Bond graphs and object-oriented modelling—a comparison

W Borutzky
Bonn-Rhein-Sieg University of Applied Sciences, D-53754 Sankt Augustin, Germany

Abstract: Bond graph modelling was devised by Professor Paynter at the Massachusetts Institute of Technology in 1959 and subsequently developed into a methodology for modelling multidisciplinary systems at a time when nobody was speaking of object-oriented modelling. On the other hand, so-called object-oriented modelling has become increasingly popular during the last few years. By relating the characteristics of both approaches, it is shown that bond graph modelling, although much older, may be viewed as a special form of object-oriented modelling. For that purpose the new object-oriented modelling language Modelica is used as a working language which aims at supporting multiple formalisms. Although it turns out that bond graph models can be described rather easily, it is obvious that Modelica started from generalized networks and was not designed to support bond graphs. The description of bond graph models in Modelica is illustrated by means of a hydraulic drive.

Since VHDL-AMS as an important language standardized and supported by IEEE has been extended to support also modelling of non-electrical systems, it is briefly investigated as to whether it can be used for description of bond graphs. It turns out that currently it does not seem to be suitable.

Keywords: bond-graph-based physical systems modelling, object-oriented modelling, textual model description languages, model exchange

NOTATION

A	cross-sectional area of a hydraulic orifice
C_p	capacitance of the oil in a hydraulic pump
dp	pressure difference across a hydraulic orifice
e	effort variable at a port of an element or component
f	flow variable at a port of an element or component
N	number of power ports
p	power port
q	generalized displacement
Q	volume flowrate
R_l	leakage coefficient for a hydraulic pump
t	time
x	displacement
α_j	coefficients equal to ± 1
τ	time

1 INTRODUCTION

Object-oriented modelling (OOM) has been inspired by the paradigm of object-oriented programming (OOP) in software engineering [**1**] in the sense that general properties are captured in generic model classes and are inherited by more special models. Accordingly, *encapsulation* and *inheritance* are frequently used terms also in OOM together with the notions *non-causal modelling* and *reuse of models.* While modellers prefer to develop models of systems to be designed at a graphical level, the OOM approach has been backed up by a number of textual modelling languages such as Dymola [**2**], Omola [**3**], ULM [**4**], or SIDOPS+ [**5**], just to mention some of them. While these languages have proved useful in various application areas, features of them have been combined into the so-called unified OOM language Modelica [**6**] in an international effort. A major goal of the design of Modelica has been to promote the exchange of physical systems models between various (proprietary) simulation packages and their reuse and to introduce in that way a new *de facto* standard. The core language specifications were completed in December 2000. Further development of Modelica and of Modelica libraries is organized by the non-profit non-governmental Modelica Association. The language together with some libraries is freely available. Almost all language constructs are supported by the modelling and simulation package Dymola [**2**]. The first textbook on Modelica has been published recently [**7**].

At the same time the hardware description language

The MS was received on 9 January 2001 and was accepted after revision for publication on 23 August 2001.

VHDL has been extended in a standardization effort by IEEE to allow for modelling also all kinds of non-electrical systems [**8**]. The extension is formally denoted as VHDL 1076.1 although the notation VHDL-AMS standing for VHDL analogue mixed signal extensions seems to have become more common. In regard to modelling of energy flows such as Dymola and Modelica this language extension is based on the concept of *generalized networks* and on Kirchhoff's laws generalized to power variables called *across* and *through* variables [**9**]. Moreover, like Dymola and Modelica it also supports non-conservative signal-flow models at the functional level represented by block diagrams. Standardized by IEEE, VHDL-AMS is expected to receive wide attention, acceptance and support from software companies in the near future. For instance, commercial simulation packages VeriasHDL™ [**10**] from Analogy Inc., ADVance MS [**11**] from Mentor Graphics Corporation and hAMSter [**12**] from SIMEC do support VHDL-AMS.

On the other hand the *bond graph methodology* based on the concept of energy exchange between system components, and especially suited to modelling mechatronic systems has been used worldwide in academia as well as in industry for nearly 40 years since it was devised by H. Paynter at MIT in 1959 [**13**]. A recent comprehensive presentation of bond graph methodology may be found in references [**14**] to [**17**]. It seems that both methodologies, object-oriented physical systems modelling and bond-graph-based physical systems modelling, so far have had only little impact on each other.

In this paper the characteristics of both physical systems modelling approaches are related to each other. It is shown that bond graphs, although much older than OOM, can be viewed as a special form of the latter methodology. In particular, by using the new modelling language Modelica as a representative of a class of OOM languages it becomes obvious that bond graph models may be easily described in the multi-purpose object-oriented language Modelica. For illustration purposes a controlled hydraulic drive is modelled as an application example by means of bond graphs and is described in Modelica. Since the aim of the paper is a comparison of two modelling approaches, it focuses on those features of Modelica that can be related to bond graph modelling. A comprehensive presentation of the latest state of Modelica in terms of specification, implementation and applications can be found on the Modelica homepage [**6**].

In addition to Modelica relations between bond graphs and VHDL-AMS are also briefly discussed. Extensions to VHDL in regard to a full support of OOM have been proposed by Benzakki and Djafri [**18**]. Relations between bond graphs and Dymola have been discussed in reference [**19**]. A more comprehensive survey which also covers other modelling languages has been given by Borutzky [**20**].

The paper is organized in the following manner. The next section summarizes essential features of the OOM approach. Section 3 considers bond graph modelling from an object-oriented perspective and clearly shows that bond graph modelling may be viewed as a form of OOM. With this result it is an obvious question whether modern OOM languages can be exploited for representing hierarchical bond graph models. The description of hierarchical bond graph models by means of Modelica is demonstrated in Sections 4 and 5. The compatibility of VHDL-AMS with bond graph methodology is briefly considered in Section 6. Finally, results are summarized and some conclusions are drawn in Section 7.

2 CHARACTERISTICS OF OBJECT-ORIENTED PHYSICAL SYSTEMS MODELLING

In this section, essential characteristics of object-oriented physical systems modelling are summarized in order to relate them in the next section to the bond-graph-based physical systems modelling approach. A presentation of the object-oriented physical systems modelling approach may be found, for example, in references [**3**] and [**21**] to [**23**].

Object-oriented physical systems modelling may be characterized by the following features.

2.1 Objects

In object-oriented physical systems modelling, models of components of technical systems as well as models of physical processes are considered *objects*. Since basic models consist of inherent parameters, e.g. length, mass, moment of inertia, resistance and data provided from the outside world through so-called *interfaces*, e.g. values of power variables and constitutive equations, there is a correspondence to objects in the sense of OOP. Physical systems models may be viewed as an aggregation of data and methods operating on them. In OOP the term *method* means a function or a procedure that can process data of a defined type. For instance, the coordinates of a point, a length, and a function that returns the area of a circle around that point may be aggregated into a data type or class called a *circle*. A circle of given radius around a given point may be named c1. Then c1 is a particular object of the class circle. Likewise, the voltage across and the current through a resistor, parameters and the constitutive (non-linear) relation between voltage and current may be considered a class *resistor* corresponding to the element type resistor. A copy of that class resistor with given actual values for the parameters in the constitutive relation called for instance R5 corresponds to a particular resistor in a circuit.

2.2 Model hierarchy

As is well known, a model of a system may be composed of lower level submodels which in turn may contain submodels as well; i.e. physical-system models are *hierarchical* in nature.

2.3 Model classes and instantiation

As explained above, physical system models and submodels of components can viewed as particular objects of a certain class. In OOM the term *instantiation* is adopted from OOP. A model or submodel is called an *instance* of a model class. Models or submodels are instantiated from generic models or model classes in which more general properties common to a class of models are described. The members of a model class have the same structure and exhibit the same general dynamic behaviour, e.g. an instance of the model class *diode* is obtained by giving particular values to its parameters. The resulting instance corresponds to a particular diode in a circuit.

2.4 Inheritance

If a submodel class is instantiated into a particular submodel, its properties are inherited by the submodel; i.e. the particular submodel definition is a special version of the more general submodel definition. In Section 4.2 an incomplete model class is defined that captures just the property of being a one-port store. In that class the state and the rate variable are not specified. By adding this information a particular class may be derived. The more special model class inherits all properties of the class from which it is derived. Consequently, it describes a particular type of a store. Again, by specifying the parameters a particular object or instance of that class is obtained that is associated with a particular store in the system. Another example is an incomplete model class *passiveOnePort* for the characteristic of being a passive one-port element (cf. Section 4.2). A special class *diode* may be obtained from the incomplete general class by specifying the constitutive relation as that of a diode. Obviously, the subclass inherits the characteristic of being a passive one port from the superclass. The advantage of using inheritance in the definition of library model classes is that the potential for errors in the code is reduced. All features of the superclass are copied. Only features that characterize the more special class need to be added.

2.5 Encapsulation

Knowledge contained in a model is encapsulated. Only a well-defined part of it may be accessed in a well-defined manner via interfaces to the world outside an object. This means that the internal definition of a submodel is not affected by the connection of the submodel to other submodels. That part of a submodel describing its interfaces is separated from the part in which the behaviour is described by means of non-causal mathematical equations. The latter do not need to be known when submodels are connected in order to build a hierarchical model. In Modelica all information about an interface of a submodel is encapsulated in a special class for which the keyword **connector** is used, while the equations of the submodel are collected in a section starting with the keyword **equation**. In generalized networks an interface of a submodel is a pin. It is described by two power variables called *across* and *through* variables.

2.6 Polymorphism

In a submodel definition the description of its interfaces to the outside world is separated from the internal description of its dynamic behaviour. In this internal description, multiple possible cases may be taken into account. Consequently, depending on actual conditions, the same submodel may show different behaviours.

2.7 Connection of submodels

Submodels are connected according to the physical structure of the system. Figure 1 shows an example in which the component models of a hydraulic system are interconnected according to the physical structure (see Fig. 10). The full squares denote the submodel interfaces to the outside world. Inheritance implies that knowledge already available can be reused, allowing for a more safe development of complex models. Only those properties of a submodel that are particular to the submodel need to be specified. Other properties that a submodel shares with others are described in a common superclass and this information is used without any modification.

Technical components are interconnected by electrical wires, mechanical shafts and hydraulic pipes, while physical processes interact by exchanging energy, mass and/or information. In both cases there is a mutual influence between the objects. An essential consequence of an interconnection according to the physical structure is that equations in the internal description of the dynamic behaviour must be *non-causal* or *declarative*. It cannot be decided *a priori* which variables in an equation are independent and which are dependent. Such a distinction rather is determined by the actual environment in which a model is embedded, i.e. by the submodels that it is connected to. Consequently, connecting models according to the physical structure in general requires *symbolic reformulation of equations* in order to transform the model description into a *simulation model* consisting of an ordered set of equations.

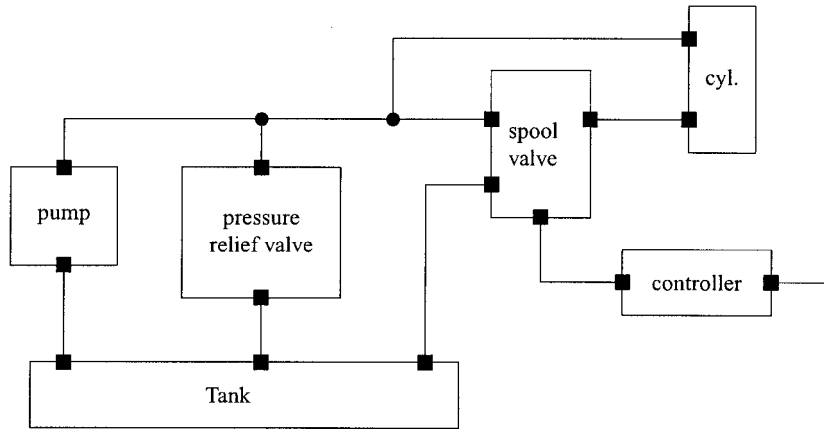

Fig. 1 Top-level representation of an object-oriented model of a hydraulic drive (cf. Fig. 10)

3 BOND GRAPH MODELLING FROM AN OBJECT-ORIENTED MODELLING PERSPECTIVE

Introduced by H. Paynter in 1959 and subsequently elaborated into a methodology by Karnopp and Rosenberg [**24**], bond graph modelling is much older than OOM and has not been conceived as a special form of it. Nevertheless, the characteristics listed and explained above can be found in bond graph modelling as well.

3.1 Objects

Although the notion of an object is not a keyword of the bond graph language, models of technical components, of basic physical processes and of their interconnections are the major subjects of bond graph modelling. From a modern point of view these models represented by the nodes in a bond graph may be considered objects.

3.2 Model hierarchy

Hierarchical structuring has proved an appropriate means in many formalisms to cope with large-scale systems. It is used in iconic diagrams, networks, block diagrams and bond graphs as well. On the hierarchy levels above the level of elementary submodels in bond graphs, submodels are represented by words (encircled by an ellipse). Such hierarchical bond graphs are called *word bond graphs.*

Elementary submodels are described by means of equations. A submodel above the level of elementary submodels is described by a bond graph or may also be given by a set of equations as well. In that case it is a behavioural model with no internal structure. In any case, submodels do have ports to the outside world. Word bond graphs do support a systematic top-down design of a structured model of a complex system guided by the consideration of the energy exchange between system components. Since energy exchange is mostly bound to real physical connections, e.g. electrical wires, mechanical shafts or hydraulic conduits, bond graphs clearly reflect the physical structure of a system before they are simplified. Furthermore, since bonds connecting power ports can carry information about the reference direction of the energy flow and also information about computational causality, word bond graphs are not merely iconic diagrams with icons reduced to an alphanumeric mnemonic code. (Word) bond graphs may be regarded rather as an intermediate format between an iconic diagram with application-specific icons and a mathematical model of the system.

3.3 Model classes and instantiation

On the lowest hierarchy level, elementary models are grouped into classes according to the type of the fundamental energy mechanism that they describe, i.e. there is a classification into energy storage, power-conserving distribution or transport of energy, transduction of energy into a different form, and transformation into heat in particular. The type (class) is denoted by a reserved symbol. A node in a bond graph representing a basic process may be viewed as an instantiation of its corresponding generic model. It is a special member of the class that it belongs to. Its constitutive equations and parameters characterize the particular submodel and distinguish it from other instantiations of the generic model. If a vertex of a bond graph represents a component model, the latter most often is an instantiation of a model class from a library augmented by specific parameters.

3.4 Inheritance

The process of instantiation, i.e. of copying and adapting a model class, implies that properties of the superclass are inherited by the instantiated model. For instance, the

generic model of a store captures the fundamental properties of passivity, of storing a physical quantity, e.g. electrical charge, and of being energy conservative. In regard to the constitutive equations it is only determined which variables are involved. Moreover, a preferred causality may be assigned, i.e. just the constraint that one of the two conjugate power variables at a port is related to the integral of the other one with respect to time. The actual functional (linear or non-linear) dependence, however, as well as (additional) parameters are specified in the model of the store under consideration.

3.5 Encapsulation

Since a bond graph model can be accessed only via its interfaces which are power ports and signal ports, the principle of encapsulation of knowledge also holds for bond graph models. The model of a store for instance should not pass the value of its state variable via its ports to the ports of incident vertices, i.e. the state variable represents encapsulated information. However, the state variable can be inherited by the model of a store from a superclass capturing the energy property of stores (see Section 4.2). Moreover, since computational causality at the power ports of a submodel depends on the connections of a submodel to other submodels, the internal description of its dynamic behaviour must be non-causal, i.e. for example independent of its connection to a port of another submodel the constitutive equation of a (non-linear) one-port C element may be specified in the form:

$$\dot{q} = f \tag{1a}$$

$$q = \Phi_C(e) \tag{1b}$$

In the case when the effort e must be the output of the store, an inversion of the function Φ_C is required (*integral causality*):

$$e = \Phi_C^{-1}\left[q(t=0) + \int_0^t f(\tau)\,\mathrm{d}\tau\right] \tag{2}$$

Otherwise, if the causality assignment procedure used determines the flow variable f to be the output of the store, differentiation of the function Φ_C with respect to time is required (*derivative causality*):

$$f = \frac{\mathrm{d}}{\mathrm{d}t}\Phi_C(e) = \frac{\partial \Phi_C}{\partial e}\dot{e} \tag{3}$$

Note that, in order to be able to adjust the constitutive equation according to the computational causality assigned, the function Φ_C must be bijective and sufficient smooth.

3.6 Polymorphism

Polymorphism as explained above can also be used in the definition of bond graph models.

3.7 Connection of submodels

It is a characteristic of bond graphs that submodels are connected together according to the way that their corresponding components exchange energy with others. Therefore, there is a strong topological affinity between the structure of a bond graph that has not been simplified and the physical structure of the system.

4 DESCRIBING BOND GRAPH MODELS IN Modelica

A model description language that has been particularly designed to support bond-graph-based physical system modelling is SIDOPS+ [**5**]. While it is the underlying textual language of the integrated modelling and simulation environment 20-sim [**25**], the OOM language Modelica has emerged from an international effort to combine features of present OOM languages, to promote the exchange and reuse of models and in that way to establish a *de facto* standard that might replace the old CSSL standard from 1967 [**26**]. Because of the increasing attention that Modelica is receiving in various application areas as a powerful general-purpose OOM language, and because Modelica represents a class of present OOM languages, in the following it is used as a working language. The description of the basic bond graph elements and of their interconnection in a textual OOM language such as Dymola or Modelica is quite straightforward [**27**–**29**]. However, while there are features in Dymola and Modelica that are useful for describing bond graph models, there are also some limitations. The following section focuses on three aspects.

4.1 Description of bond graph power ports and their interconnection

Since Modelica relies on the concept of generalized networks in regard to the modelling of energy flows, submodel interfaces to the outside world are *pins* and not ports as in bond graphs. The power variables of an interface are called across and through variables [**6**]. They correspond to *efforts* and *flows* as used by bond graph modellers. All information needed for the description of an interface is encapsulated in a special model class called **connector**. Contrary to bond graph ports, connectors in Modelica in addition to power variables may pass also other quantities such as generalized displacements and/or accelerations. Moreover, since Modelica is network oriented, together with the connection of two interfaces, normally the flows involved are summed up to zero. This is not appropriate for bond graphs because separate from the connection of power ports by bonds the summation of flows is carried out in a special element, the 0-junction. However, Modelica

```
connector PowerPort      "bond graph power port"
// Both effort and flow are treated as across variables
   Real e "effort variable";
   Real f "flow variable";
end PowerPort;
```

Fig. 2 Description of a bond graph power port in Modelica

allows for suppressing this summation of flow (or through) variables by declaring both conjugate power variables as efforts (Fig. 2), or, in other words, by omitting the keyword **flow** usually associated with the declaration of the flow variables of an interface. This keyword means that with the connection of two interfaces the flow variables are summed up to zero. If the prefix **flow** is missing, corresponding power variables of the two power ports connected together are just set equal. If M*i*.A denotes port A of submodel M_i ($i = 1, 2$), then the connection of the two power ports M1.A and M2.A is expressed by the statement

connect(M1.A, M2.A)

As the statement represents a port-to-port connection, it may be used to describe the bonds in a bond graph.

4.1.1 0- and 1-junctions

For interconnection of more than two power ports, bond graphs provide two types of junction corresponding to Kirchhoff's current law and voltage law respectively. Both types of junction allow for a finite but *a priori* undetermined number of ports. Based on the concept of generalized networks the OOM language Dymola provides an element called a *node* with an undetermined number of interfaces. It corresponds to a 0-junction in bond graphs. However, a built-in model that corresponds to a bond graph 1-junction and allows for an undetermined number of ports is not available. (An element that represents Kirchhoff's voltage law generalized to across variables is not needed in generalized networks.) The reason for this lack in flexibility in regard to the support of bond graphs is that neither Dymola nor Modelica model classes are conceived to have a class parameter as in SIDOPS, the predecessor of SIDOPS+. A SIDOPS model class of a 1-junction (Fig. 3) with the number N of ports as a formal parameter is given in Fig. 4. In the **interface** section causality constraints at the ports may be specified. **constraint 1_effort** means that only at one of the N ports can the effort variable be the output variable. The variable out denotes the flow that

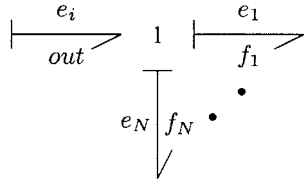

Fig. 3 Bond graph 1-junction

```
class one1 (N) version 1
# one1: 1-junction with N one-dimensional bonds attached
# N: number of power ports a-priori not determined
interface
   ports: P[N]
      causality restrictions
         constraint 1_effort P
   outputs: real out
equations
   sum ( P[].e )      = 0
   identity ( P[].f )  = out
```

Fig. 4 SIDOPS model class of a 1-junction with the number of ports as a formal parameter

enters at one port and is the output variable at all remaining $N-1$ ports. During model processing, the equation

$$\text{sum}(P[].e) = 0$$

in the **equations** section in Fig. 4 is expanded into

$$e_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} \alpha_j e_j$$

and

$$\text{identity}(P[].f) = \text{out}$$

is translated into

$$f_j = f_i, \qquad j = 1, \ldots, N, \quad j \neq i$$

$$\text{out} = f_i$$

[**30**]. Note that the value of N is now determined. According to the power orientation of the incident bonds the coefficients α_j are equal to either $+1$ or -1.

One way to bypass the problem of a missing element corresponding to a 1-junction with an undetermined number of ports is to replace a 1-junction by a 0-junction and to switch the role of the power variables at each incident bond by means of a so-called *symplectic gyrator* (modulus $r = 1$) as has been proposed by Cellier for the language Dymola [**27**]. Alternatively, a model class may be introduced that depends on the number of incident bonds; i.e. for each number of power ports a model class is defined which may be stored in a library [**28**]. This needs to be done also in Modelica version 1.0 for 0-junctions and 1-junctions. As an example the Modelica description of a 3-port 1-junction is depicted in Fig. 5. In the **equation** part of the model One3P **direction** is a predefined function that can be used to account for the orientation of a bond. If the bond starts from port1 and ends at port2, it may be specified by the statement **connect**(port1, port2). Then **direction**(port1) returns the value -1, and **direction**(port2) gives the value 1.

As has been mentioned above, in the definition of a connector any variable appropriate for the description of a connection of that submodel interface to another may be used; i.e. not only power variables but also for

```
model One3P  "3-port 1-junction"
  PowerPort port1, port2, port3;
equation
  // all flows are equal
  port2.f  =  port1.f;
  port3.f  =  port1.f;
  // all efforts sum up to zero
  0  =  direction(port1) * port1.e  +
        direction(port2) * port2.e  +
        direction(port3) * port3.e;
end One3P;
```

Fig. 5 Modelica description of a three-port 1-junction

example generalized displacements and accelerations may be passed from one submodel to another. With this flexibility in the modelling language it is convenient to provide information needed for evaluation of model equations. On the other hand it may obscure the expressiveness of a graphical model description, since there is no clear distinction between power and signal ports. Consequently, it may not be fully clear which information is passed from which submodel. Moreover, by using information passed in addition to the power variables it is possible to use (*ad hoc*) constitutive equations in a submodel which are not in accordance with physical conservation laws for energy or momentum, for example. If a connector does not pass both power variables, power continuity is not ensured by the interconnection mechanism. In that case, in addition to the evaluation of the constitutive equations, power must be calculated inside a submodel and passed to submodels connected to it. Power flowing into and out of a submodel must be checked. In contrast, bond graph modelling is more stringent. Power is passed through power ports, and signals through signal ports. It is an essential feature of bond graphs that power continuity is expressed explicitly. If signal modulation of elements is used with care, standard bond graph elements and the way that they are allowed to pass information help to avoid setting up model equations that may violate the energy balance. From these observations it seems that not every description of a dynamic system in Modelica may be translated into a bond graph. The description of a bond graph model in Modelica is feasible, although the fundamental 1- and 0-junctions with an undetermined number of ports so far are not appropriately supported. The next section illustrates how encapsulation and inheritance can be exploited even in the definition of the basic bond graph models.

4.2 Encapsulation and inheritance

In describing the basic bond graph elements in Modelica the object-oriented features encapsulation and inheritance may be exploited for a safe development of submodels to be stored in a library. For instance, both resistors and stores do have the property of being passive. This property can be encapsulated in a superclass and inherited by the definition of an actual submodel. Since inheritance means no rewriting or modification of parts of a model definition, this strategy avoids inconsistencies. This advantage of a textual OOM language is illustrated by confining the presentation to one-port elements. Passivity of resistors and stores is taken into account in bond graphs by the convention of an inward positive reference direction of the energy flow at the bond incident to the element port. (Power flows into the port when the product of both power variables e and f is positive.) A Modelica class capturing the property of a one-port element being passive is shown in Fig. 6. The definition of a model or submodel class starts with the keyword **model**. Since constitutive equations are missing, the model class `passiveOnePort` cannot be instantiated into the model of a passive element. This is indicated in Modelica by the keyword **partial**. In the definition given in Fig. 6 the word **assert** denotes a predefined function in Modelica that evaluates a Boolean expression and returns an error message in case the value of the expression is false, while the predefined function **cardinality** returns the number of connections to a connector p. The superclass `passiveOnePort` may be used to introduce a subclass `energeticOnePort` in which the property of a store being energetic is encapsulated (Fig. 7). The still incomplete model class `energeticOnePort` inherits the properties of the superclass `passiveOnePort` which is expressed by the clause **extends**. The keyword corresponds to **inherit** in the language Dymola. The property being energetic can be inherited by the definition of a store. According to the type of the store (either a C or an I element) the internal variables rate and state must be related to the power port variables and the constitutive law of the store must be expressed. As an example the Modelica

```
partial model passiveOnePort
  PowerPort p;
equation
  // ensure that the number of bonds is one
  assert(cardinality(p) == 1);
  // ensure that the orientation of the power flow is inward
  assert(direction(p) == 1);
end passiveOnePort;
```

Fig. 6 Encapsulation of the property being passive for one-port elements

```
partial model energeticOnePort
  extends passiveOnePort;
  Real state, rate;
equation
  // der(state): time derivative of state
  der(state) = rate;
end energeticOnePort;
```

Fig. 7 Encapsulation of the property being energetic for one-port stores

```
model progressiveSpring
// non-linear one-port C-element
   extends energeticOnePort;
   parameter Real k = 1.0 "spring stiffness";
equation
   rate  = p.f;
   state = k * (p.e)^3 ;
end progressiveSpring;
```

Fig. 8 Modelica description of a one-port C store using inheritance

description of a one-port C store representing a mechanical non-linear spring is shown in Fig. 8.

4.3 Computational causality

An essential consequence of the connection of submodels according to the physical structure, in which the corresponding system components are connected, is that model equations must be non-causal. In modelling languages such as Dymola, Modelica or SIDOPS+, all equations are declarative unless explicitly expressed as assignment statements. Processing of such models, i.e. transforming an initially large and redundant set of differential algebraic equations (DAEs) into a smaller set of appropriate form, which allows for an efficient solution, in general requires the symbolic reformulation of some of the equations or even the symbolic solution of small linear subsystems. Bond graph modelling does support the generation of equation systems in a form that can be efficiently solved by providing the concept of *computational causality*; i.e. there are rules for assigning computational causality to the power ports of the elements and for propagation of causalities through the junction structure of a bond graph. These rules impose constraints on the choice of port causalities. Unlike the modelling language SIDOPS+ which relies on the bond graph methodology, causality constraints unfortunately cannot be expressed as attributes of the interface elements in Dymola or in Modelica, since the model processors of both languages do not exploit information about causal constraints during the generation of equations.

On the other hand, computational causality of switches depend on their switch state. If switches are used in a bond graph, the concept of causality loses some of its virtue. This has given rise to several proposals in the literature as to how to model so-called hybrid systems properly in a bond graph framework. In a textual modelling language such as Dymola or Modelica, *variable causality* can be easily expressed as shown in Fig. 9.

```
model IdealSwitch    "ideal switch"
   PowerPort p;
   input switchstate;
equation
   0 = if switchstate then p.f else p.e;
end IdealSwitch;
```

Fig. 9 Description of an ideal switch in Modelica

5 DESCRIPTION OF COMPOSED BOND GRAPH MODELS

Once the basic bond graph elements have been described in Modelica and stored in a bond graph library, it is an obvious step to use them for the description of bond graph models of standard components in an application area and to store those descriptions in a library for that application area. Normally a number of models of different complexities are provided for standard components, so accounting for different aspects. This is a way to make models as accurate as needed in a context under consideration and to keep them as simple as possible at the same time. If the library is just a file or a set of several files containing the Modelica code, good documentation is required in order to enable the designer to select an appropriate model for the actual design task. A better user support is to provide a librarian procedure which accepts functional model specifications and allows for navigation through the library to find a model that meets given requirements. For hydraulic systems the description of bond graph models of standard hydraulic devices in Modelica and a library still under development have been briefly presented in reference [**31**]. The way in which composed bond graph models of mechatronic systems can be described in Modelica in a systematic step-by-step manner is roughly illustrated by the example of a controlled hydraulic drive given in Fig. 10. The mechanical load connected to the cylinder shaft may be an application-specific complex mechanism and, therefore, has not been specified in this example. The signal into the controller is generated from signals provided by sensors which monitor the dynamic behaviour of the mechanical load.

A reasonable bond graph model of the hydraulic pump in that example is shown in Fig. 11. It accounts for the transformation of mechanical into hydraulic power by means of the power-conserving two-port transformer. Moreover, it takes into account the compressibility of the fluid in the outlet port by means of a C store attached to the 0-junction and accounts for internal leakage by means of the R element. If a signal input port is added and if the transformer is allowed to be a displacement modulated transformer (MTF) as in Fig. 11, the model may even represent a variable displacement pump controlled by the angle α of inclination of the swashplate.

Figure 12 shows the Modelica description of a model of the hydraulic resistor in Fig. 11 which inherits the model class `passiveOnePort`. A Modelica description of the pump model in Fig. 11 is given in Fig. 13. Now suppose that bond graph models have been developed for standard hydraulic components, described in Modelica and stored in a library. Moreover, assume that

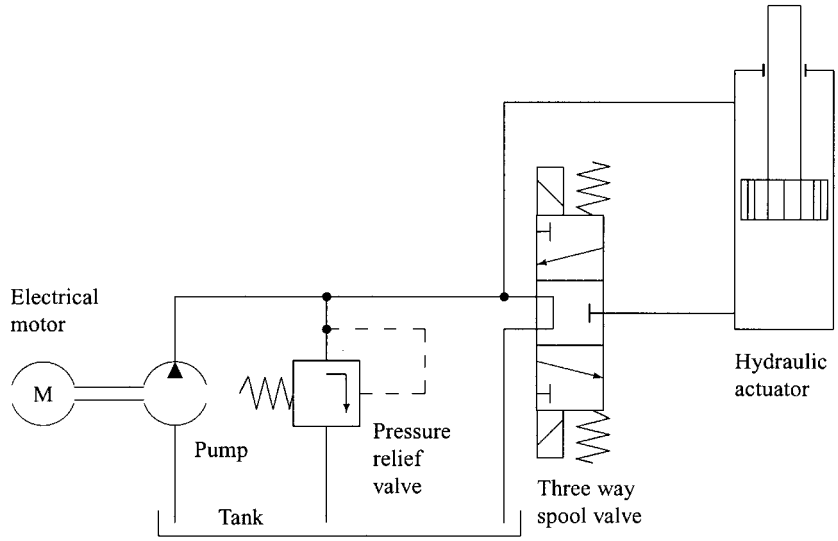

Fig. 10 Circuit schematic diagram of a hydraulic drive

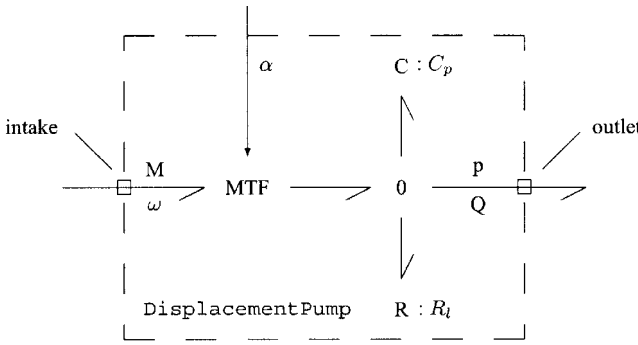

Fig. 11 Bond graph model of a (variable)displacement pump

bond graph models described in Modelica are also available for electromechanical devices such as electric motors and that a bond graph model for the application-specific mechanical load of the hydraulic drive has been composed of lower-level bond graph models. Then it is sufficient to translate the system schematic into a composed bond graph such that for each submodel a Modelica description is available from a library. Figure 14 shows a word bond graph of the controlled hydraulic drive. The word bond graph closely corresponds to the physical structure of the system as depicted in the schematic. A model of the tank and all bonds corresponding to the return lines are missing in the word bond graph because the return pressure is usually chosen as a reference. Consequently, the 0-junction representing the return pressure and all incident bonds are eliminated. The word bond graph is not just another graphical representation but a first step from a schematic representation towards a non-causal mathematical model that can be carried out automatically. A word bond graph may be viewed as an intermediate format between a schematic diagram that uses domain specific icons and a mathematical model. A textual object-oriented description of such a word bond graph is straightforward and rather intuitive in a modelling language such as Modelica. The result is given in Fig. 15. As in the Modelica description of the bond graph of the variable-displacement pump, first all submodels involved are listed. Each submodel is instantiated from a model class which is given in the first column. The second part with the heading **equation** describes the connectivity of the submodels so that for each power bond and the low-power signal connection between the controller and the spool valve there is a corresponding **connect** statement. What still remains to be described are the sensors and

```
model Orifice    "hydraulic orifice"
  extends passiveOnePort;
  constant Real rho (final unit="kg/m^3") = 0.85e+3 "fluid density";
  parameter Real alpha=0.61 "flow coefficient (turbulent flow conditions)";
  parameter Real area "cross section of orifice";
  Real Q "volume flow rate";
  Real dp "pressure difference across the orifice";
equation
  Q    = p.f;
  dp   = p.e;
  Q    = alpha * area * sqrt(2/rho * abs(dp)) * sign(dp);
end Orifice;
```

Fig. 12 Modelica description of a hydraulic orifice

```
model DisplacementPump

/* This bond graph model has
   - a mechanical power port labeled intake,
   - a hydraulic power port labeled outlet which provides the volume flow rate Q,
   - a signal input denoted by alpha controlling the angle of inclination of the swashplate
*/
   PowerPort intake, outlet ;
   input  Real   alpha;

/* Type and local name of submodels used */
   MTF          transformer ;
   zero4P       p ;          // 0-junction represents the load pressure p
   C            Cp ;         // accounts for fluid compressibility in the outlet port
   orifice      Rl ;         // accounts for losses due to internal leakage

/* connectivity of power ports according the bond graph */
equation
   transformer.signalIn = alpha ;
   connect(intake, transformer.PowerIn) ;
   connect(transformer.PowerOut, p.port1) ;
   connect(p.port2, Cp.port1) ;
   connect(p.port3, outlet) ;
   connect(p.port4, Rl.port1) ;
end DisplacementPump ;
```

Fig. 13 Modelica description of a bond graph model of a variable-displacement pump

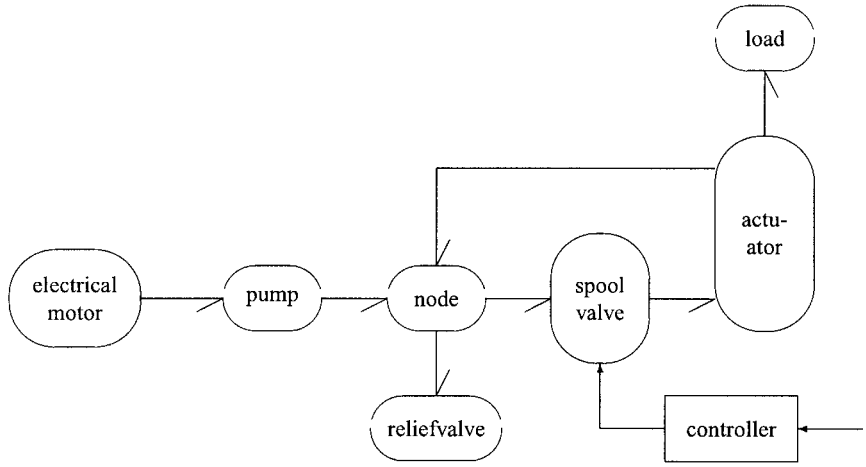

Fig. 14 Word bond graph of the controlled hydraulic drive

the input of the controller. If all submodels are available from a library, then the equations of the overall system can already be generated automatically by a model processor.

Of course, instead of developing bond graph models of standard hydraulic components and describing them in Modelica, object-oriented descriptions in Modelica can be developed directly for standard components which in turn can be combined together according to the structure of the top-level object-oriented model given in Fig. 1. However, this is not the objective of this paper. A bond graph model has the advantage that it clearly shows on a graphical level which effects have been taken into account. Modelling details can be easily added or removed without being concerned with the manipulation of equations. Finally, the description of bond graph models in an OOM language such as Modelica has the advantage that composed bond graph models of complex systems can be processed also by modelling and simulation packages that have not been designed for supporting the bond graph approach. The user even does not need to be familiar with bond graph modelling.

6 BOND GRAPHS AND VHDL-AMS

VHDL is a modelling language that focuses on the discrete description of digital electronic hardware. Its extension VHDL-AMS also allows analogue subsystems to be modelled and signal flow descriptions to be included on a functional level so that VHDL-AMS can be used as a unified modelling language for combined time-

```
model HydraulicDrive
// Submodels of all system components
   ElectricalMotor      electricalmotor;
   ConstantFlowPump     pump;
   ReliefValve          reliefvalve;
   ThreeWayValve        spoolvalve;
   DifferentialCylinder actuator;
   Zero4P               node;
   ControllerType       controller;
   MechLoad             load;
// Connectivity of ports according to the schematic
equation
   connect(electricalmotor.mech, pump.mech);
   connect(pump.hydr, node.port1);
   connect(node.port3, reliefvalve.port1);
   connect(node.port2, spoolvalve.P);
   connect(node.port4, actuator.B);
   connect(spoolvalve.A, actuator.A);
   connect(actuator.mech, load.port1);
   connect(controller.out, spoolvalve.control);
end HydraulicDrive;
```

Fig. 15 Modelica description of the word bond graph

discrete and time-continuous systems. Component models are described in a so-called `entity` section and an associated separate `architecture` section. In the architecture section the behaviour of an analogue subsystem is described by a set of non-causal equations. The equality of the left-hand and the right-hand sides of an equation is emphasized by the `==` operator. The designers of the language extension use the notion of *simultaneous statements*. Variables that are a continuous function of time or frequency are called *quantities*. In an `entity` section the interfaces of an object to the outside world are described. It also includes the parameters of the submodel. There are different types of interface depending on whether connections between interfaces model an energy flow or a signal flow. For modelling energy flows, VHDL-AMS uses the concept of generalized networks. In that context, connection points are called *interface terminals*. The variables associated with a terminal are called *across* and *through quantities*. Their type is defined in what is called a *nature* (Fig. 16). A VHDL-AMS nature hence represents the *energy domain* in which the power port variables are used. For illustration purpose a hydraulic orifice represented by a two-pin resistor modulated by the displacement x of a valve spool is described in VHDL-AMS in Fig. 17.

Using an interface terminal means that Kirchhoff's laws are enforced for the associated power variables. In that context the notion of so-called *conserved* systems is used, while systems modelled on the level of signal flows

```
– Translational mechanics
subtype velocity is real;
subtype force    is real;
nature translational is
   velocity across force through;
```

Fig. 16 VHDL-AMS nature *translational*

```
entity orifice is
   port (
      terminal np, nm : hydraulic;
      quantity x : in real);
end entity orifice

architecture MR of orifice is
   constant rho: real := 0.85; – density of liquid [g/cm^3]
   constant alpha : real := 0.6;
   quantity dp across Q through np to nm;
   quantity A real; – variable cross section area
   begin
      A == function of displacement x;
      Q**2 == alpha* * 2 * A* * 2 *2/rho * abs(dp)
   end
```

Fig. 17 VHDL-AMS description of a displacement-modulated hydraulic orifice

are called *non-conservative*. This means that, in regard to modelling energy flows, VHDL is very much linked to networks. Contrary to bond graphs the connection of interfaces is not separated from Kirchhoff's laws generalized to through variables. This is unsuitable for describing bond graphs. In bond graphs the power bonds represent point-to-point connections between power ports while Kirchhoff's current and voltage laws generalized to power variables called efforts and flows are represented by two dedicated nodes. Consequently, although VHDL-AMS extends to modelling of non-electrical continuous systems and although it is expected to receive wide acceptance and support, unfortunately the language does not seem to be general enough to support the bond graph formalism also. Unlike Modelica it does not appear to be a suitable exchange format for bond graph models.

7 CONCLUSIONS

Although bond-graph-based physical system modelling is much older than OOM, both approaches have much in common:

1. Both paradigms are *domain independent*.
2. They both support *hierarchical* modelling.
3. On the graphical level, submodels are connected according to the physical structure of a system to be designed. This implies that equations in submodels must be non-causal.

A closer look reveals that the features of OOM adopted from OOP in software engineering, e.g. inheritance, may be identified also in bond graph modelling, although the notions of object, encapsulation and inheritance have not been used in the bond graph language. In fact, from a present-day point of view, bond graph modelling may be considered a special form of object-oriented physical system modelling (see also reference [**25**]). A major difference is that OOM has adopted

the concept of generalized networks for the description of energy flows. Hence, textual modelling languages such as Dymola and Modelica, which support the OOM approach, assume that energy flows are modelled by means of generalized networks. On the other hand, Modelica has evolved from an international standardization effort to combine features of present OOM languages and to promote the exchange and reuse of engineering models. Therefore, and because of the relations between bond graph modelling and OOM, it appears to be attractive to use a language such as Modelica also for the description of bond graph models. In that regard the following observations can be summarized:

1. It is possible to describe (hierarchical) bond graph models in Modelica because, in contrast with VHDL-AMS, the summation of through variables can be separated from the connection of submodel interfaces. The **connect** statement can be used in such way that it corresponds to an (oriented) bond in the bond graph.
2. The concept of submodel interfaces is less stringent than that of power ports in bond graphs. Consequently, given the flexibility of Modelica, it seems that not every Modelica description of a system can be translated into a combination of a bond graph and a block diagram.
3. Built-in elements corresponding to 0- and 1-junctions with a number of ports that is not fixed *a priori* are not available in Modelica. Since, in contrast with SIDOPS+, model classes cannot have a class parameter, models for both types of junction must be defined that depend on the number of ports.
4. Unlike SIDOPS+ it is not possible to specify causality constraints as attributes of power ports. Information about causal constraints are not used by the Modelica model processor. On the other hand, since equations are treated as non-causal, bond graph submodels may be described independently of the causalities that a submodel is assigned at its power ports due to its connections with other submodel ports.
5. Switches and consequently systems of variable structure can be described more easily in Modelica than in bond graphs. For such systems the concept of computational causalities in bond graph modelling loses some of its virtue.
6. The object-oriented principles of encapsulation and inheritance prove useful even in the description of the basic bond graph elements.

Even though Modelica is based on generalized networks in regard to the description of energy flows and although there are some limitations, it is fairly straightforward to describe bond graph models in Modelica and to build libraries of bond graph models described in Modelica, as has been shown in this paper. While Modelica seems to become an increasingly accepted neutral exchange format, for which proprietary modelling and simulation packages are going to provide export and import filters [**25**], the advantage for bond graph models is that it becomes easier to use them in different modelling and simulation packages, even in those which have not been designed to support the bond graph modelling formalism; i.e., although Modelica is not the best-suited exchange format for bond graph models, bond graph modellers can benefit from this new exchange format. Apart from the use of Modelica for representing bond graph models an increasing number of publications, industrial applications, software tools, workshops and seminars clearly demonstrate the interest in academia and industry in the modern free general-purpose modelling language Modelica which supports multiple formalisms and might replace the old CSSL standard in the future.

In those cases in which it is appropriate to add submodels to a collection of predefined library models the language SIDOPS+ underlying the modelling and simulation package 20-sim appears to be most suitable for bond graph modelling. In regard to the increasing use of Modelica in academia and industry a translator from SIDOPS+ to Modelica would be desirable.

Concerning VHDL-AMS at least at present it does not seem to be an alternative to Modelica for bond graph modellers. Like Modelica, VHDL-AMS is closely linked to the concept of generalized networks in regard to the presentation of energy flows. If the language would allow separation of the interconnection of interface terminals from Kirchhoff's current law generalized to through variables and the definition instead of explicitly interconnection elements corresponding to bond graph 0- and 1-junctions, it appears that VHDL-AMS could be used to represent hierarchical bond graph models as well. For an effective support of bond graph modelling, a built-in 1-junction model allowing for a number of ports which are not fixed would be needed as in Modelica. Since VHDL-AMS has been defined as a new IEEE standard, it will certainly play a major role in industry and in academia in the near future, particularly for the description of mixed digital–analogue systems and of microsystems, and will be supported by several (proprietary) system simulation packages.

REFERENCES

1 **Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F.** and **Lorensen, W.** *Object-Oriented Modeling and Design*, 1991 (Prentice-Hall, Englewood Cliffs, New Jersey).
2 **Elmqvist, H., Brück, D.** and **Otter, M.** *Dymola—User's Manual*, 1996 (Dynasim AB, Lund); see also http://www.Dynasim.se.
3 **Andersson, M.** Omola—an object oriented language for model representation. Master's thesis (Licentiate thesis

TFRT-3208), Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1990.

4 **Jeandel, A. F., Boudard, F., Ravier, Ph.** and **Bushing, A.** ULM: 'un langage de modélisation': a modelling language. In Proceedings of the IMACS–IEEE Multiconference on *Computational Engineering in Systems Applications* (*CESA'96*), Lille, France, July 1996, pp. 781–789 (IMACS).

5 **Breunese, A. P. J.** and **Broenink, J. F.** Modeling mechatronic systems using the SIDOPS+ language. In Proceedings of the 1997 International Conference on *Bond Graph Modeling and Simulation* (*ICBGM'97*), Simulation Series, Vol. 29, No. 1 (Eds J. J. Granda and G. Dauphin-Tanguy), 1997, pp. 301–306 (Society for Computer Simulation, San Diego, California).

6 **Elmqvist, H.,** *et al.* Modelica™—a unified object-oriented language for physical systems modeling; http://www.Modelica.org.

7 **Tiller, M.** *Introduction to Physical Modeling with Modelica*, 2001 (Kluwer, Boston, Massachusetts).

8 Non Nomen. Standard VHDL analog and mixed-signal extensions. Technical Report IEEE 1076.1, IEEE, March 1997; see also http://www.vhdl.org/analog/ and http://www.vhdl-ams.com/.

9 **MacFarlane, A. G. J.** *Engineering Systems Analysis*, 1964 (Harrap, London).

10 Analogy Inc., http://www.analogy.com/Pub/VeriasHDL.htm.

11 Mentor Graphics Corporation, http://www.mentor.com/ams/adms.html.

12 SIMEC, http://www.hamster-ams.com.

13 **Breedveld, P. C.** and **Dauphin-Tanguy, G.** (Eds) An epistemic prehistory of bond graphs. In *Bond Graphs for Engineers*, 1992, pp. 3–17 (North-Holland, Amsterdam).

14 **Karnopp, D. C., Margolis, D. L.** and **Rosenberg, R. C.** *System Dynamics—Modeling and Simulation of Mechatronic Systems*, 3rd edition, 2000 (John Wiley, New York).

15 **Mukherjee, A.** and **Karmakar, R.** *Modelling and Simulation of Engineering Systems through Bondgraphs*, 2000 (Narosa Publishing House, New Delhi).

16 **Borutzky, W.** *Bond Graphs—A Methodology for Modeling Multidisciplinary Dynamic Systems, Frontiers in Simulation* (in German), 2000 (Society for Computer Simulation European Publishing House, Erlangen, Ghent).

17 **Dauphin-Tanguy, G.** *Les Bond Graphs*, 2000 (Hermes Science Europe Limited, Paris).

18 **Benzakki, J.** and **Djafri, B.** Object-oriented extensions to VHDL—the LaMI proposal. In Proceedings of Conference on *Computer Hardware Description Languages and Their Applications* (*CHDL'97*), Toledo, Spain, 1997, pp. 334–347.

19 **Borutzky, W.** Relations between bond graph and object-oriented physical systems modeling. In Proceedings of the 1999 International Conference on *Bond Graph Modeling and Simulation* (*ICBGM'99*), Simulation Series, Vol. 31, No. 1 (Eds J. J. Granda and F. E. Cellier), San Francisco, California, January 1999, pp. 11–17 (Society for Computer Simulation, San Diego, California).

20 **Borutzky, W.** Bond graph modeling from an object oriented modeling point of view. *Simulation Practice Theory*, 1999, **7**(5–6), 439–461.

21 **Mattsson, S. E.** Object-oriented modelling of a real continuous-time system. In Proceedings of the 1992 European Simulation Multiconference, June 1992, pp. 241–245 (Society for Computer Simulation, San Diego, California).

22 **Elmqvist, H., Cellier, F. E.** and **Otter, M.** Object-oriented modeling of hybrid systems. In Proceedings of 1993 European Simulation Symposium, Ghent, Belgium, 1993, pp. xxxi–xli.

23 **Andersson, M.** Object-oriented modeling and simulation of hybrid systems. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1994.

24 **Karnopp, D. C.** and **Rosenberg, R. C.** *Analysis and Simulation of Multiport Systems—The Bond Graph Approach to Physical System Dynamics*, 1968 (MIT Press, Cambridge, Massachusetts).

25 **Broenink, J. F.** 20-sim software for hierarchical bond-graph/block-diagram models. *Simulation Practice Theory*, 1999, **7**(5–6), 481–492.

26 **Strauss, J. C.,** *et al.* The sci continuous system simulation language (CSSL). *Simulation*, December 1967, 281–303.

27 **Cellier, F. E.** Hierarchical non-linear bond graphs: a unified methodology for modeling complex physical systems. *Simulation*, April 1992, **58**(4), 230–248

28 **Borutzky, W.** and **Cellier, F. E.** Tearing algebraic loops in bond graphs. *Trans. Soc. Computer Simulation*, June 1996, **13**(2), 102–115.

29 **Broenink, J. F.** Bond-graph modeling in Modelica. In *Simulation in Industry*, Proceedings of the 9th European Simulation Symposium (*ESS'97*) (Eds W. Hahn, A. Lehmann, W. Borutzky and H. Ziegler), Passau, Germany, 1997, pp. 137–141.

30 **Broenink, J. F.** Computer-aided physical-systems modeling and simulation: a bond-graph approach. PhD thesis, University of Twente, Enschede, The Netherlands, 1990.

31 **Borutzky, W., Barnard, B.** and **Thoma, J. U.** Describing bond graph models of hydraulic components in modelica. In Proceedings of the 3rd Mathmod Vienna IMACS Symposium on *Mathematical Modelling* (Eds I. Troch and F. Breitenecker), 2000, pp. 715–719 (Arbeitsgemeinschaft Simulation News).