3 Modelling and Simulation of Mixed Systems

3.1 Introduction

The majority of technical systems are mixed; i.e. they incorporate components from various fields, such as electronics, mechanics, software and other domains. This raises significant design problems because hitherto design methodologies and the associated design tools have usually been developed for a single field only. This means that the overall function of the system cannot be investigated until the prototype construction phase. However, by the time this stage is reached, changes to the design have already become very expensive and time-consuming. The consideration of virtual prototypes, which allow virtual experiments to be performed on a computer by simulation, offers an elegant solution to the problem described above.

This chapter introduces the consideration of the simulation of mixed systems by describing common ground and differences between electronics and mechanics in Section 3.2. This lays the foundation for the modelling and simulation of electromechanical systems.

This chapter also describes various approaches to the modelling of mechatronic and micromechatronic systems. One possibility is to transfer mechanical models into the form of electronic models (and vice-versa). This permits the consideration of the mechanics in a electronics simulator (and vice-versa), see Section 3.3. Thus half of the modelling problem can take place using standard methods. On the other hand, this raises the question of how to formulate electronics within the modelling world of mechanics (and vice versa). For the transformation of mechanics into a circuit simulator the use of hardware description languages is the method of choice. This approach — the main theme of this work — will be discussed in detail in Chapters 4–6. Before hardware description languages became widely accepted in recent years, equivalent circuits were often developed to describe the behaviour of mechanical components. Relatively few attempts were made to consider electronics along with mechanics in a mechanical simulator. Although some mechanical simulators permit the inclusion of simple components such as capacitors, resistors

Mechatronic Systems Georg Pelz
© 2003 John Wiley & Sons, Ltd ISBN: 0-470-84979-7

or inductances, the consideration of active components or entire circuits has hitherto only been realised in experiments. One possible reason for this is that when developing mechanical parts of the system it is often sufficient to describe the electronics in abstract form using controller equations and thereby to avoid the circuit level.

There are also some approaches that attempt to model the entire electro-mechanical system as a unit without any preference for electronics or mechanics. These methods include bond graphs, block diagrams, and modelling languages such as Modelica. Despite the elegance of these description forms it is generally found that neither the electronics nor the mechanics can be modelled with the usual standard procedures, see Section 3.4.

Finally, the possibility of coupling together simulators for different domains represents a further approach to solving the problem. This could, for example, occur systematically with the aid of a simulator backplane, as is often created for pure electronics. Typical applications for this are the coupling of circuit and logic simulators or the distribution of simulations on a parallel computer or a cluster of workstations. However, simulator coupling is associated with a whole range of problems: Firstly the resulting simulator package is unwieldy, it is often difficult to operate, and licences are required for all of the individual simulators. Secondly, the problems associated with synchronisation between two very heterogeneous simulator cores are even more severe, see Section 3.5.

At this point it should be re-emphasised that this work deals with the simulation of mixed *systems*. Electro-mechanical *components* will be considered only within the context of the system.

3.2 Electronics and Mechanics

3.2.1 Introduction

The following section will investigate the common ground and differences between electronics and mechanics and the associated models. For this purpose the modelling of the two domains will be considered on the level of an abstraction, see Figure 3.1. On the lowest level we find the consideration of electrical and magnetic fields and of the mechanical continuum. In electronics such considerations are required exclusively for the design of components, e.g. transistors, and this approach is known as device simulation. In the present context, however, we are interested in systems and therefore this type of simulation can be disregarded. Above this we find circuit simulation, which considers net lists of electronic components. In digital circuits we can convert continuous voltage levels into discrete values, such as 0 and 1, thereby significantly accelerating the simulation. Using digital electronics we can build processors on which software runs, which can itself act as an abstraction level. In mechanics, on the other hand, it has hitherto

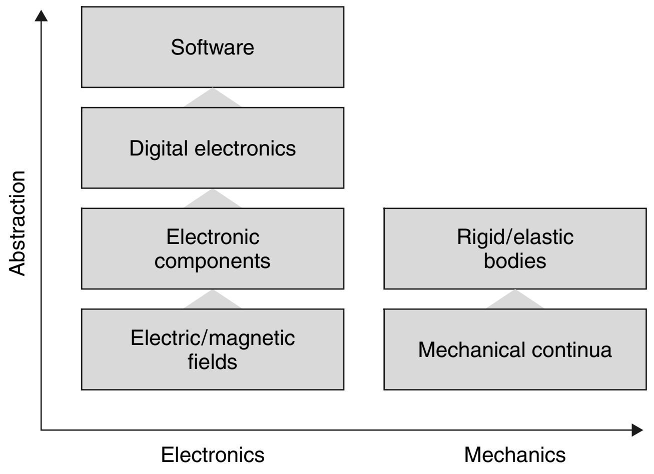


Figure 3.1 Levels of abstraction for electronic and mechanical models

only been possible to differentiate between two levels of abstraction, the continuum level and the level of multibody systems in which rigid and elastic bodies are each considered as a unit. In particular, we cannot neglect the continuum level for the consideration of systems since an electro-mechanical transformation, e.g. sensors and actuators, occasionally cannot be abstracted to the multibody level. The demonstrators from the chapter on micromechatronics are a good example of this.

3.2.2 Analogies

Analogies on the level of electronic components and mechanical bodies represent the predominant theme running through the joint consideration of electronics and mechanics. By this we mean that electronics and mechanics can be described using equations that have the same structure. This is also made clear by the fact that the equations from both mechanics and electronics can be derived from the Lagrange principle, see Maißer and Steigenberger [252] and Section 6.2.2. 'Langrange approach'. The analogies between electronics and mechanics will first be explained by means of an example, see Ogata [300]. The diagram on the left-hand side of Figure 3.2 shows a simple mass-spring-damper system.

The differential equation describing the system is as follows:

$$m\ddot{x} + b\dot{x} + kx = F \tag{3.1}$$

First we have to find out which variables can be identified as being analogous with one another. One possibility is to associate forces with currents and velocities with voltages. In order to construct an analogue circuit, let us now consider the mechanical system more closely. In this all forces act upon the mass, i.e. upon a point, and correspondingly add up to zero. In electronics this corresponds with the situation

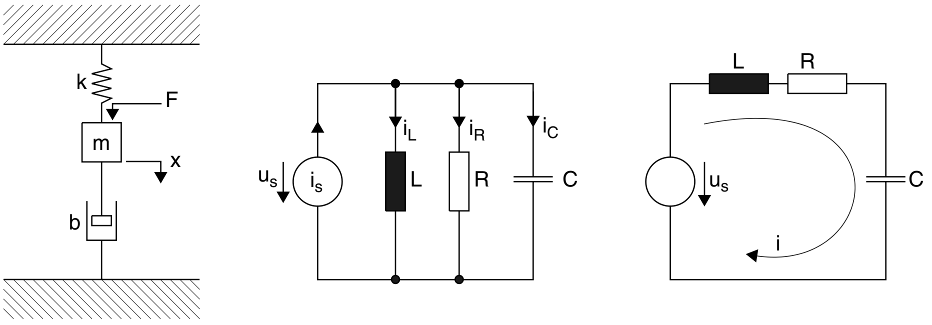


Figure 3.2 Mechanical system and two analogue circuits

in which all currents of analogue components meet at a node and also add up to zero there. Thus the circuit shown at the centre of Figure 3.2 represents an analogy with the mechanical system. Using Kirchhoff's current law the following is true:

$$i_L + i_R + i_C = i_s \tag{3.2}$$

where

$$i_L = \frac{1}{L}\int u_s\,dt, \qquad i_R = \frac{u_s}{R}, \qquad i_C = C\dot{u}_s \tag{3.3}$$

So equation (3.2) becomes:

$$\frac{1}{L}\int u_s\,dt + \frac{u_s}{R} + C\dot{u}_s = i_s \tag{3.4}$$

The magnetic flux ψ has the following relationship to the voltage u_s:

$$\dot{\psi} = u_s \tag{3.5}$$

Since voltage u_s is analogous to velocity, ψ, as an integral of the voltage, represents deflection. Thus equation (3.4) can be formulated as follows:

$$C\ddot{\psi} + \frac{1}{R}\dot{\psi} + \frac{1}{L}\psi = i_s \tag{3.6}$$

The structure of this equation exactly corresponds with equation (3.1). Capacitance is linked to mass here, damping to the inverse of resistance and the spring constant to the inverse of inductance. Lastly, the current i_s of the source corresponds with the activating force F.

Alternatively, we can also associate forces with voltages and velocities with currents. In this case the voltages, as the counterpart to the currents, must add up to zero and therefore must be arranged in a loop, see the right-hand side of

Figure 3.2. The following applies:

$$\mathrm{L}\dot{\mathrm{i}}_s + \mathrm{R}\mathrm{i}_s + \frac{1}{\mathrm{C}}\int \mathrm{i}_s \mathrm{d}t = \mathrm{u}_s \tag{3.7}$$

If we formulate the equation with the aid of charge q, it becomes:

$$\mathrm{L}\ddot{q} + \mathrm{R}\dot{q} + \frac{1}{\mathrm{C}}q = \mathrm{u}_s \tag{3.8}$$

This equation too corresponds with the structure of equation (3.1). Now, however, the inductance is linked to the mass, the resistance to the damping, and the spring constant to the inverse of capacitance. The voltage u_s of the source is associated with the activating force F here.

We can thus differentiate between two types of analogy, which differ from one another primarily in the assignment of variables and basic elements. The force–current analogy that we investigated first has the advantage that it retains the structure of the mechanical system, see Crandall *et al.* [75]. Parallel circuits remain parallel circuits, series circuits remain series circuits. Kirchhoff's current and voltage laws apply accordingly, i.e. forces/currents at a node and (relative) velocities/voltages in a loop cancel each other out. The two Kirchhoff's analogies do not apply, if — as in the second case — forces and voltages are identified as analogous. Table 3.1 shows the most important relationships for the force-current analogy.

3.2.3 Limits of the analogies

The analogies described above are based upon linear relationships. However, this circumstance often cannot be guaranteed. For example, the Stokes' friction or viscous friction has a linear relationship with velocity in a first approximation and can thus be represented as a resistance. However, this is very definitely not the case for the Coulomb friction. Here we can differentiate between two states of static and sliding friction, for which different coefficients of friction apply. Furthermore, the Coulomb friction is not dependent upon velocity but on another variable — the perpendicular force. The Newton friction of bodies moved quickly through a fluid finally depends upon a few parameters, such as the frontal area, the drag coefficient and the density of the fluid, but above all on the square of the velocity. In order to construct an analogy for the Coulomb friction we need a resistance controlled via the normal force, i.e. via the corresponding current, which switches the coefficient of friction in an event-oriented manner upon the transition from static to sliding friction and vice versa. The Newton friction of bodies moved through a fluid, on the other hand, can best be represented as a resistance with a quadratic characteristic. We have thus already dealt with a good proportion of the components normally considered in analogue electronics.

The transition from one-dimensional to three-dimensional mechanics represents the limit of the consideration of analogies. The analogies can no longer be used

Table 3.1 Analogies between analogue electronics, translational and rotational mechanics

Analogue electronics	**Translational mechanics**	**Rotational mechanics**
Current i	Force F	Torque M
Voltage u	Velocity v	Angular velocity ω
Coil $u(t) = L \cdot \frac{d}{dt} i(t)$	Spring $v(t) = \frac{1}{k} \cdot \frac{d}{dt} F(t)$	Torsion spring $\omega(t) = \frac{1}{k} \cdot \frac{d}{dt} M(t)$
Capacitor $i(t) = C \frac{d}{dt} u(t)$	Inertia $F(t) = m \frac{d}{dt} v(t)$	Rotational inertia $M(t) = J \frac{d}{dt} \omega(t)$
Resistor $i(t) = \frac{1}{R} \cdot u(t)$	Damping $F(t) = b \cdot v(t)$	Rotational damping $M(t) = b \cdot \omega(t)$
Electrical power dissipation at resistor $P(t) = u(t) \cdot i(t)$	Mechanical power dissipation due to damping $P(t) = v(t) \cdot F(t)$	Mechanical power dissipation due to damping $P(t) = \omega(t) \cdot M(t)$
Magnetic energy $T(t) = \frac{1}{2} L i^2(t)$	Elastic energy $T(t) = \frac{1}{2} \cdot \frac{1}{k} F^2(t)$	Elastic energy $T(t) = \frac{1}{2} \cdot \frac{1}{k} M^2(t)$
Electrostatic energy $T(t) = \frac{1}{2} C u^2(t)$	Kinetic energy $T(t) = \frac{1}{2} m v^2(t)$	Kinetic energy $T(t) = \frac{1}{2} J \omega^2(t)$
Transformer $i_1 \cdot u_1 = i_2 \cdot u_2$ $i_1 = \alpha i_2$ $u_1 = \frac{1}{\alpha} u_2$	Lever $F_1 \cdot v_1 = F_2 \cdot v_2$ $F_1 = \alpha F_2$ $v_1 = \frac{1}{\alpha} v_2$	Gear $M_1 \cdot \omega_1 = M_2 \cdot \omega_2$ $M_1 = \alpha M_2$ $\omega_1 = \frac{1}{\alpha} \omega_2$
Sum of all currents at a node is zero	Sum of all forces at a point is zero	Sum of all moments at a point is zero
Sum of all voltages in a closed loop is zero	Sum of all relative velocities in a closed loop is zero	Sum of all relative angular velocities in a closed loop is zero

in this case, see Crandall *et al.* [75]. This becomes clear intuitively if we look at the example of a robotic arm. In the calculation of kinematics and dynamics, three-dimensional translational movements and triaxial rotational movements are calculated independently of one another. There is no parallel to this in electronics. Furthermore, analogies in the sense described are defined exclusively for the consideration of concentrated components and continuous quantities. Continuum mechanics, digital electronics and software thus remain outside their scope and must be considered separately.

3.2.4 Differences between electronics and mechanics

In what follows the primary differences between electronics and mechanics will once again be briefly summarized, see also Cellier [62].

With the exception of high-frequency circuits, electronics can be considered exclusively in topographic form in the simulation. The precise geometry is unimportant or can be considered using simple parameters. This is not the case in mechanics, where three components of translation and three components of rotation have to be taken into account.

Furthermore, translation and rotation cannot be considered independently of one another, as illustrated by gyroscopic forces such as the Coriolis force.

A whole range of reference systems are relevant to the description of position, movement and acceleration. We have the inertial system and various fixed body reference systems, the origins of which may lie approximately at the centre of gravity or at the coupling points. In electronics there is only a reference voltage (ground) as the 'inertial system', and voltage or current arrows as fixed-component 'reference systems'.

In electronics, and in particular in microelectronics, we sometimes have some tens of millions of components. In mechanics, at most, a few tens to a few hundreds of basic elements, e.g. rigid bodies, joints, springs, etc. have to be taken into account.

The movements of mechanical bodies are typically subject to a whole range of limitations. Mechanical stops are one example. Springs can only be extended up to a certain degree. Elastic bodies deform under the effects of force. Similar effects can also be found in electronics but they are far less prominent than is the case in mechanics.

3.3 Model Transformation

3.3.1 Introduction

We can now specify a class of simulators and use this as the basis for the description of models in the other domains. In principle, the basic simulator should be sought out on the basis of the focal point of the desired investigation. In what follows we will describe approaches based upon circuit simulators, logic or Petri net simulators, multibody simulators, and finite-element simulators.

3.3.2 Circuit simulation

Introduction

In a circuit simulator the formulation of transformed models classically takes place in a hardware description language. This approach is the main theme of the present work and will be described comprehensively in the following chapters. Alternatively, it is also possible to draw up equivalent circuit diagrams for mechanical components. We can initially differentiate between two possibilities here. Firstly, we can use the analogies introduced in Section 3.2.2 to associate electronic components with basic mechanical elements. The other option is to model not the mechanics itself, but rather the differential equations that describe the mechanics.

Analogy approach

In order to consider the analogies we must first refer to Section 3.2.2. The force/current analogy is normally used. In addition to the basic elements, other mechanical phenomena such as Coulomb friction have to be considered. These require behavioural modelling based upon sources that can be controlled by arbitrary mathematical functions. Such voltage and current sources are available in PSpice, for example. This represents a rudimentary form of modelling in a hardware description language.

Yli-Pietilä *et al.* [431] use this method to investigate mechatronic systems such as a linear drive. They model a DC motor with an electronic control system and a mechanical load. The same approach is further elaborated by Scholliers and Yli-Pietilä in [369] and applied to other examples, such as a double pendulum. In [368] Scholliers and Yli-Pietilä introduce a whole library of such models, which expand the field of application of a circuit simulator such as Spice in the direction of mechatronics.

Examples for the use of equivalent circuit diagrams in micromechatronics are supplied, for example, by Antón *et al.* [13] (pressure sensor elements), Garverick and Mehregany [111] (micromotors), or Lo *et al.* [236] (resonators).

Modelling of differential equations using equivalent circuits

As an alternative to the analogy approach described above we can also find an equivalent circuit for the underlying system of equations. In principle, this procedure is similar to the construction of a rudimentary analogue computer from electronic components. In this context we can differentiate between explicit and implicit methods, see Bielefeld *et al.* [31]. In the explicit version the values of the state variables are represented as voltages in the network. In this, the highest time derivative of each state variable is set depending upon lower derivatives and other state variables using a controlled voltage source. In addition, there are integrators, see the left-hand side of Figure 3.3, which again provide the low derivatives in the form of voltages, see Herbert [139]. As an alternative to this, the implicit method, see Paap *et al.* [312], solves a set of n equations in the form:

$$f(\mathbf{x}, \dot{\mathbf{x}}, t) = 0 \tag{3.9}$$

where $\mathbf{x}$ represents a vector of n unknowns. As in Herbert [139] the states are represented as node voltages. Each equation is defined by a current from a voltage-controlled current source. This sets the input current of the differentiator in the

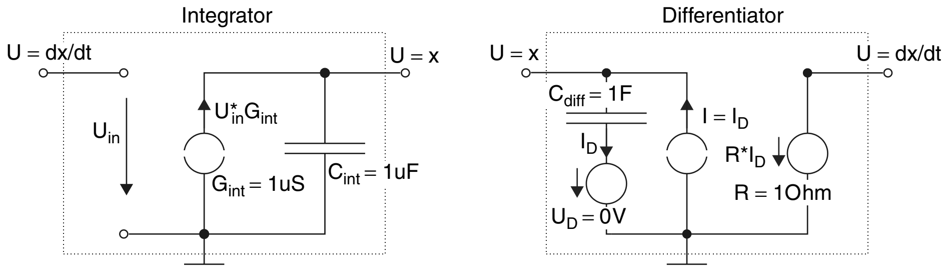


Figure 3.3 Equivalent circuits for integrator and differentiator

right-hand side of Figure 3.3 to the value $f(\mathbf{x}, \dot{\mathbf{x}}, t)$. The circuit simulator ensures that no current flows into the differentiator and thus solves the differential equation.

It is also worth mentioning that the, somewhat tiresome, process of converting a system of differential equations has been automated using the MEXEL CAE tool, see Pelz *et al.* [322]. A model transformer reads in the differential equation system, simplifies it if necessary, and then writes out a Spice net list in explicit or implicit formulation.

3.3.3 Logic/Petri net simulation

Introduction

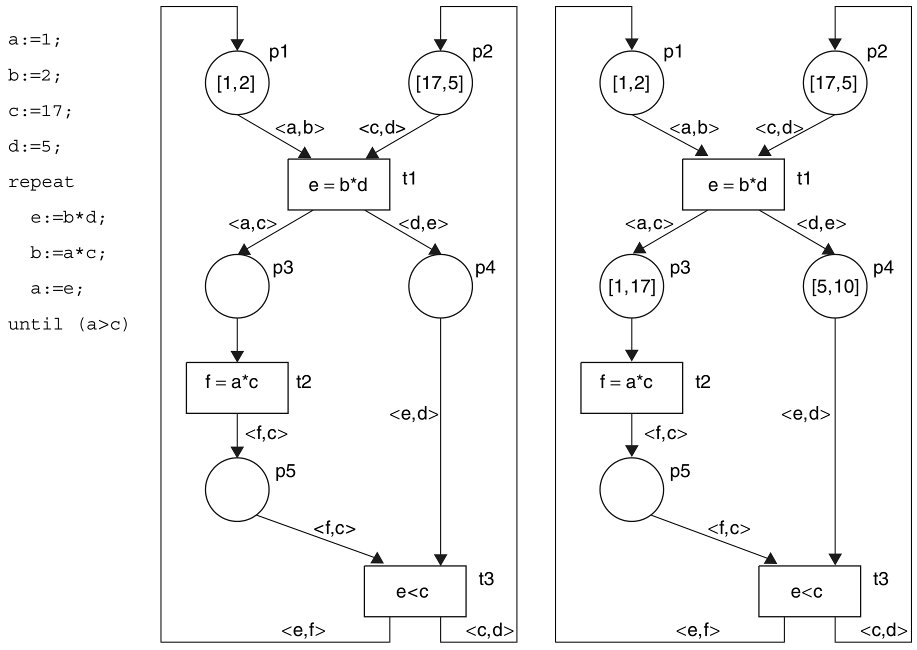
Predicate/transition networks (Pr/T networks), see [115] represent an extension of Petri nets and are often used for the modelling of software and/or digital electronics. They permit a system description on a very abstract level in which the use of hierarchies permits particularly compact representations. The strength of Pr/T networks lies in the effective consideration of parallel processes. Brielmann *et al.* [46], [47], [48] and Kleinjohann *et al.* [199] introduce methods for describing mechanics and other physical domains, plus the associated interfaces using the resources of the Pr/T networks. Such model transformations thus provide the option of describing and simulating mixed systems in a consistent manner. The representation of the hardware description language VHDL in a coloured Petri net by Olcoz and Colom in [301] shows that Petri net simulation and logic simulation are not so very different from each other, which means that the events portrayed in the following section could well be achieved on the basis of digital hardware description languages.

Definition of Predicate/Transition nets

Pr/T nets consist of places, transitions, and directional edges between these. Places can contain identifiable markings, which represent the state of the network. If a marking is sufficiently high at the inputs of a transition and if these satisfy any additional conditions, then the transition can 'fire'. In this case the markings in question are cleared from the input places, new markings are generated at the output places and predefined actions may be performed where applicable. Such a network can be formulated in very compact form using the tools of predicate logic, e.g. in the Prolog language, see Negretto [295]. In this connection, a marking at a place conveys the information that a predicate assigned to that place is fulfilled. In order also to correctly take account of the timing of the individual components it is necessary to add in a concept of time. So in [47] two delays are assumed for a transition. One relates to the period of time for which the markings must be present at the input places before the associated transition can fire. The other describes the time that elapses between the firing of the transition and the generation of the output markings.

Modelling of discrete relationships

Let us now clarify how a Pr/T net works on the basis of a small example from [46], see Figure 3.4. On the left-hand side a piece of code is represented at the start of which some variables are initialised. There follows a loop, in the body of which various arithmetic operations are performed. The termination condition for the loop is located at the end of the loop and is based upon a comparison of two variables. In the centre and at the right-hand side of Figure 3.4 Pr/T nets are represented in different states. The variant in the middle shows the initial occupation of the markings and thus the situation after initialisation. The two calculations are located



```
a:=1;
b:=2;
c:=17;
d:=5;
repeat
  e:=b*d;
  b:=a*c;
  a:=e;
until (a>c)
```

Figure 3.4 Modelling of digital behaviour using Pr/T nets

in the transitions t1 and t2. Transition t3 compares the corresponding values. If the newly calculated a is less than c, then the values are once again entered into places p1 and p2, which represent the input places of the loop. Otherwise the calculation of the loop is broken off and further transitions that are not shown can fire. The diagram on the right-hand side of Figure 3.4 shows a state in which the transition t1 has already fired for the first time. Accordingly, the new values of a and c have been entered at place p3 and the new values of d and e have been entered at place p4. Other constructs of a programming language can be depicted in the same manner.

Modelling of continuous relationships

Continuous relationships are classically modelled using differential equations that can be either linear or nonlinear. Let us now model such equations on the basis of Pr/T nets using the event-oriented modelling introduced in the previous section. A solution for linear differential equations on the basis of the Z transformation was proposed by Brielmann and Kleinjohann [46]. In what follows we will, however, predominantly consider nonlinear systems. This property can have two causes: firstly, nonlinearity can arise as a result of discontinuities; secondly, it may be caused by nonlinear functions in the system equations [47]. The first case is very simple to solve. Here only the current equation has to be activated, which can be performed by simply distinguishing between cases. The example from Figure 3.4 illustrates this state of affairs. The termination condition at the end of the loop corresponds with such a case differentiation. More difficult is the realisation of the other variant. In [47] it is proposed to put the equations together step-by-step from linear components, meaning that here too a swapping of linear components would be necessary. Furthermore, the differential equations have to be numerically integrated, which is achieved using the Euler principle:

$$\dot{x}(t) \approx \frac{x(t+h) - x(t)}{h} \tag{3.10}$$

where h is the time step of the integration. Now if the differential equation of interest is

$$\dot{x}(t) = f(x, t) \tag{3.11}$$

the integration formula is found to be

$$x(t+h) \approx h \cdot \dot{x}(t) + x(t) = h \cdot f(x, t) + x(t) \tag{3.12}$$

which, along with an additional function g(x,u,t) to determine the outputs, can be directly represented on a Pr/T net, see Figure 3.5.

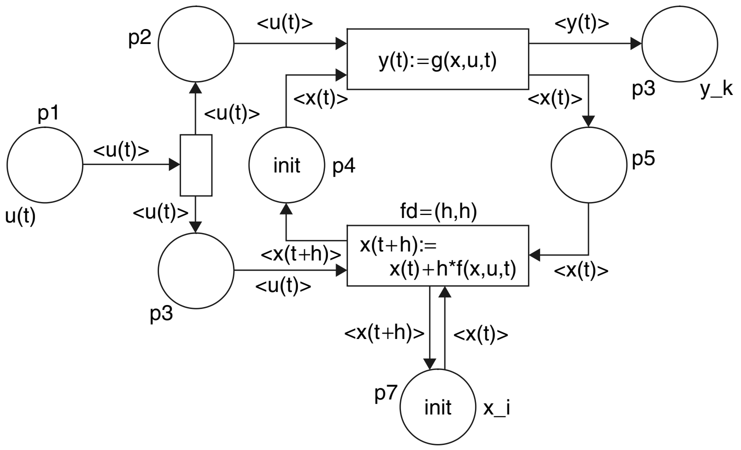


Figure 3.5 Modelling of a nonlinear differential equation using a Pr/T network

3.3.4 Multibody simulation

Introduction

In this section two approaches will be introduced: Firstly the equations of electronics will be obtained using the Lagrange principle, so that they can be seamlessly incorporated into a multibody simulator based upon the Lagrange principle. The other method is based upon object orientation, thus allowing the non-mechanical components to be modelled more or less independently of the system as a whole.

Electronic modelling using the Lagrange approach

In [253] Maißer describes a principle that uses the Lagrange approach from mechanics in order to find model equations for the electronics of a mechatronic system. In this manner the electronics can be easily incorporated into the multibody simulator, which may also be based upon the Lagrange equations. Mechanics and electronics are thus modelled using a unified approach and simulated as a whole system.

Object-oriented approach

This section introduces an approach that combines modelling on a component level with the automatic creation of a system model. As in software development this 'local' procedure is called object-orientation. Such approaches are naturally particularly well suited for describing nonmechanical parts of the system in a form that is suitable for a multibody simulator.

Kecskeméthy [185] and [186] as well as Risse *et al.* [346] describe a simulation environment for mechatronic systems that includes the electronics of a controller. This takes place in the form of abstract controller equations, developed using a suitable tool, e.g. MATLAB/Simulink. In this connection a class of controllers is prepared in [346] that includes continuous, proportional, discrete and mixed controllers. Simple, electronic components can also be described on the same basis. The underlying equations are added to the equations of motion of mechanics, and the equations of sensors and actuators, and are then solved as a whole.

3.3.5 Finite-element simulation

One possibility for system simulation using a FE simulator is to fuse the equation system of electronics together with the equation system of finite elements. The resulting equations include the sought-after unknowns from electronics and mechanics. The complete system can thus be processed using a standard solver.

Particularly important in this context is the work of Bedrosian [22], who expanded a finite element simulator for the calculation of electromagnetic fields so that it could process both analogue circuits and also the kinematics of rigid bodies. A significant aspect of this is to obtain a few desirable properties of FE matrices. So in contrast to the matrix for the finite elements, the system matrix would be neither positive definite nor sparse. Bedrosian therefore insists upon a separate consideration of the matrices for the individual domains, which requires a suitable iteration in order to obtain a consistent solution for the system as a whole.

3.3.6 Evaluation of the model transformation

The introduction of analogue hardware description languages has caused interest in equivalent circuits for mechanical components to fall sharply. This is primarily because a hardware description language is significantly more flexible in its formulation. This is true particularly for components for which the analogies provide no direct parallel. Furthermore, the overview is quickly lost if it is unclear what the equivalent voltages and currents represent.

In principle, the modelling of continuous relationships on an event-oriented basis — for example using digital logic or a Pr/T network — is nothing unusual. Every simulator for analogue processes that is run on a digital computer has the same fundamental problem to solve. The difference in the present case is that the basic functions of the simulation, such as the integration procedure or the automated selection of a suitable step size, have to be modelled fully by the user, which firstly can be very cumbersome and secondly presumably raises a performance problem.

When discussing the simulation of mechatronic systems in a multibody simulator it is particularly worth mentioning the elegant solution of Maißer [253], which models the electronics according to the Lagrange principle, so that the resulting equations are compatible with multibody simulation, which is also based upon the Lagrange approach. However, the lack of any significant libraries of transistor models and the fact that digital electronics and software are disregarded, are problematic.

For the variant of model transformation on the basis of a FE simulator, the field of application of a corresponding solution is only a little wider than that of the FE simulator. This is also emphasised by the fact that there is comparatively little literature in this field.

3.4 Domain-Independent Description Forms

In this section approaches will be described that cannot be classed as an expansion of the tools in a certain domain. The most important representatives here are bond graphs, block diagrams and modelling languages such as Modelica, Dymola or ACSL.

3.4.1 Bond graphs

The bond graph approach, see for example Karnopp and Rosenberg [180] or Thoma [398], fundamentally rests upon the same principles as the analogies in electronics and mechanics, see Section 3.2.2. However, there is one significant difference. In the analogies, currents were generally identified with forces/moments and voltages with velocities, so that an analogy in the form of an equivalent circuit has the same structure as the original system. This is true because according to Kirchhoff's laws, currents and forces add up to zero at a node and voltages and relative velocities add up to zero in a closed loop.

By contrast, in the bond graphs, the following classifications are made. Voltages are normally associated with forces/moments and called effort, currents are associated with velocities/angular velocities and called flow. The elements used in the bond graph approach can be divided into one, two and three-port networks. The one-port networks are the so-called C, I and R elements, which in electronics correspond with capacitors, inductors and resistors and in mechanics correspond with springs, masses and dampers, see Table 3.2. In addition there are sources for effort and flow. Transmission elements and gyrators are defined as two-port networks. The former transmit effort to effort or flow to flow in a fixed or variable relationship to one another; the latter put the effort, on the one hand, into a relationship with the flow, on the other (and vice versa). Transmission elements can thus be transformers, gears or levers for small deflections. A gyrator could for example describe a DC motor. The three-port networks finally represent serial or parallel junctions (s-junction, p-junction). The one, two and three-port networks are linked together by half arrows, so-called bonds, which each bear an effort and a flow. The direction of the arrow shows the direction of the positive power flow. The work done is found by the product of effort and flow. In addition to the half arrows of the bonds there are also connections with a full arrow, in which either the effort or the flow is neglected. These connections carry information, but no energy.

The calculation of bond graphs first of all requires the drawing up of a suitable system of equations, which is generally explicitly formulated. This means that the

Table 3.2 Assignment of magnitudes and elements in bond graphs

Bond graphs	Electronics	Mechanics, translational	Mechanics, rotational
Effort	Voltage	Force	Torque
Flow	Current	Velocity	Angular velocity
C element	Capacitor	Spring stiffness	Torsional spring stiffness
I element	Inductor	Mass inertia	Moment of inertia
R element	Resistor	Damping, translational	Damping, rotational
Transmission element	Transformer	Lever, pulley block	Gears

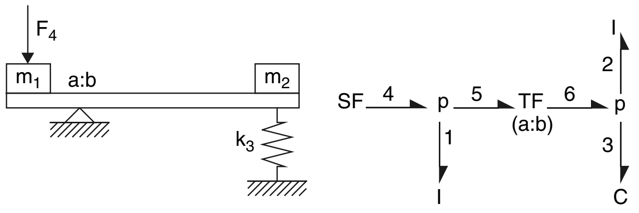


Figure 3.6 Bond graph of a simple mechanical system

equations take the form of instructions, and this fact requires a consideration of the causality of the system. Therefore, cause and effect have to be specified for each element. If we take any C, I or R element we can ask whether the effort causes the flow or vice versa. Both are possible and there are equations for both cases, which can be used in a system of equations if required. Overall, it is a question of creating continuous chains of cause–effect relationships, which can be illustrated by a suitable sequence of assignments. In the case of algebraic loops this cannot be achieved, so additional measures are necessary.

In what follows, a few examples of bond graphs will be presented. Figure 3.6 shows the bond graph of a simple mechanical system, which consists of two masses, a spring and a lever. In addition to I and C elements the bond graph contains a flow source, which represents the force F_4 and is designated SF. The transmission element TF represents the lever, which sets a ratio (a : b).

Figure 3.7 shows a simple circuit and the associated bond graphs. This again includes the flow source SF. However, this now describes a current source. The transmission element TF is also present and represents the transformer.

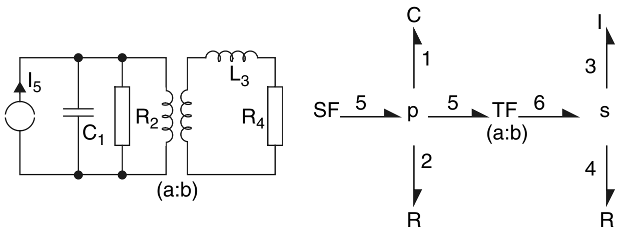


Figure 3.7 Bond graph of a simple electrical system

3.4.2 Block diagrams

Block diagrams are often used in control technology and, like bond graphs, represent a form of structural modelling, see Cellier [62]. However, this type of representation primarily shows the structure of equations, whereas the structure of the system tends to be found indirectly from the structure of the equation system.

Block diagrams include blocks and directional connections between the blocks. These connections describe signals, which are converted into other signals by the blocks. In addition there are taps and summing points, so that the important elements of block diagrams can be fully represented in Figure 3.8.

In what follows, modelling using block diagrams will be illustrated on the basis of a simple example. For this purpose we will consider the circuit on the left-hand side of Figure 3.9. This can be described on the basis of the following equations:

$$
\begin{aligned}
& i_1 = \frac{u_1}{R_1}, i_2 = \frac{u_2}{R_2} \\
& \dot{i}_L = \frac{u_L}{L_1}, \dot{u}_C = \frac{i_C}{C_1} \\
& u_1 = U_0 - u_C, u_2 = u_C, u_L = u_1 + u_2 \\
& i_0 = i_1 + i_L, i_C = i_1 - i_2
\end{aligned}
\tag{3.13}
$$

If the above equations are translated into blocks, connections and summations, we obtain the block diagram on the right-hand side of Figure 3.9. The main problem

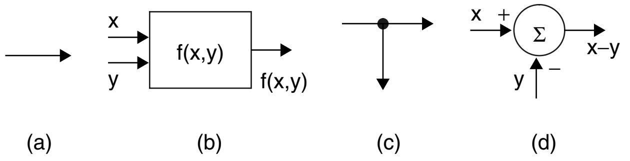


Figure 3.8 Basic elements of block diagrams: Connection (a), block (b), tap (c) and summer (d)

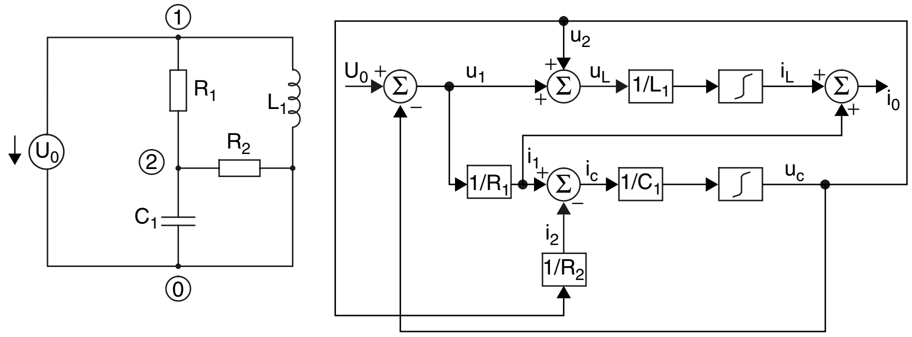


Figure 3.9 Block diagram of an electronic circuit

here is that the structure of the circuit no longer corresponds with the structure of the block diagram.

3.4.3 Modelling languages for physical systems

Languages such as ACSL [2], DSL [227], Dymola [90], [310] or Modelica [94], [272] in particular deserve a mention. All these languages support the description of physical systems. In what follows we will investigate Modelica in particular, as this language includes the most up-to-date research results and furthermore is currently being expanded to a standard, see [272] and [273]. An excellent introduction to object-oriented modelling of mixed systems in general and of Modelica in particular can be found in Otter [308].

Modelica is a language for the modelling of physical systems and was developed specifically in order to support the exchange of models and the development of libraries. Modelica does not insist upon an exclusively causal modelling, in which cause and effect of every component have to be determined even before the simulation. The description of the models can also take place in the form of genuine equations and not on the basis of assignments. Modelica supports the description of continuous systems, which can be calculated on the basis of differential-algebraic equation systems (DAE). In addition there are constructs for dealing with discontinuities, which may occur in mechanical stops, or static to sliding friction transitions. In principle it is also possible to use the discontinuities to describe event-oriented processes, e.g. transitions in a state graph or the movement of markings in a Petri net, but this possibility is limited by the underlying equation solver.

In principle, Modelica can be compared with an analogue hardware description language, see also Tiller *et al.* [400]. Both structural and behavioural modelling is possible. A particularly prominent feature of Modelica is object-orientation, which is used, for example, to declare a model — or to be specific a model class — once and instance it many times, with the option of setting certain parameters individually for each instance. Similar concepts also exist in hardware description languages, such as VHDL, with the possibilities of instancing and configuration. Modelica also offers the option of transmission between model classes, so that more complex model classes can easily be traced back to simpler ones.

To illustrate modelling in Modelica the description for an electronic circuit will be given in what follows, see Figure 3.10 and [273].

The associated Modelica model is represented in Hardware description 3.1, with key words shown in bold type. After the declaration of the `circuit` model the components along with their main parameters are declared. At this level the `equation` section specifies only the connectivity of the circuit.

```
model circuit
 Resistor R1 (R=10);
 Capacitor C (C=0.01);
 Resistor R2 (R=100);
```

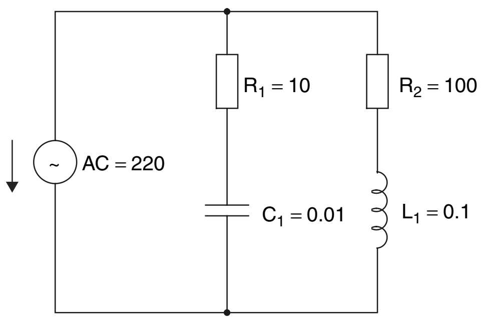


Figure 3.10 Electronic circuit as an example of a Modelica model

```
 Inductor L (L=0.1);
 VsourceAC AC;
 Ground G;
equation
  connect (AC.p, R1.p);
  connect (R1.n, C.p );
  connect (C.n, AC.n);
  connect (R1.p, R2.p);
  connect (R2.n, L.p );
  connect (L.n, C.n);
  connect (AC.n, G.p);
end circuit;
```

Hardware description 3.1 Modelica model of the circuit from Figure 3.10

Thus the components such as resistors, capacitors, etc. remain to be described, see Hardware description 3.2. These are successively built up via the model of a pin and the model of an electrical component with two terminals. One interesting feature here is the use of inheritance in the transition from the model with two terminals to the component. Using the key word `extends` the roles of voltage and current and Kirchhoff's current laws are loaded into the component model and do not need to be formulated there again. An electrical component can thus be simply described by its constituent equation. In the case of the capacitor, the time derivative of voltage is designated by the function `der()`.

```
type Voltage = Real (unit="V");
type Current = Real (unit="A");
...
connector Pin
 Voltage v;
  flow Current i;
end Pin;
...
partial model TwoPin "Parent class of the element with 2 elec.
  pins"
 Pin p, n;
 Voltage v;
 Current i;
equation
 v = p.v -n.v;
 0 = p.i + n.i;
 i = p.i
end TwoPin;
...
model Resistor "Ideal electrical resistor"
  extends TwoPin;
  parameter Real R (unit="Ohm") "Resistance"
equation
 R*i = v;
end Resistor;
model Capacitor "Ideal electrical capacitor"
  extends TwoPin;
  parameter Real C (unit="F") "Capacitance"
equation
 C* der(v) = i;
end Capacitor;
...
```

Hardware description 3.2 Model of the components from Hardware description 3.1

3.4.4 Evaluation of domain-independent description forms

From the examples shown above it is clear that bond graphs can describe both analogue electronics and mechanics (and also a range of further domains) in compact and graphic form. However, if we go beyond unidimensional mechanics and passive electronics there are significant problems to be solved. Although the modelling of transistors is also possible in principle using bond graphs, a meaningful simulation of circuits of substantial complexity remains the exclusive preserve of a dedicated circuit simulator. The same applies for three-dimensional multibody mechanics. Moreover, bond graphs are in principle limited to continuous systems, so that digital electronics and software cannot be illustrated using classical bond graphs, or at least this cannot be done efficiently. Furthermore, every element must be assigned a fixed causality prior to the simulation. This causality may alter during a simulation, for example, if an electric motor becomes a generator, so that such systems cannot be simply investigated using bond graphs. The same applies in principle for block diagrams.

Domain-independent languages, and Modelica in particular, are broadly comparable with analogue hardware description languages. However, they don't have the model basis of a circuit simulator. Furthermore, the event-oriented field is much weaker in comparison to hardware description languages in general, and VHDL-AMS in particular, so that digital electronics or software, as is demonstrated by Scherber and Müller-Schloer in [360], require the coupling of appropriate simulators to the equation solver that underlies the language.

Perhaps the most important objection against domain-independent description forms lies in the fact that it is necessary to start modelling up from scratch in every domain. Alternatively, if we build up from a circuit or multibody simulator, a large part of the system is already covered by the best available methodology.

3.5 Simulator Coupling

3.5.1 Introduction

The option of simulator coupling tackles the problem highlighted above in a straightforward manner. Appropriate simulators are already available for the various domains in the system and in the ideal case these would only have to exchange their current simulation results. The use of simulator coupling can protect investments in models and facilitate the use of the best available simulator for a field. However, simulator coupling is also associated with a whole range of problems. For example, it generally requires access to the internals of the simulators involved, which means that if commercial simulators are to be considered, the co-operation of the provider in question is required. Furthermore, the coupled simulation forms a very intricate software package, which is difficult to get to grips with. Perhaps the most important disadvantage, however, lies in the synchronisation of two normally very different simulator cores. In the coupling of analogue electronics and mechanics, differential equations are solved in both cases. However, their origin, nature and formulation are very different. Furthermore, this form of co-simulation is also associated with convergence problems, particularly in the case of a strong coupling between two analogue solvers.

3.5.2 Simulator backplane

When coupling two simulators, the principle of 'simulator backplane' represents a particularly systematic solution, see also Jorgensen and Odryna [171] or Maliniak [255]. This principle is equally suited to the coupled simulation of exclusively continuous, exclusively event-oriented or mixed systems. In principle, the simulator backplane is a standardised procedure, see Kemp [187], for the inclusion of simulators into an overall simulation, see Figure 3.11 from Zwoliński *et al.* [441]. The main task of a backplane is to undertake a partitioning of the design data before the actual simulation and to assign the individual parts of the simulators in question. The backplane also looks after the synchronisation between the linked simulators and the exchange of data. In the ideal case the backplane also has a unified user interface with the associated output tools, but this tends to be rare.

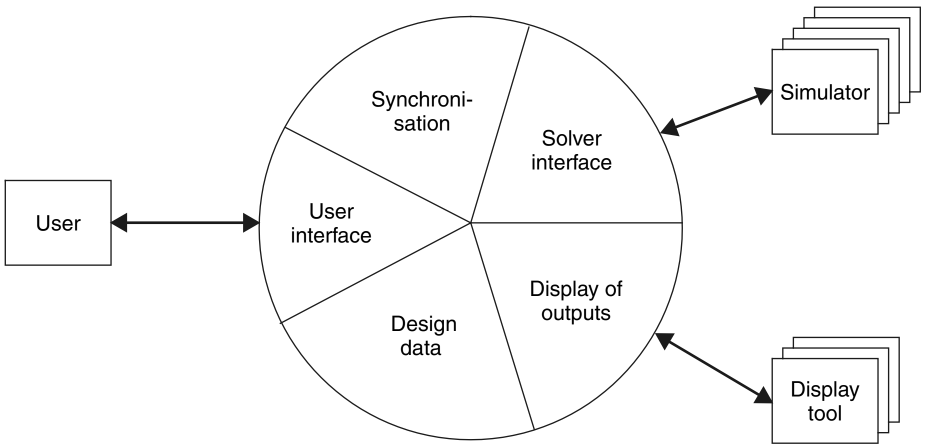


Figure 3.11 Structure of a simulator backplane

Otherwise, the corresponding settings in the individual simulators are used, which often leads to confusion. The data exchange between the backplane and the simulators can take place by means of an IPC interface.[1] This does not necessarily require that all simulators are processed on the same workstation. The load can be distributed across various computers as long as the synchronisation does not prevent this. However, the cost of communication via this comparably slow interface has to be borne. Faster simulations are generally achieved by the binding together of backplane and simulators into an overall programme. This is particularly true if a great deal of communication via the backplane is expected as a result of a strong coupling between domains in the simulated system, because in this case the addressing of a simulator from the backplane becomes a function call.

As for the general case of simulator coupling, the main problem of the backplane lies in the handling of the synchronisation between the simulator cores.[2] We can make an initial differentiation here between two classical approaches: the conservative, see Chandy and Misra [65], [66], and the optimistic, see Jefferson [168], Jefferson and Sowizral [169].

The conservative approach allows simulator A to proceed only for a time period in which it can be proven that no events sent out from other simulators are expected. These events would have to be taken into account in the simulation of A and consequently the simulation has to wait for them. In the conservative approach, simulator A is thus safe to proceed for this time period. In the extreme case, this conservative approach is called the ‘lockstep’ algorithm,[3] in which a fixed time

[1] IPC (inter process communication), communication between processes on the level of the operating system.

[2] The same problem also emerges in the parallelisation of simulations, see Fujimoto [107], where here the same simulator cores are synchronised on different processors.

[3] Other authors, such as Le Marrec *et al.* [218] or Olcoz *et al.* [302] describe the ‘Lockstep’ synchronisation as the specification of a global real time so that all participating simulators may each proceed their local time up to the global time.

interval is specified for all participating simulators. Particularly for systems with very different time constants this hinders an efficient processing of the simulation. In recent years a whole range of simulator couplings have been developed in the form of variations on the conservative method, e.g. Bechtold *et al.* [20], Buck *et al.* [52], Patterson [316], Sung and Ha [392], Todesco and Meng [402], Zwoliński *et al.* [441], although frequent task changes between simulators can still gives rise to performance problems in these approaches.

In the optimistic case, every simulator processes its internal events until no more activity can be determined, which in the ideal case is by far the most efficient way. Unfortunately, it may occur that another simulator generates an event for the first in this period. Then all of the first simulator's results from the moment in question must be discarded. To achieve this the simulator in question must perform a leap backwards (timewarp) and then start again at the time point in question. Depending upon the system under consideration this is associated with a high storage requirement for the saving of old states. Furthermore, depending upon the nature of the system under investigation, these timewarps can themselves become a performance problem. Normally, however, electronics simulators [402] and mechanics simulators do not provide the option of performing a timewarp, so that only the conservative approach and variations upon it remain. However, this is not necessarily the case for the co-simulation of hardware and software, see Chapter 5.

In addition to the synchronisation between two simulator cores, the question of the convergence of the solution also requires some consideration. This is particularly relevant for the coupling of two analogue cores, see Klein and Gerlach [196]. The reason for this lies in the back-coupling between the two simulator cores, which we will call A and B here. A maps its input x_A to its output y_A using a function f_A. B does the same with the function f_B. In the simplest case, the Gauss–Seidel iteration, the rule for the (k+1)th iteration step is:

$$\begin{aligned} x_B^{k+1} &= y_A^k = f_A(x_A^k) \\ x_A^{k+1} &= y_B^{k+1} = f_B(x_B^{k+1}) \end{aligned} \tag{3.14}$$

In this case oscillations may occur. In the worst case the iteration does not converge at all. Numerically more demanding methods, such as, for example, the Newton procedure, tend to converge better, but are not universally applicable due to the costly calculation of the required Jacobi matrices, see [196].

3.5.3 Examples of the simulator coupling

Introduction

In what follows the options and limitations of simulator coupling will be illustrated in more detail on the basis of a few examples from mechatronics and micromechatronics. This description will include the direct coupling between two simulators as well as the systematic consideration of several simulators with a backplane.

Mechatronics

In [302] Olcoz *et al.* describe the coupling of the VHDL simulator VSS with the mechanics simulator COMPAMM. Sensors and actuators are incorporated at the interface between electronics and mechanics and these are characterised by a pair of corresponding variables — one for each of the two simulator sides. The correspondence of such pairs is achieved by an interface written in C and C + +. The mechanics simulators can thus be operated using a fixed or variable time interval. In the former case the synchronisation between electronics and mechanics takes place at discrete, evenly distributed points in time that are specified by the fixed interval of the mechanics. In the latter case the mechanics simulator proceeds by a time interval and then informs the electronics simulator that it may proceed to this point. After confirmation from the electronics simulator the sequence begins again from the start.

A further approach for the coupling of simulators is mentioned by Scholliers in [367]. This approach emphasises the coupling of multibody mechanics, analogue electronics and control technology. ADAMS, PSpice and MATLAB/Simulink are the simulators used. The simulation process is centrally controlled and a fixed increment thereby specified. The application considered is a controlled drive and the mechanical load is a mechanism described in ADAMS. The actuator is a direct current motor described in the form of Spice components, whereas the PI controller exists on a purely functional level in MATLAB/Simulink.

Le Marrec *et al.* [218] describe a coupling between C routines, the VHDL simulator VSS and MATLAB/Simulink using a co-simulation bus that exchanges data between the individual simulators. The simulation can take place on two levels. Firstly, simulation can be purely functional, with electronics, mechanics, and software being investigated for the application under consideration. In the other case, the timing has to be taken into account too, necessitating a processor model in VHDL for the software. In this case the problem is merely that of the co-simulation of electronics and mechanics. The approach described is illustrated on the basis of two examples, an electronic accelerator pedal for an electric car and the control of a hydraulic suspension system for a car.

In [360] Scherber and Müller-Schloer proposed a simulator backplane that represents a mechanism for the linking of very different simulators. The approach is based upon a unified model for the heterogeneous components involved. These are termed actuators; their interfaces are called ports; every two ports can be linked by a channel. The access mechanisms are always the same. Thus the interfacing of a component and its simulation is unified without having to make limitations with regard to the nature or function of the actuators. A scheduler decides which actuators shall be executed when and for how long by means of a priority analysis. In this manner a software simulator, a simulator for finite state machines, a simulator for the Modelica language — see Section 3.4.3–and MATLAB/Simulink were connected together.

Micromechatronics

In [121] Götz *et al.* produce a coupling between a finite element simulator and a circuit simulator. The idea consists of calculating the deformation of a pressure sensor structure using a finite element simulator and using the results to determine changes of piezo resistances. The circuit simulator is then used to determine the output of the read-out circuit. This consists primarily of a frequency modulation. Using this process the mechanical structure can be optimised very simply. However, the feedback of the electronics on the mechanics and any dynamic effects of the mechanics have not been taken into account.

A coupling between a FE simulator (ANSYS) for continuum mechanics and the circuit simulator PSpice for analogue electronics is proposed by Dötzel and Billep [86] and Klein *et al.* [198]. The applications used here are the simulation of micromirrors [86] and force sensors [198]. However, details of the coupling are not given in either case.

3.5.4 Evaluation

In simulator coupling the performance of the coupled tool is beneficial because the optimal modelling method can be selected for each field. This reduces the total modelling cost incurred, whilst the validation of the models can utilise the results within the domain in question. Further domains can be taken into account by linking in appropriate simulation tools. On the other hand, simulator coupling leads to problems in the operation of the simulator package and in handling the data flow at the interface. The simulators involved must be suitably synchronised with one another. The simulation time is typically very high due to the necessary iterations for each time interval. Finally, convergence problems may occur if there is strong coupling between the subsystems.

3.6 Summary

Various approaches to the modelling and simulation of mechatronic and micromechatronic systems have been considered in this chapter. We can differentiate between three groups of methods: model transformation, modelling in a domain-independent form, and simulator coupling. There are currently two options on offer that allow us to cover the whole spectrum of analogue electronics, digital electronics, software, multibody mechanics and continuum mechanics. These are simulator coupling and the universal modelling in hardware description languages, which will be described comprehensively in what follows.