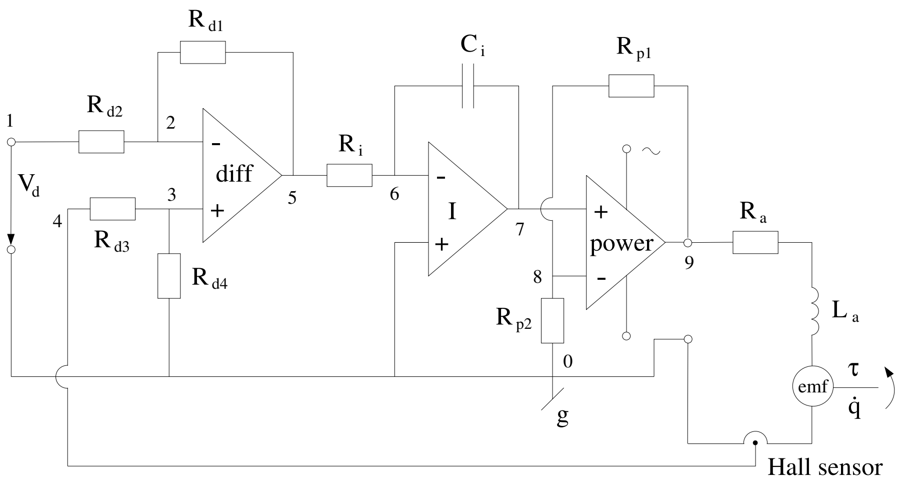

Figure 3: Model structure of Manutec r3 motors

The Dymola model reflects directly the underlying electrical circuit. Dymola supports a variety of different formats to define the connection of components. A statement of the form "**connect** *Rd1* **at** *(n2,n5)*" means, that object *Rd1* is placed between the two nodes *n2, n5*. In the same way, the other classes *R3control, R3drive1* and *R3drive2* are defined, but they are omitted here due to space limitations. Class *R3mbs*, to describe the mechanical (multibody) part of the robot, is discussed after deriving the Dymola class library for multibody systems in the next sections.

3. Object-oriented modeling of multibody systems

In this section an overview about the Dymola classes for mechanical systems are given by an example. The object-oriented view is based on ideas of [25]. In Figure 4, three different views of a double pendulum with an attached spring are shown: a mechanical, an object-oriented and a bond graph view, respectively. The object-oriented model can be directly mapped to the following Dymola model:

```
model DoublePendulum
    submodel (Inertial)  i   (ng3=1, g=9.81)
    submodel (Revolute)  r1  (n2=1)   , r2 (n2=1)
    submodel (Bar)       b1  (r3=0.5), b2 (r3=0.4)
    submodel (Bar)       b3  (r3=1.3)
    submodel (Body)      m1  (m=1, r3=0.25)
    submodel (Body)      m2  (m=1, r3=0.2)
    submodel (Spring)    s   (c=1, s0=0.5)

    connect i to r1 to b1 to r2 to b2,  i to b3,
            r1 to m1,  r2 to m2,  s at (b3:b,b2:b)
end
```

The object-oriented model consists of several basic mechanical components that are rigidly connected to each other. For example, the inertial system *"i"* is connected to revolute joint *"r1"*, which in turn is attached to massless bar *"b1"* and body *"m1"*. Again, all objects are first instantiated with **submodel** statements and afterwards connected to each other with the **connect** statement according to the physical coupling. All object parameters, which have to be entered relative to a specific coordinate system, are defined in the home position of the multibody system with respect

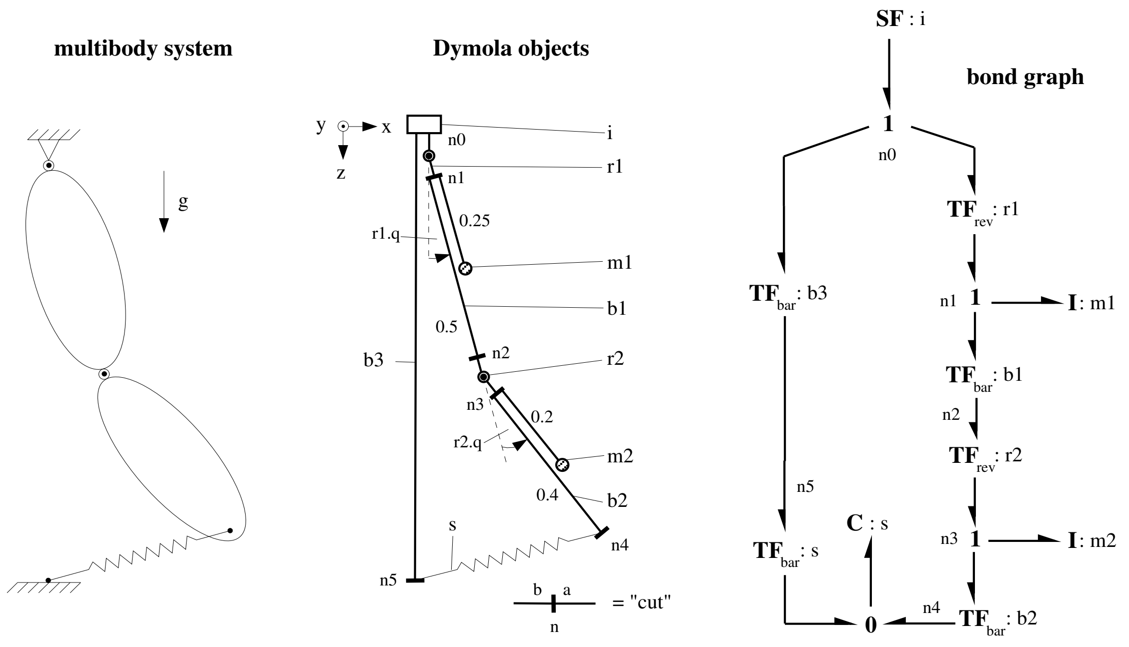

Figure 4: Mechanical, object-oriented, and bond graph view of a double pendulum.

to the inertial system. For example, to define a massless bar, the position vector from one end of the bar to the other has to be specified. Therefore the statement "**submodel** (*Bar*) *b2* (*r3=0.4*)" defines a massless bar of length 0.4m, located in the direction of the z-axis of the inertial system, when all joint coordinates are zero (= home position, i.e., the double pendulum is hanging in downward position).

4. Basic classes of multibody library

In order to formulate the equations of mechanical elements, some definitions are necessary: Matrix $\mathbf{E}$ is the identity matrix. The coordinates of a vector $\vec{h}^i$, which is resolved in frame j, is given as a column vector ${}^j\mathbf{h}^i$. If $i = j$, index j can be removed, i.e., $\mathbf{h}^i$ is the vector $\vec{h}^i$ resolved in frame i. Additionally, operator **skew** and its inverse **vec** are defined as ($\mathbf{h}$ is a (3×1) column vector, $\mathbf{H}$ is a skew-symmetric (3×3) matrix):

$$\mathbf{H} = \mathbf{skew}(\mathbf{h}) = \mathbf{skew}(\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}) = \begin{bmatrix} 0 & -h_3 & h_2 \\ h_3 & 0 & -h_1 \\ -h_2 & h_1 & 0 \end{bmatrix} \quad ; \quad \mathbf{h} = \mathbf{vec}(\mathbf{H})$$

Every basic mechanical component has at least one or two interfaces to connect the element rigidly to other mechanical elements. In Figure 5, the different views of such a mechanical interface, or cut, are shown. The cut is named a and contains a local coordinate system, also called frame, which is rigidly attached to the cut-plane. The orientation of the cut-frame is (automatically) defined in such a way that the frame is located in parallel to the inertial frame, if the multibody system is in its home position. Therefore, the cut-frames of all cuts connected at the same point coincide always with each other.

The movement of cut-frame a is uniquely described by the direction cosine matrix ${}^0\mathbf{T}^a$ transforming tensors resolved in frame a into the inertial frame, and the absolute position vector ${}^0\mathbf{r}^a$, which

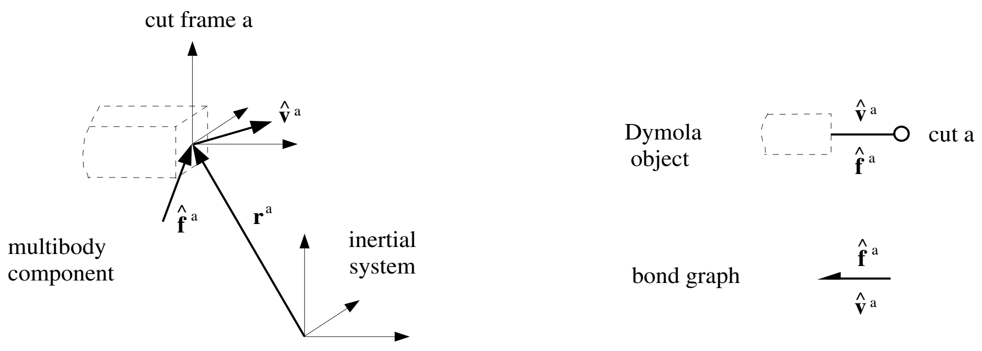

Figure 5: Definition of a mechanical cut.

is a vector pointing from the origin of the inertial frame to the origin of cut-frame a. Equivalently, the movement on velocity level is described by the absolute angular velocity $\boldsymbol{\omega}^a$ of frame a and the absolute linear velocity $\mathbf{v}^a$ of the origin of this frame, and the movement on acceleration level is described by the absolute angular acceleration $\boldsymbol{\alpha}^a$ and the absolute linear acceleration $\mathbf{a}^a$. The variables on velocity and on acceleration level are formally defined as (see e.g. [29]):

$$\begin{bmatrix} \boldsymbol{\omega}^a \\ \mathbf{v}^a \end{bmatrix} = \begin{bmatrix} \mathbf{vec}\left({}^0\mathbf{T}^{a^T}\, {}^0\dot{\mathbf{T}}^a\right) \\ {}^0\mathbf{T}^{a^T}\, {}^0\dot{\mathbf{r}}^a \end{bmatrix} , \qquad \begin{bmatrix} \boldsymbol{\alpha}^a \\ \mathbf{a}^a \end{bmatrix} = \begin{bmatrix} {}^0\mathbf{T}^{a^T}\, {}^0\dot{\boldsymbol{\omega}}^a \\ {}^0\mathbf{T}^{a^T}\, {}^0\dot{\mathbf{v}}^a \end{bmatrix} , \tag{1}$$

where ${}^0\boldsymbol{\omega}^a = {}^0\mathbf{T}^a\, \boldsymbol{\omega}^a$, ${}^0\mathbf{v}^a = {}^0\mathbf{T}^a\, \mathbf{v}^a$.

The forces and torques acting in the cut-plane are described by a resultant cut-force $\mathbf{f}^a$ and a resultant cut-torque $\boldsymbol{\tau}^a$. For notational convenience, these two quantities are concatenated into a *generalized force vector* $\hat{\mathbf{f}}^a$, and equally are the angular and linear velocity as well as the angular and linear acceleration concatenated:

$$\hat{\mathbf{v}}^a = \begin{bmatrix} \boldsymbol{\omega}^a \\ \mathbf{v}^a \end{bmatrix} ; \qquad \hat{\mathbf{a}}^a = \begin{bmatrix} \boldsymbol{\alpha}^a \\ \mathbf{a}^a \end{bmatrix} ; \qquad \hat{\mathbf{f}}^a = \begin{bmatrix} \boldsymbol{\tau}^a \\ \mathbf{f}^a \end{bmatrix} .$$

The "effect" of a mechanical cut is therefore completely described by ${}^0\mathbf{T}^a, {}^0\mathbf{r}^a, \hat{\mathbf{v}}^a, \hat{\mathbf{a}}^a, \hat{\mathbf{f}}^a$. When several mechanical elements are connected at the same point, the kinematic quantities are equal to each other, i.e., they are *across* variables. On the other hand, the cut-forces and cut-torques sum up to zero, due to the *actio=reactio* principle, i.e., they are *through* variables.

Mechanical cuts are defined in the superclasses *MbsOneCut* and *MbsTwoCut*, which are used to define the common properties of mechanical elements with one and with two mechanical cuts. For example, *MbsTwoCut* has the following class description:

```
model class MbsTwoCut
          cut   a   ( Ta(3,3), ra0(3), va(6), aa(6) / fa(6) )
          cut   b   ( Tb(3,3), rb0(3), vb(6), ab(6) / fb(6) )
    main cut   mc [a, b]
    main path  mp < a − b >

    terminal Pa, Pb

    Pa = trans(fa) * va
    Pb = trans(fb) * vb
end
```

The terminal variables Pa, Pb are the energy flowing *into* the corresponding cuts.

The absolute position vector ${}^0\mathbf{r}^a$ is resolved in the inertial frame, whereas all the other variables are resolved in cut-frame a. The reason for this is that the position vector is e.g. needed in kinematic analysis problems or to determine animation data. In such problems, the vector has to be resolved in the inertial frame. On the other hand, velocities, accelerations and forces are utilized in dynamic analysis problems. For the most important of these applications, i.e., the direct dynamics problem, a detailed analysis of a certain class of multibody systems described in [15] unveils that the O(n) algorithm yields more efficient code when the calculation is performed in body-fixed coordinate systems rather than in the inertial system. Furthermore, it is more natural to resolve forces in body-fixed frames since the directions of the forces can be associated with geometric properties of the corresponding component. Using the multibody library, it is of course easily feasible to resolve every vector in any desired coordinate system since the absolute direction cosine matrix of every frame is determined[3].

In the lower right part of Figure 5, the bond graph description of a cut is shown. Usually, bonds transmit only effort and flow variables. This is not the case in the multibody bond graph. Here, the position variables (= integral of the flow variables) and the acceleration variables (= derivative of the flow variables) are also transmitted. There are two reasons for this decision: First of all, the integral of the flow variables does not always exist, since in a general three-dimensional movement, the angular velocity is not integrable to a position coordinate[4]. Secondly, nearly all multibody systems represent higher-index differential algebraic equations (DAE), if the connection of bond graph elements is done on velocity level only. The explicit usage of the additional position and acceleration variables can be viewed as an application of the general "dummy derivative" technique to reduce the index of a DAE [5, 21]. Note that, for notational simplicity, the additional variables are not shown explicitly on the bonds of Figure 5 and of the following Figures.

A mechanical component, like a joint, a force, or a sensor, has the property, that the element has two mechanical cuts and that the relative quantities between the two cuts are needed, in order to formulate the (local) equations of the element. Therefore, a common superclass *Interact* is introduced as subclass of *MbsTwoCut*, to define these common properties only once. In Figure 6, the different views of an object of class *Interact* are shown. The relative quantities of class *Interact* are defined in the following way (see also Figure 6):

${}^b\mathbf{T}^a$	Relative direction cosine matrix from cut-frame a to cut-frame b.
${}^a\mathbf{r}^{ab}$	Relative position vector from the origin of cut-frame a to the origin of cut-frame b, resolved in cut-frame a.
${}^a\hat{\mathbf{v}}^{ab}$	Relative angular and relative linear velocity, resolved in cut-frame a:

$$ {}^a\hat{\mathbf{v}}^{ab} = \begin{bmatrix} {}^a\boldsymbol{\omega}^{ab} \\ {}^a\mathbf{v}^{ab} \end{bmatrix} = \begin{bmatrix} \mathbf{vec}\left({}^b\dot{\mathbf{T}}^{a^T}\, {}^b\mathbf{T}^a\right) \\ {}^a\dot{\mathbf{r}}^{ab} \end{bmatrix} \tag{2} $$

${}^a\hat{\mathbf{a}}^{ab}$ Relative angular and relative linear acceleration, resolved in cut-frame a:

$$ {}^a\hat{\mathbf{a}}^{ab} = \begin{bmatrix} {}^a\boldsymbol{\alpha}^{ab} \\ {}^a\mathbf{a}^{ab} \end{bmatrix} = \begin{bmatrix} {}^a\dot{\boldsymbol{\omega}}^{ab} \\ {}^a\dot{\mathbf{v}}^{ab} \end{bmatrix} . \tag{3} $$

[3]The Dymola translator has an option that *all* equations are removed which are not needed to compute the derivative of the state vector and the output variables. Therefore, equations to compute e.g. ${}^0\mathbf{T}^a$ do only show up in the generated code, if ${}^0\mathbf{T}^a$ is really needed.

[4]Instead of the integral of the "critical" flow variable (= angular velocity ω^a), the closely related direction cosine matrix ${}^0\mathbf{T}^a$ is used ($\omega^a = \mathbf{vec}({}^0\mathbf{T}^{a^T}\, {}^0\dot{\mathbf{T}}^a)$).

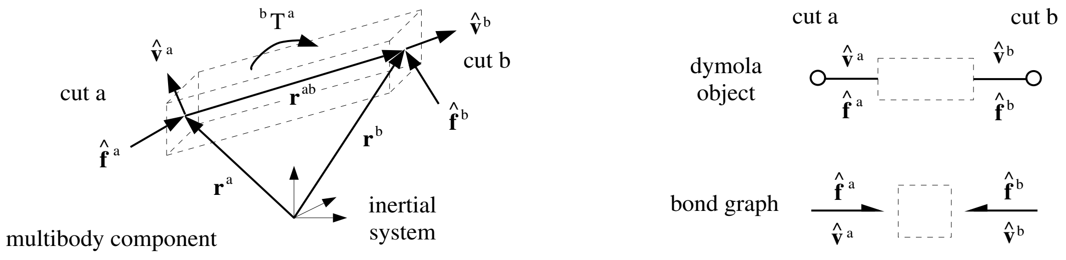

Figure 6: The different views of an object of class *Interact.*

The relative quantities of an Interact-object may be needed both in cut-frame a as well as in cut-frame b. The relationships between the relative and the absolute quantities are derived from the defining equations (1,2,3) and are given by the following equations (the derivation of the equations is shown in the appendix):

$$\begin{aligned}
{}^b\mathbf{T}^a &= {}^0\mathbf{T}^{b^T}\,{}^0\mathbf{T}^a && \text{(4a)}\\
{}^a\mathbf{r}^{ab} &= {}^0\mathbf{T}^{a^T}\,({}^0\mathbf{r}^b - {}^0\mathbf{r}^a) && \text{(4b)}\\
{}^a\hat{\mathbf{v}}^{ab} &= {}^b\hat{\mathbf{T}}^{a^T}\,{}^b\hat{\mathbf{v}}^{ab} && \text{(4c)}\\
{}^a\hat{\mathbf{a}}^{ab} &= {}^b\hat{\mathbf{T}}^{a^T}\,{}^b\hat{\mathbf{a}}^{ab} && \text{(4d)}
\end{aligned}$$

$$\begin{aligned}
{}^b\mathbf{r}^{ab} &= {}^0\mathbf{T}^{b^T}\,({}^0\mathbf{r}^b - {}^0\mathbf{r}^a) && \text{(4e)}\\
{}^b\hat{\mathbf{v}}^{ab} &= \hat{\mathbf{v}}^b - \mathbf{C}\hat{\mathbf{v}}^a && \text{(4f)}\\
{}^b\hat{\mathbf{a}}^{ab} &= \hat{\mathbf{a}}^b - \mathbf{C}\hat{\mathbf{a}}^a - \boldsymbol{\xi} && \text{(4g)}\\
\mathbf{0} &= \hat{\mathbf{f}}^a + \mathbf{C}^T\,\hat{\mathbf{f}}^b && \text{(4h)}
\end{aligned}$$

where

$$\begin{aligned}
{}^b\hat{\mathbf{T}}^a &= \begin{bmatrix} {}^b\mathbf{T}^a & \mathbf{0} \\ \mathbf{0} & {}^b\mathbf{T}^a \end{bmatrix} && \text{(5a)}\\
\mathbf{C} &= \begin{bmatrix} {}^b\mathbf{T}^a & \mathbf{0} \\ \mathbf{0} & {}^b\mathbf{T}^a \end{bmatrix} \begin{bmatrix} \mathbf{E} & \mathbf{0} \\ -\mathbf{skew}({}^a\mathbf{r}^{ab}) & \mathbf{E} \end{bmatrix} && \text{(5b)}\\
\boldsymbol{\xi} &= \begin{bmatrix} {}^b\mathbf{T}^a & \mathbf{0} \\ \mathbf{0} & {}^b\mathbf{T}^a \end{bmatrix} \begin{bmatrix} \boldsymbol{\omega}^a \times {}^a\boldsymbol{\omega}^{ab} \\ \boldsymbol{\omega}^a \times (\boldsymbol{\omega}^a \times {}^a\mathbf{r}^{ab} + 2\,{}^a\mathbf{v}^{ab}) \end{bmatrix}. && \text{(5c)}
\end{aligned}$$

Equations (4a–4g) define, that the relative quantities are given by the "difference" of the absolute quantities of cut b and of cut a. (4h) results from a force/torque balance under the assumption, that the element has no mass and no inertia.

Ideal joints can be easily derived as subclasses from class *Interact.* Joints are Interact objects, where the relative quantities are no longer independent from each other, but are functions of n generalized coordinates $\mathbf{q} = [q_1, q_2, \ldots, q_n]$. Here, n is the number of degrees of freedom of the joint ($0 \le n \le 6$). In Figure 7, the different views of a joint object are shown. In order to define a specific joint, the relative quantities must be given as functions of the generalized joint coordinates:

$${}^b\mathbf{T}^a = {}^b\mathbf{T}^a(\mathbf{q})\,, \quad {}^b\mathbf{r}^{ab} = {}^b\mathbf{r}^{ab}(\mathbf{q})\,, \quad {}^b\hat{\mathbf{v}}^{ab} = \boldsymbol{\Phi}(\mathbf{q})\dot{\mathbf{q}}\,, \quad {}^b\hat{\mathbf{a}}^{ab} = \boldsymbol{\Phi}(\mathbf{q})\ddot{\mathbf{q}} + \boldsymbol{\zeta}(\mathbf{q},\dot{\mathbf{q}})\,. \tag{6}$$

E.g. for a revolute joint, equations (6) are given by (see e.g. [29]):

$${}^b\mathbf{T}^a = \mathbf{n}\mathbf{n}^T + (\mathbf{E} - \mathbf{n}\mathbf{n}^T)\cdot\cos(q) - \mathbf{skew}(\mathbf{n})\cdot\sin(q)\,, \quad {}^b\mathbf{r}^{ab} = \mathbf{0}\,, \quad \boldsymbol{\Phi} = [\mathbf{n}; 0]\,, \quad \boldsymbol{\zeta} = \mathbf{0}\,,$$

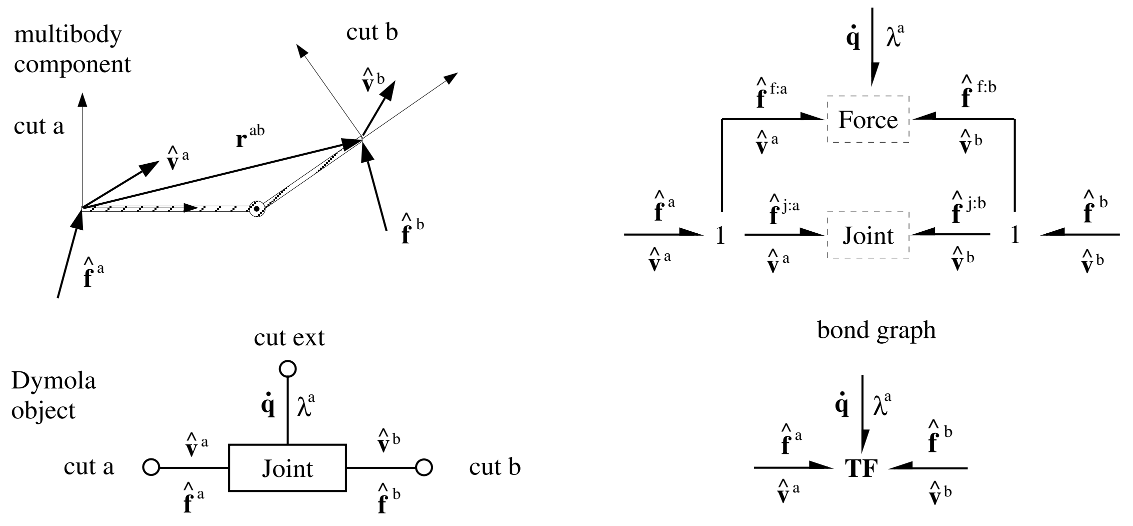

Figure 7: The different views of an object of class *Joint*.

where $\mathbf{n}$ is a unit vector in direction of the axis of rotation and q is the angle of rotation.

It is often the case that a force element acts between the two cuts of a joint. In order to simplify the treatment of such elements, a joint object consists of an ideal joint element and an optional force element, see upper right part of Figure 7. For this, an additional cut *ext* is needed, where the power of the force element flows into the joint element. Variable $\boldsymbol{\lambda}^a$ of cut *ext* is the generalized applied force of the force element, e.g. the torque along the axis of rotation of a revolute joint.

Due to the reduced possibilities of movement, additional relationships exist between the forces and torques at the two cuts of a joint. The corresponding equations can be determined from a power balance of the three cuts of a joint, because no energy is stored in an ideal joint. With (6,4f,4h) follows:

$$\begin{aligned} \sum P_i \;=\; 0 \;&=\; \hat{\mathbf{f}}^{a^T}\hat{\mathbf{v}}^a + \hat{\mathbf{f}}^{b^T}\hat{\mathbf{v}}^b + \boldsymbol{\lambda}^{a^T}\dot{\mathbf{q}} \\ &=\; (-\hat{\mathbf{f}}^{b^T}\mathbf{C})\hat{\mathbf{v}}^a + \hat{\mathbf{f}}^{b^T}(\mathbf{C}\hat{\mathbf{v}}^a + \boldsymbol{\Phi}\dot{\mathbf{q}}) + \boldsymbol{\lambda}^{a^T}\dot{\mathbf{q}} \\ &=\; (\boldsymbol{\lambda}^a + \boldsymbol{\Phi}^T\hat{\mathbf{f}}^b)^T\dot{\mathbf{q}}\ . \end{aligned} \tag{7}$$

Since $\mathbf{q}$ are the minimal coordinates of the joint, the variables $\dot{\mathbf{q}}$ are independent from each other and can assume any value. Therefore, the term in parantheses must vanish on its own, in order to satisfy the power balance constraint. Note that this requirement is equivalent to d'Alembert's principle.

Optionally, a joint may have a variable structure in the sense that the number of degrees of freedom (= dof) may vary during simulation. If a joint coordinate q_i is *locked*, the coordinate remains fixed and its derivatives remain zero ($q_i = const, \dot{q}_i = 0, \ddot{q}_i = 0$). In such a situation, an additional constraint force λ_i^l has to be applied that acts in the same direction as the generalized applied force λ_i^a.

To summarize, a general joint object is described by the following equations, in addition to equations (6):

$$\mathbf{0} = \boldsymbol{\lambda}^a + \mathbf{L}\boldsymbol{\lambda}^l + \boldsymbol{\Phi}^T\hat{\mathbf{f}}^b \tag{8a}$$

$$\mathbf{0} = \mathbf{L}\ddot{\mathbf{q}} + (\mathbf{E} - \mathbf{L})\boldsymbol{\lambda}^l \tag{8b}$$

where

$$\mathbf{L} = diag(L_{ii})\,; \quad L_{ii} = \textbf{if } Locked_i \textbf{ then } 1 \textbf{ else } 0 \ . \tag{9}$$

The two possible configurations of a joint (free/locked) are defined by the diagonal matrix **L**. If the j-th diagonal element of **L** is one, the j-th dof is locked. If it is zero, the j-th dof applies. Therefore, equation (8b) states that $\ddot{\mathbf{q}} = \mathbf{0}$, if all degrees of freedom are locked, and that otherwise the constraint forces $\boldsymbol{\lambda}^l = \mathbf{0}$. During integration, matrix **L** remains constant. The matrix may change its value only before the integration starts or immediately after an *event* has occurred.

Dymola has powerful, yet simple, language constructs do define e.g. the value of the boolean matrix **L** as a function of other variables (e.g. L_{ii} becomes one, if the generalized coordinate $\dot{q}_i$ vanishes). Dymola converts such descriptions automatically into appropriate state or time events in the generated code, see [9] for details. In [26] an example of a robot with 6 dof is given, where Coulomb friction is present in every joint. Here, the above equations are used and matrices **L** are defined as functions of $\dot{\mathbf{q}}, \boldsymbol{\lambda}^l$ in accordance to the Coulomb friction model.

Class *Joint* allows a rather general description of joints. Also, *rheonomic* joints are included in the description, although the quantities of (6) do not explicitly depend on time. This is due to the fact that it is *not* defined what is known and what is unknown. For example, an object of class *Revolute* is used as revolute joint if $\ddot{q}$ is *unknown* whereas $q, \dot{q}, \lambda^a$ are known. On the other hand, an object of the same class is used as pure rheonomic joint if $q, \dot{q}, \ddot{q}$ are known functions of time, whereas λ^a is unknown. The possibility to use the same class description for problems with different causality requirements is a unique feature of Dymola and one of its most important strengths.

In order to be able to describe general multibody systems, a few additional simple classes are needed, as shown in Figure 8. Class *Body* defines the mass and inertia properties of a rigid body. An object of this class has only one *cut* and can be connected to every other mechanical component with a "mechanical" cut, especially to one of the cuts of an Interact object. An object of class *Body* is described by the following equations (see, for example [29]):

$$\hat{\mathbf{f}}^a = \mathbf{I}^a \hat{\mathbf{a}}^a + \mathbf{b}^a \tag{10}$$

where

$$\mathbf{I}^a = \begin{bmatrix} \bar{\mathbf{I}}^a & m \cdot \mathbf{skew}({}^a\mathbf{r}^{aCM}) \\ -m \cdot \mathbf{skew}({}^a\mathbf{r}^{aCM}) & m \cdot \mathbf{E} \end{bmatrix} \tag{11a}$$

$$\mathbf{b}^a = \begin{bmatrix} \boldsymbol{\omega}^a \times \bar{\mathbf{I}}^a \boldsymbol{\omega}^a \\ m \cdot \boldsymbol{\omega}^a \times (\boldsymbol{\omega}^a \times {}^a\mathbf{r}^{aCM}) \end{bmatrix} \tag{11b}$$

$$\bar{\mathbf{I}}^a = {}^a\mathbf{I}^{CM} - m \cdot \mathbf{skew}({}^a\mathbf{r}^{aCM}) \cdot \mathbf{skew}({}^a\mathbf{r}^{aCM}) \ . \tag{11c}$$

Here, ${}^a\mathbf{r}^{aCM}$ is the position vector from the origin of cut-frame a to the center of mass of the body. ${}^a\mathbf{I}^{CM}$ is the inertia tensor of the body, relative to the center of mass, resolved in a frame that is in parallel to cut-frame a. Equation 10 incorporates the Newton-Euler equations, i.e., it states that the derivatives of the generalized momenta of a body are equal to the resultant generalized forces $\hat{\mathbf{f}}^a$ acting at the reference point used for the momentum balance. In Figure 8, the corresponding bond graph, i.e., the (slightly generalized) bond graph inertia element **I**, is shown.

Finally, class *Force* (cf. Figure 8) describes a force element. Similarily to class *Joint*, class *Force* is derived by inheritance from class *Interact* and contains the common properties of all force elements. A specific force element, such as a spring, is a subclass of class *Force*. The relative kinematic quantities between cut a and cut b defined in class *Interact*, usually serve to formulate the force law of the force element.

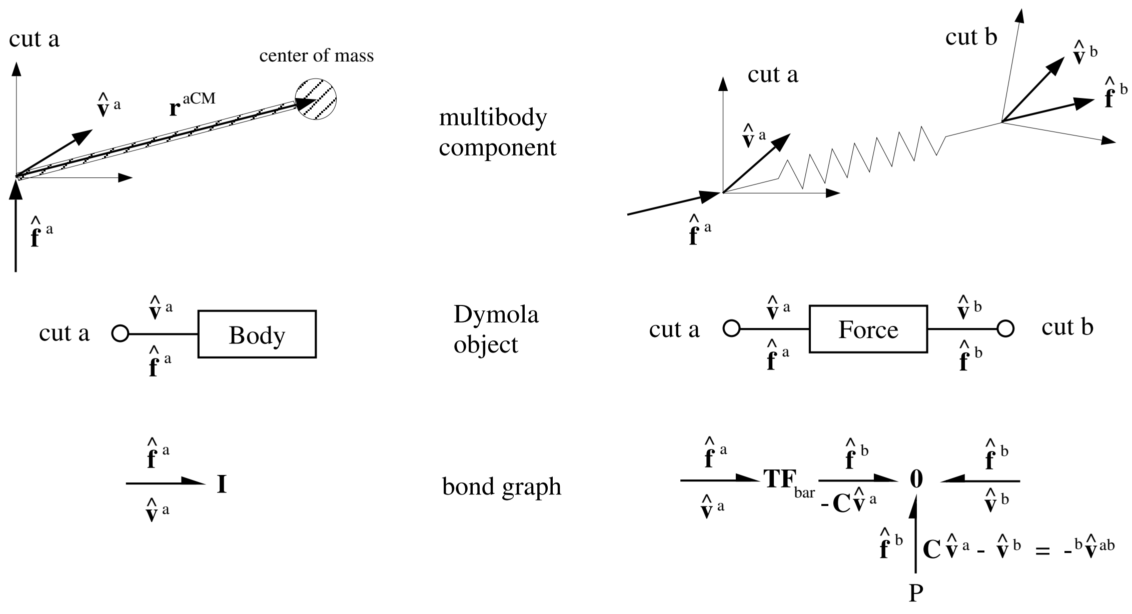

Figure 8: The different views of objects of classes *Body* and *Force*

In the multibody library of Dymola, some additional classes are defined, e.g. class *Sensor* to measure kinematic quantities between two cut-frames. Since these classes are not important in the context of this paper, they are not described here.

5. Solution of different problem formulations

The classes introduced so far are sufficient to describe rigid multibody systems in tree-structure and with kinematic loops[5]. Up to this stage, only relationships between variables have been defined. In order to arrive at a well defined mathematical formulation, it must be defined, which variables are known and which are unknown. By default, Dymola assumes that a simulation problem is under consideration, i.e., that the highest derivatives are unknown and have to be solved for. However, it is easy to modify this default behavior, e.g., for inverse problems or stationary point calculations.

To summarize, for a given object-oriented model, Dymola instantiates the equations for every involved object from the class library and adds the equations for the object connections. The problem formulation determines whether a variable is known or unknown. As a result, a (usually large) set of linear and nonlinear equations is obtained that must be solved for the unknown variables. Dymola uses efficient graph theoretical algorithms to sort the equations and variables in order to arrive at a set of equations that can be evaluated in a sequential manner [6]. Especially, algebraic loops of minimal dimensions are determined with the algorithm of Tarjan [30]. The application of this scheme to some problem formulations for mechanical systems is discussed in more detail below.

First of all, the *inverse dynamics problem* of *tree-structured* mechanical systems is considered. Here, the generalized coordinates $\mathbf{q}$ and their first and second derivatives $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$ of all joints are known, whereas all other quantities, and in particular the generalized applied forces $\boldsymbol{\lambda}^a$ of all joints, are

[5]For systems with kinematic loops, specific types of joints are provided to "cut" the loops into a tree-structure.

unknown. This problem formulation is especially encountered in robotics applications. It turns out that Dymola can always sort the equations in such a way, that all unknown variables are calculated from the known variables without encountering any algebraic loops. It can be shown that the Dymola generated code of the sorted equations is equivalent to the O(n) algorithm of Luh/Walker/Paul [19], where n is the total number of degrees of freedom, see [26].

For the *simulation problem*, also called direct dynamics problem, of *tree-structured* mechanical systems, all the generalized joint coordinates $\mathbf{q}$ and their first derivatives $\dot{\mathbf{q}}$ are used as state variables and are therefore known. Furthermore, the generalized applied forces $\boldsymbol{\lambda}^a$ are known too. All other quantities are unkown, in particular the second derivatives of the state variables $\ddot{\mathbf{q}}$. It turns out that Dymola always ends up with one huge, sparse, linear system of equations, containing as unknown variables the accelerations, cut-forces, and cut-torques of every mechanical cut, as well as the generalized accelerations $\ddot{\mathbf{q}}$ and several auxiliary quantities. For example, an object-oriented model of a typical robot with 6 revolute joints leads to one sparse, linear system of equations with about 600 equations. Dymola can transform such a huge system of equations to a small system of equations by a method called *tearing*, see [10] for details. In particular, the transformation leads to:

$$\begin{bmatrix} \mathbf{M}(\mathbf{q}) & -\mathbf{L} \\ -\mathbf{L} & \mathbf{L}-\mathbf{E} \end{bmatrix} \begin{bmatrix} \ddot{\mathbf{q}} \\ \boldsymbol{\lambda}^l \end{bmatrix} = \begin{bmatrix} \boldsymbol{\lambda}^a + \mathbf{h}(\mathbf{q},\dot{\mathbf{q}}) \\ \mathbf{0} \end{bmatrix}, \tag{12}$$

where $\mathbf{q}$ are the generalized coordinates of all joints, $\boldsymbol{\lambda}^a$ are the generalized forces of all joints, $\boldsymbol{\lambda}^l$ are the generalized constraint forces of all joints that can be locked, and matrix $\mathbf{L}$ is a diagonal matrix containing all the $\mathbf{L}$ matrices from the corresponding joints. (12) reduces to the well-known standard form (see e.g. [29]), if joints cannot be locked, i.e., when $\boldsymbol{\lambda}^l$ has dimension zero. E.g. for the mentioned robot, tearing reduces the number of equations from 600 to 6. It can be shown that the generated code is equivalent to algorithm 1 of Walker/Orin [31], i.e., $O(n^2)$ operations are needed to calculate (12), and another $O(n^3)$ operations are needed to solve (12) for $\ddot{\mathbf{q}}$, see [26]. Dymola automatically generates appropriate code in order to solve linear systems of equations, such as (12), either symbolically with an intelligent variant of Cramer's rule or numerically with a LINPACK routine.

For the *simulation problem* of mechanical systems with *kinematic loops*, the user has to select three types of joints when defining the Dymola model: *cut-joints, state-variable tree-joints* and *remaining tree-joints*. The removal of the *cut-joints* must result in a tree-structured system, called spanning tree. The joints of the spanning tree are called *tree-joints*. The generalized positional and velocity coordinates of the *state-variable tree-joints* are used as state variables $\mathbf{q}_{min}, \dot{\mathbf{q}}_{min}$, whereas the corresponding coordinates $\mathbf{q}_{rest}, \dot{\mathbf{q}}_{rest}$ of the *remaining tree-joints* are treated as unknown quantities. In a similar way as for tree-structured systems, huge linear and nonlinear systems of equations are present in the sorted Dymola equations. Again, appropriate tearing transforms these systems of equations into much smaller ones, see [10] for details. In particular, the following set of systems of equations appears as an intermediate step (assuming that no joints can be locked):

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} = \mathbf{h}(\mathbf{q},\dot{\mathbf{q}}) + \boldsymbol{\lambda}^a + \mathbf{G}^T(\mathbf{q})\,\boldsymbol{\lambda}^c \tag{13a}$$

$$\mathbf{0} = \mathbf{g} = \mathbf{g}(\mathbf{q}) \qquad (\mathbf{G} = \frac{\partial \mathbf{g}}{\partial \mathbf{q}}) \tag{13b}$$

$$\mathbf{0} = \dot{\mathbf{g}} = \mathbf{G}(\mathbf{q})\dot{\mathbf{q}} \tag{13c}$$

$$\mathbf{0} = \ddot{\mathbf{g}} = \mathbf{G}(\mathbf{q})\ddot{\mathbf{q}} + \frac{\partial \mathbf{G}(\mathbf{q})\dot{\mathbf{q}}}{\partial \mathbf{q}}\dot{\mathbf{q}}\;. \tag{13d}$$

Here, (13a) are the equations of motion of the spanning tree mechanism, (13b,13c,13d) are the constraint equations of all cut joints on position, velocity, and acceleration levels, respectively, $\mathbf{q}$ are the relative coordinates of the joints of the spanning tree, $\boldsymbol{\lambda}^a$ are the (known) generalized applied forces of the joints of the spanning tree, and $\boldsymbol{\lambda}^c$ are the (unknown) generalized constraint forces of the cut-joints.

Dymola transforms (13) into at least three sets of systems of equations: A non-linear system of equations for $\mathbf{q}_{rest}$ (13b), which is solved with a numerical non-linear equation solver, a linear system of equations for $\dot{\mathbf{q}}_{rest}$ (13c), and a linear system of equations for $\ddot{\mathbf{q}}_{min}, \ddot{\mathbf{q}}_{rest}, \boldsymbol{\lambda}^c$ (13a,13d). Depending on the loop structure of the mechanism, the systems of equations may be further divided into smaller subsystems, which can be solved independently from each other.

6. Object-oriented O(n) solution of the direct dynamics problem

In this section, an alternative approach is explained for the simulation problem. Here, a second class library called *mbssim.lib* is provided, which has essentially the same interfaces as the class library *mbs.lib* explained in the last sections. A multibody system is defined in *exactly* the same way as before. However, the equations of the class-library *mbs.lib* are transformed in such a way that Dymola will produce very efficient code for tree-structured multibody systems (O(n) algorithm). The new library *mbssim.lib* can only be used for the solution of the direct dynamics problem, contrary to library *mbs.lib*, due to a built-in causality.

Equations (4,6,8) form the starting point for reformulating the class library. The reformulation is based on the property that all cut-forces and cut-torques $\hat{\mathbf{f}}$ can be expressed as linear functions of the absolute linear and angular acceleration $\hat{\mathbf{a}}$ *at the same cut*, i.e.,

$$\hat{\mathbf{f}} = \mathbf{I}\hat{\mathbf{a}} + \mathbf{b} \;; \qquad \mathbf{I} = \mathbf{I}^T \;. \tag{14}$$

$\mathbf{I}$ and $\mathbf{b}$ are functions of positional and velocity coordinates, but not of acceleration variables. (14) is true for classes *Body* and *Force*, since the corresponding equation (10) of a body object is already in this form, and since the equation for a force object is a special case with $\mathbf{I} = \mathbf{0}$, because a force law does not depend on accelerations. What remains to be shown is that this property also holds for class *Joint*. For the proof of this statement, assume first that equation (14) holds true for cut b of an object of class *Joint*, i.e, $\hat{\mathbf{f}}^b = \mathbf{I}^b\hat{\mathbf{a}}^b + \mathbf{b}^b$. According to Wehage [32], equations (4g,6,8) can be combined to form the following linear, symmetric matrix equation:

$$\begin{bmatrix} \mathbf{I}^b & -\mathbf{E} & \mathbf{0} & \mathbf{0} \\ -\mathbf{E} & \mathbf{0} & \boldsymbol{\Phi} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\Phi}^T & \mathbf{0} & \mathbf{L} \\ \mathbf{0} & \mathbf{0} & \mathbf{L} & \mathbf{E}-\mathbf{L} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{a}}^b \\ \hat{\mathbf{f}}^b \\ \ddot{\mathbf{q}} \\ \boldsymbol{\lambda}^l \end{bmatrix} = \begin{bmatrix} -\mathbf{b}^b \\ -\mathbf{C}\hat{\mathbf{a}}^a - \boldsymbol{\xi} - \boldsymbol{\zeta} \\ -\boldsymbol{\lambda}^a \\ \mathbf{0} \end{bmatrix} . \tag{15}$$

Equation (15) states that the unknown variables of cut-frame b can be expressed as linear functions of the unknown acceleration $\hat{\mathbf{a}}^a$ of cut-frame a. Solving (15) explicitly for the unknown quantities of cut-frame b results in:

$$\begin{aligned} \boldsymbol{\lambda}^l &= -\mathbf{L}^*\mathbf{h} && (16a)\\ \ddot{\mathbf{q}} &= -\mathbf{M}^{-1}(\mathbf{E}-\mathbf{L}^*)\mathbf{h} && (16b)\\ \hat{\mathbf{a}}^b &= \mathbf{C}\hat{\mathbf{a}}^a + \boldsymbol{\Phi}\ddot{\mathbf{q}} + \boldsymbol{\eta} && (16c)\\ \hat{\mathbf{f}}^b &= \mathbf{I}^b\hat{\mathbf{a}}^b + \mathbf{b}^b && (16d) \end{aligned}$$

with

$$\begin{aligned} \mathbf{h} &= \left(\boldsymbol{\Phi}^T\,\mathbf{I}^b\,\mathbf{C}\right)\hat{\mathbf{a}}^a + \left(\boldsymbol{\Phi}^T\,\mathbf{I}^b\,\mathbf{C}\boldsymbol{\eta} + \boldsymbol{\lambda}^a + \boldsymbol{\Phi}^T\,\mathbf{b}^b\right) && (17a)\\ \mathbf{M} &= \boldsymbol{\Phi}^T\,\mathbf{I}^b\,\boldsymbol{\Phi} && (17b)\\ \boldsymbol{\eta} &= \boldsymbol{\xi} + \boldsymbol{\zeta} && (17c)\\ \mathbf{L}^* &= \left(\mathbf{L}\mathbf{M}^{-1}\mathbf{L} + \mathbf{L} - \mathbf{E}\right)^{-1}\mathbf{L}\mathbf{M}^{-1} \quad = \quad \mathbf{L} \ \textbf{if} \ \ dim(\mathbf{L}) = 1\times 1 \ . && (17d) \end{aligned}$$

Inserting these equations into (4h), i.e., into "$\mathbf{0} = \hat{\mathbf{f}}^a + \mathbf{C}^T \hat{\mathbf{f}}^b$", yields the required relationship (14), because $\hat{\mathbf{f}}^a$ is linear in $\hat{\mathbf{f}}^b$, $\hat{\mathbf{f}}^b$ is linear in $\hat{\mathbf{a}}^b$ due to (16d), and $\hat{\mathbf{a}}^b$ is linear in $\hat{\mathbf{a}}^a$ due to (16c,16b,17a):

$$\hat{\mathbf{f}}^a = \mathbf{I}^a \hat{\mathbf{a}}^a + \mathbf{b}^a \quad ; \qquad \mathbf{I}^a = \mathbf{I}^{a^T} \tag{18}$$

where

$$\begin{aligned}
\mathbf{I}^a &= -\mathbf{C}^T \mathbf{N}\, \mathbf{C} && \text{(19a)}\\
\mathbf{b}^a &= -\mathbf{C}^T \left(\mathbf{b}^b + \mathbf{N}\boldsymbol{\eta} - \mathbf{I}^b \boldsymbol{\Phi} \mathbf{M}^{-1}(\mathbf{E} - \mathbf{L}^*)\,(\boldsymbol{\lambda}^a + \boldsymbol{\Phi}^T \mathbf{b}^b)\right) && \text{(19b)}\\
\mathbf{N} &= \mathbf{I}^b - \mathbf{I}^b \boldsymbol{\Phi}\, \mathbf{M}^{-1}\,(\mathbf{E} - \mathbf{L}^*)\, \boldsymbol{\Phi}^T\, \mathbf{I}^b && \text{(19c)}\\
\mathbf{M} &= \boldsymbol{\Phi}^T\, \mathbf{I}^b\, \boldsymbol{\Phi} && \text{(19d)}\\
\mathbf{L}^* &= \left(\mathbf{L}\mathbf{M}^{-1}\mathbf{L} + \mathbf{L} - \mathbf{E}\right)^{-1} \mathbf{L}\mathbf{M}^{-1} \;\; = \;\; \mathbf{L} \textbf{ if } \mathit{dim}(\mathbf{L}) = 1 \times 1\ . && \text{(19e)}
\end{aligned}$$

If all objects of classes *Body* and *Force* are removed from a tree-structured mechanical system, the cut-forces and cut-torques of the removed objects, which are acting with negative sign on the remaining multibody system, can be expressed in the form of (14). The remaining system contains objects of class *Joint*, only. The cut-forces and cut-torques acting at the "leaves" of the remaining tree can be expressed in the form of (14) too, since the linear factors $\mathbf{I}$ and $\mathbf{b}$ of the cut-forces and cut-torques are through variables, and all the other cut-forces and cut-torques at the leaves are already in this form because of the Body- and Force-objects having been removed. Assuming, that cut a of every joint object is always directed "closer" to the inertial system as cut b of the same joint, the cut-forces and cut-torques at cut a of the leave-joints can be given in the desired form too, due to (18). Now, all the leave-joints are removed, and the same situation applies. This means, that by backward recursion from the "leaves" to the "root" of the tree it can be shown that all cut-forces and cut-torques can be expressed through (14). Q.E.D.

The class library *mbssim.lib* is built by using the already explained class library *mbs.lib* and by replacing equations (4g,4h,8) by equations (16–19). Furthermore, not the torque and force $\hat{\mathbf{f}}$ are propagated through cuts, but the linear factors $\mathbf{I}, \mathbf{b}$ of the torque and force in accordance with (14,18).

The main advantage of this reformulation lies in the fact that Dymola can sort the equations of a tree-structured multibody system explicitly for the generalized accelerations $\ddot{\mathbf{q}}$ without encountering algebraic loops (with the exception of small linear systems of equations within *Joint*-objects, which are solved by inverting matrix $\mathbf{M}$ of (19d)). This property can be explained as follows: The linear factors $\mathbf{I}, \mathbf{b}$ are known at *Body*- and *Force*-objects. Using equations (19a,19b), these factors are propagated through all *Joint*-objects finally arriving at the inertial system. Since the acceleration of the inertial system is known, equations (16) can be used to calculate all unknown variables of the objects that are directly attached to the inertial frame. Afterwards the accelerations of these objects are known, and therefore the unknown variables of all objects attached to them can be calculated, and so on. Of course, this feature is only valid for tree-structured multibody systems.

The number of operations in the generated code is proportional to the number of degrees of freedom ($= O(n)$) of the multibody system and is therefore very efficient, especially if n is high. In the same sense as the inverse dynamics problem solution with library *mbs.lib* is equivalent to the Luh/Walker/Paul algorithm [19], the direct dynamics problem solution with library *mbssim.lib* is equivalent to the recursive O(n) algorithm of Brandl/Johanni/Otter [1]. Note that these algorithms are *not* explicitly programmed. Instead, only local properties of objects are stored in the corresponding class libraries. Due to the built-in connection rules of Dymola together with its sorting algorithm, Dymola "reinvents" these algorithms on its own.

The above class library extends the recursive algorithms such as [1] in one important aspect: it allows the handling of variable structure systems. As already noted, matrix $\mathbf{L}$ signals whether a degree of freedom of a joint is locked or not. As can be seen, $\mathbf{L}$ appears in some places in the recursive relations (19). The occurrences of matrix $\mathbf{L}$ can easily be interpreted: Assume that all degrees of freedom of a joint are free, i.e., $\mathbf{L} = \mathbf{0}$, and therefore $\mathbf{L}^* = \mathbf{0}$. In this case, (19) are the usual recursive relations. On the other hand, if all degrees of freedom are locked, $\mathbf{L} = \mathbf{E}$ and therefore $\mathbf{L}^* = \mathbf{E}$, and equations (19) reduce to the recursive relations of a fixed joint with zero degrees of freedom. The generated equations are nearly as efficient as if matrix $\mathbf{L}$ were not present. This can easily be seen for a joint with one degree of freedom. In this case, matrices $\mathbf{M}$ and $\mathbf{L}$ are scalars, and only one division, one subtraction, and one multiplication are needed in order to calculate the additional terms $\mathbf{M}^{-1}(\mathbf{E} - \mathbf{L}), \mathbf{Lh}$, i.e., the efficiency reduction due to the variable structure is negligible.

The O(n) algorithm can also easily be generalized for multibody systems with kinematic loops. Using the same technique as in section 5, i.e., using *cut-joints*, *state-variable tree-joints*, and *remaining tree-joints*, the following equations are obtained when using library *mbssim.lib* and the tearing technique (see also equation (13)):

$$\mathbf{0} = \mathbf{g}(\mathbf{q}) \qquad (\mathbf{G} = \frac{\partial \mathbf{g}}{\partial \mathbf{q}}) \tag{20a}$$

$$\mathbf{0} = \mathbf{G}(\mathbf{q})\dot{\mathbf{q}} \tag{20b}$$

$$\mathbf{G}\mathbf{M}^{-1}\mathbf{G}^T \boldsymbol{\lambda}^c = -\mathbf{G}\mathbf{M}^{-1}(\mathbf{h}(\mathbf{q},\dot{\mathbf{q}}) + \boldsymbol{\lambda}^a) - \frac{\partial \mathbf{G}(\mathbf{q})\dot{\mathbf{q}}}{\partial \mathbf{q}}\dot{\mathbf{q}} \tag{20c}$$

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{h}(\mathbf{q},\dot{\mathbf{q}}) + \boldsymbol{\lambda}^a + \mathbf{G}^T(\mathbf{q})\boldsymbol{\lambda}^c) \tag{20d}$$

(20a) is a non-linear system of equations for $\mathbf{q}_{rest}$, (20b) is a linear system of equations for $\dot{\mathbf{q}}_{rest}$, (20c) is a linear system of equations for $\boldsymbol{\lambda}^c$, and (20d) is the O(n) algorithm for the spanning tree to calculate $\ddot{\mathbf{q}}$. Note, that $\mathbf{M}^{-1}$ is never calculated explicitly, but is implicitly contained in the O(n) algorithm. This procedure is equivalent to the algorithm of Brandl/Johanni/Otter [2] for multibody systems with kinematic loops.

7. Dymola model of mechanical part of robot

The Dymola library for multibody systems as explained in the last two sections is not provided here due to space limitations. However, to demonstrate its usage on a more complicated problem, the multibody model of the 6 dof robot of section 2 is presented:

```
model class R3mbs
    submodel (Inertial)  i(ng3=-1, g=9.81)
    submodel (Revolute)  r1(n3=1), r2(n1=1), r3(n1=1)
    submodel (Revolute)  r4(n3=1), r5(n1=1), r6(n3=1)
    submodel (Bar)       b3(r3=0.5), b5(r3=0.73), bL(r1=RL1,r2=RL2,r3=RL3)
    submodel (Body)      m1(I33=1.16)
    submodel (Body)      m2(m=56.5, r1=0.172, r3=0.205,
                            I11=2.58, I22=2.73, I33=0.64, I31=-0.46)
                    ...
    submodel (Body)      load(m=mL)
    parameter mL=0, rL1=0, rL2=0, rL3=0
    cut j1, j2, j3, j4, j5, j6

    connect i    to r1 to r2 to b3 to r3 to r4 to b5 to r5 to r6 to bL,
            m1   at r1:b,    j1 at r1:ext,
            m2   at r2:b,    j2 at r2:ext,
                     ...
            load at bL:b,    j6 at r6:ext,
end
```

Note, that the interface of the robot to the outside world is given by **cuts** *j1*, ..., *j6* of the revolute joints. At these cuts, the gear boxes can be attached as shown in section 2.

Generating the equations of motion and the inverse dynamics model for an object of the above class (i.e., the multibody part of the robot only), Dymola produces code for each of these problems within about 10 seconds[6] containing the following number of operations:

	direct problem	inverse problem
$*, /$ operations	727	269
$+, -$ operations	493	190

The numbers of operations are approximately the same as for other symbolic multibody programs.

8. Conclusions

In this paper, a new library for the object-oriented modeling language Dymola was presented that supports the modeling of general multibody systems consisting of a connection of rigid bodies, ideal joints, and force elements. The presented library generates efficient code for tree-structured multibody systems. Multibody systems with kinematic loops can also be handled. In both cases, models are transformed into state space form.

Dymola should not be viewed as just "yet another" multibody program. The unique feature of Dymola is its support for modeling components from *several different engineering disciplines* within *one* environment. For example, Dymola has been successfully applied to the modeling of the thermodynamic behavior of a house using bond graphs [33], chemical reaction systems [3], electronic circuits [14], and a detailed robot model [26]. Furthermore, Dymola has sophisticated language elements for the definition and handling of discrete-event systems based on time and state events [9]. It is therefore easy to model e.g. sampled data systems, discontinuous elements, interconnected friction elements, or impact.

References

1. Brandl H., Johanni R., and Otter M.: A very efficient algorithm for the simulation of robots and similar multibody–systems without inversion of the mass–matrix. In: Theory of Robots. Selected Papers from the IFAC/IFIP/IMACS Symposium, Vienna/Austria, December 1986, edited by P. Kopacek, I. Troch, K. Desoyer, pp. 95–100, Pergamon Press, 1988.
2. Brandl H., Johanni R., and Otter M.: An algorithm for the simulation of multibody systems with kinematic loops. Proceedings of the 7th World Congress on The Theory of Machines and Mechanisms, pp. 407–411, Pergamon Press, 1987.
3. Brooks B.A., and Cellier F.E.: Modeling of a Distillation Column Using Bond Graphs. Proceedings SCS Int. Conf. on Bond Graph Modeling. San Diego, California, pp. 315–320, January 17-20, 1993.
4. Cellier F.E.: Continuous System Modeling. Springer-Verlag, New York, 1991.
5. Cellier F.E., and Elmqvist H.: Automated Formula Manipulation Supports Object-Oriented Continuous-System Modeling. IEEE Control System Magazine 13(2), pp. 28-38, 1993.
6. Elmqvist H.: A Structured Model Language for Large Continuous Systems. Ph.D. dissertation. Report CODEN:LUTFD2/(TFRT–1015), Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1978.
7. Elmqvist H., Åström K.J., Schönthal T., and Wittenmark B.: *Simnon — User's Guide for MS-DOS Computers*, SSPA Systems, Gothenburg, Sweden, 1990.
8. Elmqvist H.: Dymola — User's Manual. Dynasim AB, Research Park Ideon, Lund, Sweden, 1994.
9. Elmqvist H., Cellier F.E., and Otter M.: Object–Oriented Modeling of Hybrid Systems. Proceedings ESS'93, European Simulation Symposium, S. xxxi-xli, Delft, The Netherlands, Oct. 1993.

[6]on an Apollo workstation with a Motorola 68040 processor

10. Elmqvist H., and Otter M.: Methods for Tearing Systems of Equations in Object-Oriented Modeling. Proceedings ESM'94, European Simulation Multiconference, Barcelona, Spain, June 1994.

11. Franke J., and Otter M.: The Manutec r3 Benchmark Models for the Dynamic Simulation of Robots. Technical Report TR R101-93, Institute for Robotics and System Dynamics, DLR Oberpfaffenhofen, March 1993.

12. Glocker C., and Pfeiffer F.: Complementary problems in multibody systems with planar friction. Accepted for publication in "Applied Mechanics", to appear.

13. Haug E.J., Wu S.C., and Yang S.M.: Dynamics of Mechanical Systems with Coulomb Friction, Stiction, Impact and Constraint Addition–Deletion – Part I, Theory. Mechanism and Machine Theory, Vol. 21, pp. 401–506, 1986.

14. Hild D.R., and Cellier F.E.: Object-Oriented Electronic Circuit Modeling Using Dymola. Proceedings OOS'94, SCS Object Oriented Simulation Conference, Tempe, Arizona, pp. 68–75, January 1994.

15. Hiller M., Kecskeméthy A., and Stelzle W.: Ein Vergleich rekursiver Verfahren. Internal report, Universität Duisburg, Fachgebiet Mechatronik & Mechanik. A short version of this paper appeared in: Stelzle W.: Ein Vergleich rekursiver Verfahren zur Untersuchung der Dynamik baumstrukturierter Mehrkörpersysteme. ZAMM 73, pp. 107 – 109, 1993.

16. Karnopp D.C., Margolis D.L. and Rosenberg R.C.: System Dynamics: A Unified Approach. John Wiley, 2^{nd} edition, 1990.

17. Korn G.A.: Interactive Dynamic-System Simulation. McGraw-Hill, 1989.

18. Lötstedt P.: Coulomb friction in two-dimensional rigid body systems. ZAMM 61, pp. 605–515, 1981.

19. Luh J.Y.S., Walker M.W., and Paul R.P.C.: On-line computational scheme for mechanical manipulators. Trans. ASME Journal of Dynamic Systems Meas. and Control, Vol. 102, pp. 69-76, 1980.

20. Mathworks Inc.: SIMULINK – User's Manual. South Natick, Mass., 1992.

21. Mattsson S.E., and Söderlind G.: A New Technique for Solving High-Index Differential-Algebraic Equations Using Dummy Derivatives. IEEE Symposium on Computer-Aided Control System Design, CACSD'92, Napa, California, March 1992.

22. Mitchell E.E.L., and Gauthier J.S.: ACSL: Advanced Continuous Simulation Language – Reference Manual. Edition 10.0, MGS, Concord., Mass., 1991.

23. Nagel L.W.: SPICE2: A computer program to simulate semiconductor circuits. Berkeley, University of California, Electronic Research Laboratory, ERL – M 520, 1975.

24. Otter M.: DSblock: A neutral description of dynamic systems. Version 3.2. Technical Report TR R81-92, Institute for Robotics and System Dynamics, DLR Oberpfaffenhofen, May 1992.

25. Otter M., Hocke M., Daberkow A., and Leister G.: An Object-Oriented Data Model for Multibody Systems. "Advanced Multibody System Dynamics", edited by W. Schiehlen, pp. 19–58, Kluwer, 1993.

26. Otter M.: Objektorientierte Modellierung mechatronischer Systeme am Beispiel geregelter Roboter. Dr.-Ing. Dissertation, submitted to Ruhr-Universität Bochum, to appear.

27. Pantelides C.C: The consistent initialization of differential-algebraic systems. SIAM Journal of Scientific and Statistical Computing, No. 9, pp. 213-231, 1988.

28. Pfeiffer F.: Über unstetige, insbesondere stoßerregte Schwingungen. Zeitschrift für Flugwissenschaft und Weltraumforschung, vol. 12, pp. 358-367, 1988.

29. Roberson R.E., and Schwertassek R.: Dynamics of Multibody Systems. Springer-Verlag, 1988.

30. Tarjan R.E.: Depth First Search and Linear Graph Algorithms. SIAM Journal of Comp., Vol. 1, pp. 146–160, 1972.

31. Walker M., and Orin D.: Efficient Dynamic Computer Simulation of Robotic Mechanisms. Journal of Dynamic Systems, Measurement and Control, Vol. 104, pp. 205–211., 1982

32. Wehage R.A.: Application of matrix partitioning and recursive projection to order n solution of constrained equations of motion. In "Proc. of the 20th Biennial ASME Mechanisms Conf.", Orlando, Florida, 1988.

33. Weiner M., and Cellier F.E.: Modeling and Simulation of a Solar Energy System by Use of Bond Graphs. Proceedings SCS Int. Conf. on Bond Graph Modeling. San Diego, California, pp. 301–306, January 17-20, 1993.

Appendix

In this section, equations (4) are derived. Equations (4a–4e) are directly given by their definition. Equation (4f) can be split up into two parts to determine ${}^b\boldsymbol{\omega}^{ab}$ and ${}^b\mathbf{v}^{ab}$, respectively:

$$\begin{aligned}
{}^b\boldsymbol{\omega}^{ab} &= {}^b\mathbf{T}^a\,{}^a\boldsymbol{\omega}^{ab} \\
&= {}^b\mathbf{T}^a\,\mathbf{vec}({}^b\dot{\mathbf{T}}^{a^T}\,{}^b\mathbf{T}^a) \qquad \text{(due to (2))} \\
&= {}^b\mathbf{T}^a\,\mathbf{vec}\left(\frac{d}{dt}\left({}^0\mathbf{T}^{b^T}\,{}^0\mathbf{T}^a\right)^T\,{}^b\mathbf{T}^a\right) \qquad \text{(due to (4a))} \\
&= {}^b\mathbf{T}^a\,\mathbf{vec}\left(\left({}^a\dot{\mathbf{T}}^0\,{}^0\mathbf{T}^b + {}^a\mathbf{T}^0\,{}^0\dot{\mathbf{T}}^b\right){}^b\mathbf{T}^a\right) \\
&= {}^b\mathbf{T}^a\,\mathbf{vec}\left(\left({}^a\dot{\mathbf{T}}^0\,{}^0\mathbf{T}^a\,{}^0\mathbf{T}^{a^T}\,{}^0\mathbf{T}^b + {}^a\mathbf{T}^0\,{}^0\mathbf{T}^b\,{}^0\mathbf{T}^{b^T}\,{}^0\dot{\mathbf{T}}^b\right){}^b\mathbf{T}^a\right) \\
&= {}^b\mathbf{T}^a\,\mathbf{vec}\left(\left(-\mathbf{skew}(\boldsymbol{\omega}^a)\,{}^a\mathbf{T}^b + {}^a\mathbf{T}^b\,\mathbf{skew}(\boldsymbol{\omega}^b)\right){}^b\mathbf{T}^a\right) \qquad \text{(due to (1))} \\
&= {}^b\mathbf{T}^a\,\mathbf{vec}\left(-\mathbf{skew}(\boldsymbol{\omega}^a) + \mathbf{skew}({}^a\boldsymbol{\omega}^b)\right) \\
&= {}^b\mathbf{T}^a\,(-\boldsymbol{\omega}^a + {}^a\boldsymbol{\omega}^b) \\
&= \boldsymbol{\omega}^b - {}^b\mathbf{T}^a\,\boldsymbol{\omega}^a
\end{aligned}$$

$$\begin{aligned}
{}^b\mathbf{v}^{ab} &= {}^b\mathbf{T}^a\,{}^a\mathbf{v}^{ab} \\
&= {}^b\mathbf{T}^a\,{}^a\dot{\mathbf{r}}^{ab} \qquad \text{(due to (2))} \\
&= {}^b\mathbf{T}^a\,\frac{d}{dt}\left({}^0\mathbf{T}^{a^T}\left({}^0\mathbf{r}^b - {}^0\mathbf{r}^a\right)\right) \qquad \text{(due to (4b))} \\
&= {}^b\mathbf{T}^a\,\left({}^0\dot{\mathbf{T}}^{a^T}\,{}^0\mathbf{T}^a\,{}^0\mathbf{T}^{a^T}\,{}^0\mathbf{r}^{ab} + {}^0\mathbf{T}^{a^T}\left({}^0\dot{\mathbf{r}}^b - {}^0\dot{\mathbf{r}}^a\right)\right) \\
&= {}^b\mathbf{T}^a\,{}^a\mathbf{T}^0\,{}^0\dot{\mathbf{r}}^b - {}^b\mathbf{T}^a\left(\mathbf{skew}(\boldsymbol{\omega}^a)\,{}^a\mathbf{r}^{ab} + \mathbf{v}^a\right) \qquad \text{(due to (1))} \\
&= \mathbf{v}^b - {}^b\mathbf{T}^a\left(-\mathbf{skew}({}^a\mathbf{r}^{ab})\,\boldsymbol{\omega}^a + \mathbf{v}^a\right)\ .
\end{aligned}$$

(4f) follows by collecting the two final equations for ${}^b\boldsymbol{\omega}^{ab}$ and for ${}^b\mathbf{v}^{ab}$ together. In a similar way, equation (4g) can be derived. Equation (4h) follows from a force/torque balance under the assumption that an *Interact*-object has no mass and no inertia. The cut-forces and cut-torques at the two cuts of an *Interact*-object are given as:

$$\hat{\mathbf{f}}^a = \begin{bmatrix} \boldsymbol{\tau}^a \\ \mathbf{f}^a \end{bmatrix}, \qquad \hat{\mathbf{f}}^b = \begin{bmatrix} \boldsymbol{\tau}^b \\ \mathbf{f}^b \end{bmatrix}.$$

Transforming the cut-force and the cut-torque of cut b into cut-frame a, carrying out a torque balance with respect to the origin of cut-frame a, and forming the force sum, results in:

$$\mathbf{0} = \begin{bmatrix} \boldsymbol{\tau}^a \\ \mathbf{f}^a \end{bmatrix} + \begin{bmatrix} {}^b\mathbf{T}^{a^T}\,\boldsymbol{\tau}^b + {}^a\mathbf{r}^{ab}\times\left({}^b\mathbf{T}^{a^T}\,\mathbf{f}^b\right) \\ {}^b\mathbf{T}^{a^T}\,\mathbf{f}^b\ . \end{bmatrix}$$

(4h) follows when the abbreviation (5b) is used.