Presented at the ICBGM '99, International Conference on Bond Graph Modeling and Simulation, part of the WMC '99, the Western MultiConference, San Francisco, CA, January 17-20 1999

Object–oriented modeling with bond graphs and Modelica

Jan F. Broenink
Control Laboratory, EE Department, University of Twente, P.O. Box 217, NL-7500 AE Enschede, Netherlands
phone: +31-53-489 2793; fax: +31-53-489 2223; e-mail J.F.Broenink@el.utwente.nl

Keywords: Bond graphs, Modelica, object-oriented modeling, software development, model libraries.

ABSTRACT

A new modeling language, called Modelica, for physical systems modeling is being developed in an international effort. The main objective is to make it easy to exchange models and model libraries. The design of Modelica builds on non-causal modeling and the use of object-oriented constructs stemming from modern software development, (hierarchy, encapsulation) to facilitate reuse of models and model parts.

Mapping bond graph models to Modelica code is in principle a straightforward process, especially since the Modelica specifications of the basic bond-graph elements are available, and Modelica accepts non-causal models.

In this paper, we will discuss an export filter from 20-SIM to Modelica. We use an application example to show results.

It turned out that Modelica is not the perfect exchange language for bond–graph models, but since Modelica is still under development, it might be better in the future.

1 INTRODUCTION

Modelica™ (Modelica 1997) is a *new* modeling language for describing the dynamic behavior of physical systems. It is inspired on the principle of object-oriented software development and *acausal* modeling, thus bond–graph like ports can be used as interface elements, and the equations need not be specified in a specific computational form. The main objective is to make it easy to exchange models and model libraries. Since Modelica accepts non-causal models, bond-graphs can be translated to Modelica code as submodels (i.e. a-causally).

Bond Graphs are a domain-independent graphical notion of physical systems modeling. During modeling, the edges in the graph denote the ideal exchange of energy between the submodels (vertices). One can state that bond-graph modeling is in fact a form of object-oriented physical systems modeling.

In this paper, besides a further elaboration on the differences in modeling paradigm between bond graphs and Modelica, we will discuss the bond–graph library in Modelica and an export filter from 20-SIM (Broenink 1997) to Modelica. This export filter is actually a translator from SIDOPS (the modeling language used in 20-SIM, suitable for bond graphs, block diagrams and equations) to Modelica.

An application example will also be discussed to show the applicability for the end user.

2 MODELICA

Modelica is a new modeling language for physical systems modeling that is being developed in an international effort (Modelica 1997). The main objective is to make it easy to exchange models and model libraries.

The design of Modelica builds on two relevant modern concepts in modeling and simulation, namely *non–causal modeling* and the use of *object-oriented constructs* (encapsulation, inheritance and hierarchy) originally used in software engineering (e.g. Rumbaugh et al. 1991) essential for physical systems modeling. Through these two concepts, reuse of models and submodels is facilitated:

- Non-causal modeling allows the internals of submodels completely be *encapsulated*. The submodel interfaces can be defined as pairs of variables which are *not* committed to an input or output role while defining the submodels. So submodel *use* is *not* constrained by the chosen formulation of its internal specification.
- Inheritance is the sharing of description parts among submodels based on a hierarchical relation ship. A generic submodel can be defined broadly and then refined into successively finer submodels. The common parts need to be specified only once.

There are already several modeling languages based on these ideas, and there is significant experience in using these languages in various applications. Examples include: ASCEND (Piela et al. 1991), Dymola (Elmqvist et al. 1996), gPROMS (Barton and Pantelides 1994), NMF (Sahlin et al. 1996), ObjectMath (Viklund and Fritzson 1995), Omola (Mattson et al. 1993), SIDOPS+ (Breunese and Broenink 1997), Smile (Kloas et al. 1995), ULM (Jeandel et al. 1996) and VHDL-AMS (IEEE 1997). The aim of the Modelica effort is to unify the concepts and to design a new uniform language for model representation.

Note that Modelica is a textual language, and as such more relevant for software tool builders than for the physical systems modelers. This is rather obvious, since Modelica is meant as a new standard for exchange of models, model libraries and experiments. Our experience is that graphically representing models as interconnected submodels displayed as icons supports their quick understanding. Furthermore, most contemporary tools have graphical model editing facilities.

2.1 Essentials of Modelica

Essential features of Modelica are the following:

Models and submodels are declared as *classes*, with interfaces that are called *connectors*. A connector must contain all quantities needed to describe the interaction. Attributes can be used to specify how the connections are converted to computable code. Modification of a model definition is possible using the *extends* construct. This way, for refinement of a generic submodel into a more specific one, only the 'new' specific parts need to be described. The common parts are inherited from the more generic submodel, and need to be specified only once.

Models can have submodels that can have submodels themselves. So, *hierarchical modeling* and *inheritance* are supported.

The equations of the models are described *non–causally*, i.e. in a *delcarative* style, as real equations in mathematical sense, not as a procedure for computation (assignment statements). So, the "="–symbol means *equality* holding for all values of *time*, the independent variable. Furthermore, it is possible to describe the connections in terms of physical connections, i.e. pairs of bilaterally computed power–conjugated variables. These pairs of interface variables are *not* committed to an input or output role while defining the submodels. So submodel *use* is *not* constrained by the chosen formulation of its internal specification. This implies that the internal description of a model (its equations) can be separated from the interface, i.e. when using such a submodel the *exact* contents of it need not be known. Consequently, symbolic equation rewriting and sorting is necessary to obtain simulatable code. This is taken care of by the Modelica compiler. With this facility, *encapsulation* is taken care of.

Furthermore, Modelica has a facility to specify how the connections are converted to computable code. Two different types of connection equations can be generated, namely:

- *Equality*: the matching variables of the interface elements being connected, are equal.
- *Sum–to–zero*: the matching variables of the interface elements being connected, sum to zero.

Which connection statement should be generated is specified at the declaration of the *connector*. Using the prefix *flow* causes a zero–to–sum equation be generated.An example is the connector *Pin* for electrical networks:

```
connector Pin
   voltage v;
   flow current i;
end Pin;
```

A connection *connect*(Pin1, Pin2), with Pin1 and Pin2 of connector Pin, generates 2 equations, namely Pin1.v = Pin2.v and Pin1.i + Pin2.i = 0.

The sum–to–zero equation is an implementation of Kirchhoff's current law generalized to all physical domains. This generalization is legitimate, as is done in bond graphs (summing equations at the 0– and 1–junctions). However, this equation is specified as a part of a connection, which makes that connection *not* ideal anymore. Combining such a zero–to–sum equation in a connection, implies that two different physical–system modeling concepts, namely *ideal connection* and a *Kirchhoff's current / voltage law* are mixed. Note that, both in bond graphs and in general graph theory, the edges between the vertices of a graph denote an *ideal* connection only. For the Kirchhoff's law equations, separate submodels (vertices) are used. Also from the point of view of object orientation, the combination of ideal connection and Kirchhoff's laws in *one* connect statement is not according the standard ideas.

Note that in the standard network–like descriptions in Modelica, one of the two variables in a connection has the tag *flow*, and thus will be summed to zero.

2.2 Language constructs

Some essential language constructs are presented here, by showing examples, taken from the Modelica bond–graph library (Broenink 1997b).

Connectors specify interface elements whereby an arbitrary amount of variables can be declared (see also section 2.1). A bond–graph port, being the bond–graph interface element, is defined as follows:

```
connector BondPort "Bond Graph power port"
  Real e    "Effort variable";
  Real f    "Flow variable";
end BondPort;
```

Note that for *both* variables the *equality* connect equations are generated.

Furthermore, ports can have restrictions on the orientation of the power flow and causality of the connected bond, which can be specified as extra attributes. These restrictions are used to check the validity of a connection of bonds to submodel ports, and can also be used to support automatic connection of bonds to submodel ports. For instance, passive elements (R, C, I) always have the power flow into the element, to ensure positive parameters. Causality restrictions are limitations on the causality of the power ports, imposed on by the submodel specification itself (either the submodel graph or the equations). For instance, the storage elements (C, I) have a *preferred causality constraint* to indicate that the integral form of the resulting equation is to be preferred over the differential form. Source elements (Se, Sf) have a *fixed causality constraint* and junction structure elements (0, 1, TF, GY) have a *constrained causality constraint*, i.e. the combination of ports gives a certain constraint on the possibilities of the causality, given by the equations the element represents.

This way of having extra attributes at the ports, let the properties of the submodel internals necessary for connecting the ports, be administered at the ports themselves. This contributes to the *encapsulation* of the submodel internals (either bond graph or equations). So, when using a submodel, only the interface elements (ports including attributes) need to be checked.

For restriction on the power orientation, the *direction* attribute of the Modelica connector is used. Unfortunately, there is *no* special attribute for *causality restrictions*. Note that such a causality restriction is *only* needed to check whether a given connection of a bond onto a port is valid.

A *signal* connector is specified in the same way. Now, the direction attribute can be used to specify the *signal* direction:

```
connector RealSignal " Signal port"
   Real s  "Real signal";
end RealSignal;
```

Models and *submodels* are specified using the *model* keyword on the first line. The tag *partial* is used to stress the fact that the model specification is incomplete, and must be completed in a specialisation. In combination with the *extends* clause, inheritance can be specified in a rather straightforward way. The general *OnePortPassive* class gets specialized via a *OnePortEnergetic* class to a bond–graph C–element, c1:

```
partial model OnePortPassive
                              "One port passive bond graph element"
   BondPort p           "Generic power port p"
end OnePortPassive;

partial model OnePortEnergetic
                           "One port storage element, being passive"
   extends OnePortPassive;
   Real  state          "Conserved quantity";
end OnePortEnergetic;
```

```
model c1 “Bond Graph C element”
    extends  OnePortEnergetic;
    parameter Real  c  “Capacitance”;
equation
    der(state) = p.f;
    p.e = state / c;
end c1;
```

The initial condition of the state variable can be specified as an *attribute* belonging to that state variable.

The *connect* statement is used to connect interface elements. It basically maps the corresponding variables of the first connector to the second connector. Tagging a variable in a connector *declaration* with *flow* causes a sum–to–zero equation be generated for the variables involved, instead of an equality (see section 2.1). Usually, in Modelica at least one variable gets a *flow* tag. So, besides an ideal connection, also some network equations (generalized Kirchhoff’s current law) are generated.

For bonds and signals we *do* use the *connect* statement. Since we do *not* use the *flow* tag in the connector *BondPort* and *RealSignal* (see the declaration above), the bond and signal connections are *ideal* connections. The generalized Kirchhoff laws are implemented by the *junction* models (0– and 1–junction, denoting a common effort, and summing the flows and the other way arond for the 1–junction.

In the connect statement, however, the causality of the bonds cannot be specified, because there is no data space available. To correctly generate the equations, the Modelica compiler treats all non–causal equations, including the connect equations, as *one* large set of equations, from which via symbolic manipulation the simulation model (i.e. a set of computable differential equations) is derived.

Model equations are specified *non–causally*, and are *real* equations in mathematical sense. The equations of the C–element given above, however, are specified in a *certain* causal form. This C–element could have been specified as:

```
c * der(p.e) = p.f
```

However, now no explicit state variable is specified, which causes the inheritance tree of bond–graph basic elements rather poor.

3 MAPPING BOND GRAPHS ONTO MODELICA

Mapping bond–graph models onto Modelica code is in principle a straightforward process, especially since Modelica has both non–causal equations and port–like interface facilities. Furthermore, the Modelica specifications of the basic bond-graph elements are available (Broenink, 1997b), or can easily be made.

As indicated in section 2, a bond–graph port maps onto a connector, a bond / signal maps onto a connect statement, and bond–graph models and submodels map onto Modelica models and submodels.

The coherence of the bond–graph elements can easily be specified using the *extends* construct of Modelica. Single inheritance can be specified straightforwardly (see also the creation of the C–elment in sec. 2.2). Single inheritance is sufficient for the basic bond–graph and block–diagram elements.

Using these inheritance features gives possibilities to build libraries of submodels efficiently and elegantly.

Comments

The Modelica *connect* statement is not used in its original form (i.e. with one *flow* variable), because this functionality (zero–to–sum equations) is *not* applicable for bond–graph like modeling. This is the reason why in Modelica the *flow* tag in a connector is optional.

Bond graphs are *port based* and Modelica models are *network based*. This difference in connecting strategy give rise to the following remarks:

- The bond–graph one–port elements (C, I, R, Se, Sf) also have one connector in their Modelica description, while the original Modelica description of these elements all are specialisations of the partial model *TwoPin*, which has two connectors, and consequently represents more equations than the bond–graph elements have. When using these models in a circuit, however, neither connection strategy has the minimum amount of equations in general.
- In the network connection strategy, to specify a connection between three or more ports, more than one connect statements is needed. This implies that for some connectors, more than one connection is specified, which can disguise sophisticated semantic checking.
  Furthermore, the compiler needs to collect all these connect, and combine them into *one* connect statement, containing all flow–tagged variables. This is a rather complex procedure.
- When specifying meshes, the connect statements will generate one Kirchhoff’s current law to many (cf. Mattson et al. 1997). The special *ground* model, *not* a specialisation from *TwoPin*, takes care of this.

For exchange of bond–graph submodels, Modelica lacks some expressiveness: Causality restrictions as attributes of the power ports, cannot be specified in Modelica. Thus, Modelica is *not* the optimal exchange language for bond–graph models. This can be worked around by letting the *bond*–graph causality algorithms distill the causal constraints from the equations before they can process the model, which can be a time–consuming process. Note that without the specification of causality restrictions, all information to generate the equations *is* available. The Modelica compiler generates equations *without* using causality or causal constraints.

Note that the inheritance tree of the basic bond graph elements only uses single inheritance, while multiple inheritance is needed for more realistic models: an model of a DC motor can be a member of the classes ‘DC motor’ and ‘tacho’, since the device represented by the model can be used in two different ways. Further elaboration on this subject can be found in Breunese et al. (1998), and chapter two of Breunese (1996).

We did not investigate the way back: translating Modelica models to bond–graph / block–diagram models. Considering the expressiveness and aim of Modelica, it is problably *not* possible to translate *all* Modelica constructs to bond–graph / block–diagram constructs.

4 IMPLEMENTATION OF THE BOND GRAPH TO MODELICA FILTER IN 20-SIM

We implemented a Modelica export filter in our bond graph / block diagram modeling and simulation software 20-SIM (Broenink, 1990; Broenink and Weustink, 1995; Broenink, 1997; Broenink 1998). It is an export filter from 20-SIM 2.3 to Modelica 1.0.

Principally, in the code generation modules of the Graph Editor (for bond graphs and block diagrams) and the Equation Editor of 20-SIM version 2.3, we added Modelica code generation functionality (see figure 1). This means that Modelica code of

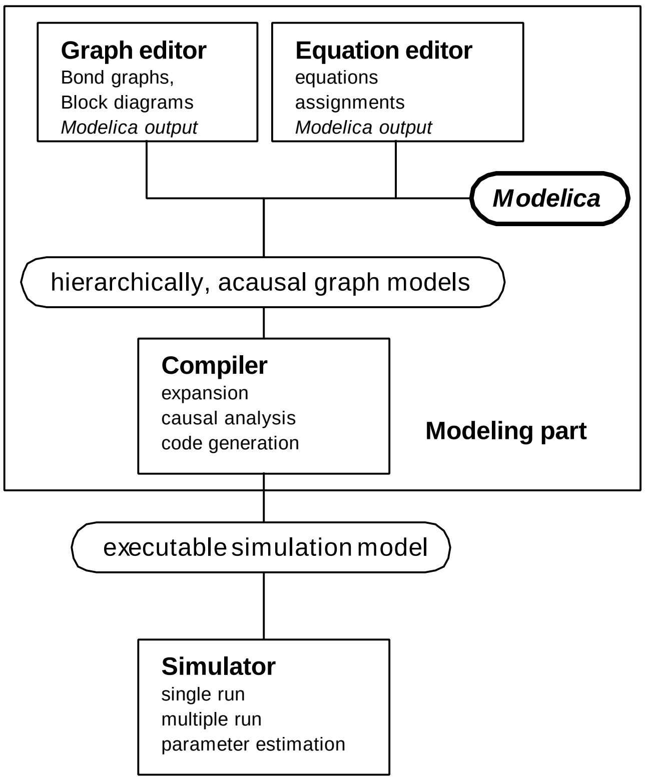

Figure 1 The structure of 20-SIM with the Modelica export filter indicated.

every distinct submodel will be generated. The Modelica translator still has to do its complete equation generation process.

Another place to generate Modelica code would be in the 20-SIM compiler, after any of the 3 phases (see figure 1). Since the hierarchy of the models is flattened out due to the expansion phase (inclusion of all submodel descriptions), the specific object–oriented facilities offered by Modelica are not used here. So, from a point of view of research on modeling languages, placing the Modelica export filter in the 20-SIM compiler is *not* a good idea.

The translation process from SIDOPS, the modeling language of 20-SIM, to Modelica is in general a translation on syntactic level: Each SIDOPS language construct translates to a Modelica language construct.

Subtle differences are:

- In Modelica, declarations are in one list, whereas in SIDOPS there are distinct declaration lists for parameters, variables, state variables etc.
- The power direction of bonds is specified implicitly: the first connector in a connect statement has power orientation flowing out (the half arrow points from the first connector to the second one).
- Since Modelica 1.0 does not support arrays of variables, for junctions a Modelica model for each amount of ports need to be specified: a two-port zero and one junction, a three-port zero and one junction, etc. Fortunately, all information to generate the correct code and use the appropriate junction, is available at the point in the 20-SIM compiler where the Modelica code generation is executed.
- The graphic coordinate systems are different, which causes the translation to be a rather extensive but straightforward procedure.

The current situation is that an export filter is made in 20-SIM version 2.3 to Modelica 1.0 for graph models (both bond graphs and block diagrams). Furthermore, a bond–graph library is build in Modelica. Fortunately, the export filter is rather simple. It is the straightforward code generation of one single (sub)model at the time. This property *is* important, since both languages are still under development. For equation models, the same design can be used. It is expected that mapping SIDOPS functions onto Modelica functions will not be a real problem. Specific functionality, like hybrid model description features, might cause a rather extensive translation procedure.

5 APPLICATION EXAMPLE

The application example we have chosen is a model of a computer controlled system. The top–level model is shown in Figure 2. The block–diagram submodels are standard 20-SIM submodels, and the bond–graph submodels (ovals) are specific for this application. The system consists of a DC motor driving a load via a belt (e.g. a circular saw). The actuator is the DC motor powered by a voltage source with a limiter in the input signal coming from the controller. The process consists of the belt system, two pullies and a load. The sensor consists of a tacho generator and an amplifier. For the tacho, we used the DC motor model. The models of the actuator, process and sensor are shown in figure 3, 4 and 5.

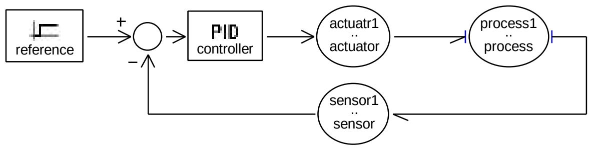

Figure 2 Application example, called *system1*

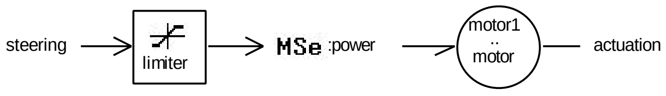

Figure 3 The actuator.

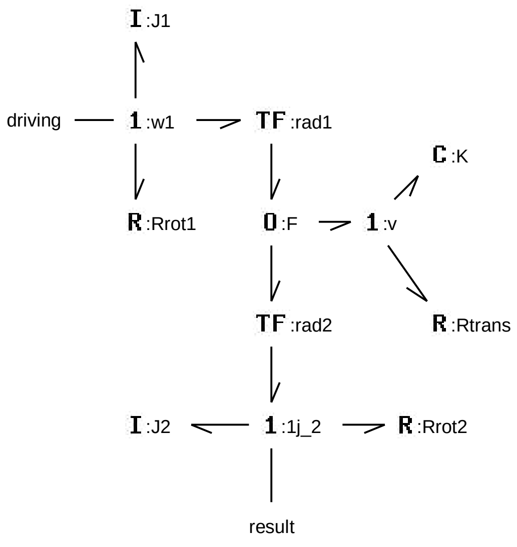

Figure 4 The process

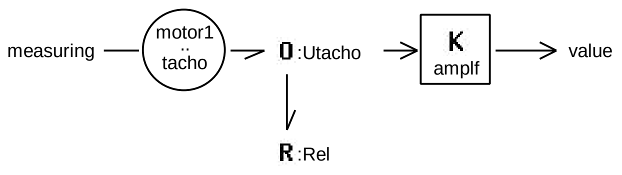

Figure 5 The sensor, using the motor model as tacho.

The 20-SIM model code (SIDOPS) is given in listing 1 and the Modelica code, generated by 20-SIM 2.3 (test version) is shown in listing 2.

```
bond graph system version 1
subclasses
        minus;1         minus_1
        stepgen;1       reference
        pid;1           controller
        actuatr;1       actuator
        process;1       process
        sensor;1        sensor
connections
        sensor`value            -:      minus_1`mininp
        controller`control      -:      actuator`steering
        minus_1`outp            -:      controller`error
        reference`outp          -:      minus_1`plusinp
        process`result          ->      sensor`measuring
        actuator`actuation      ->      process`driving
```

Listing 1: The SIDOPS (20-SIM) code of *system1*

```
:// Modelica 0.91 code generated by 20sim2.3
// Preliminary test version
// All 20sim library directories are imported:
import "D:\Bnk\20simprj\Scratch\Scratch.mo";
import "D:\Program Files\20sim\blckdiag\blckdiag.mo";
import "D:\ Program Files\20sim\\siggen\siggen.mo";
import "D:\ Program Files\20sim\\bondgrph\bondgrph.mo";
import "D:\ Program Files\20sim\\control\control.mo";

model System1
                minus1          minus_1;
                stepgen1        reference;
                pid1            controller;
                actuatr1        actuator;
                process1        process;
                sensor1         sensor;
equation
                connect(sensor.value, minus_1.mininp);
                connect(controller.control, actuator.steering);
                connect(minus_1.outp, controller.error);
                connect(reference.outp, minus_1.plusinp);
                connect(process.result, sensor.measuring);
                connect(actuator.actuation, process.driving);
end ;
```

Listing 2: The Modelica code of *system1*

While comparing both model descriptions, one can see that both codes seem rather similar. Note that *no* graphical information is printed in both cases. The connections in 20-SIM are translated to connect statements in the equation section of Modelica. In here, the direction (either power direction or signal direction) is fixed from the first to the second argument. Since this code is generated automatically, this is not a real restriction.

6 CONCLUSIONS

The export filter of 20-SIM to generate Modelica code has been built, although it is still in an experimental form. Modelica code generation of bond–graph models appeared to be a rather straightforward process.

The basic bond–graph elements and block–diagram elements have been specified in Modelica, using the essential object–orientation features *inheritance* and *encapsulation*. Equations have been specified in an *acausal* format. Thus, it can be said that the bond–graph library in Modelica is in the spirit of both bond–graph modeling and object–oriented physical systems modeling as advocated by Modelica.

Unfortunately, causality restrictions cannot be specified in Modelica. Due to this lack of expressiveness, Modelica is not the perfect exchange language for bond–graph models, but since Modelica is still under development, it might be better in the future.

This activity, building an export filter from bond graphs to some object oriented modeling language and building model libraries, indicates that bond–graph modeling can be seen as a form of *object–oriented physical systems modeling*. Since bond graphs came into existence before the term object orientation was used in the field of physical systems modeling, bond graphs can be seen as an object-oriented physical systems modeling paradigm *avant-la-lettre*.

REFERENCES

Barton, P.I. and Pantelides, C.C., “Modeling of Combined Discrete/Continuous Processes”, AIChE J., vol. 40, 966-979, (1994).

Breunese A.P.J., (1996), Automated support in mechatronic systems modeling, PhD thesis University of Twente, Enschede, Netherlands.

Breunese, A.P.J. and Broenink J.F., “Modeling mechatronic systems using the SIDOPS+ language”, Proceedings of ICBGM’97, 3rd International Conference on Bond Graph Modeling and Simulation”, Simulation Series, The Society for Computer Simulation International, vol. 29, (1), 301-306, (1997).

Breunese, A.P.J., Top, J.L., Broenink, J.F., Akkermans, J.M., (1998), Libraries of reusable submodels: theory and application, *Simulation*, Vol 77, no 1 (July), pp 7-22.

Broenink J.F., (1998), 20-SIM, software for hierarchical bond-graph / block-diagram models, Simulation In Practice and theory, to appear in the special issue on bond graphs.

Broenink, J.F. (1997b), Bond–graph modeling in Modelica, *Proceedings of 9th European Simulation Symposium*, W Hahn, A Lehmann (eds.), Passau Germany, Oct 19-22, pp 137-141.

Broenink, J.F. and P.B.T. Weustink, (1995), PC-version of the bond graph modeling and simulation tool CAMAS, *Proc Int conf, Bond graph modeling and simulation*, Las Vegas, SCS simulation series 27 no. 1: 203-208.

Broenink, J.F., (1990), *Computer aided modeling and simulation: a bond graph approach*, Ph.D. thesis, University of Twente, Enschede Netherlands.

Broenink, J.F., (1997), Modelling, simulation and analysis with 20-SIM, *Journal A* 38, no.3 (Sept): 22-25.
See also: http://www.rt.el.utwente.nl/20sim

Elmqvist, H., Brück D. and Otter M., "Dymola – User's Manual", Dynasim AB, Lund, Sweden, (1996).

IEEE, "Standard VHDL Analog and Mixed-Signal Extensions", IEEE , 1076.1, (1997).

Jeandel, A., Boudaud, F., Ravier, Ph. and Buhsing, A.", "U.L.M: Un Langage de Modélisation, a modelling language", Proceedings of the CESA'96 IMACS Multiconference, Lille, France, IMACS, (1996).

Kloas M., Friesen V. and Simons M., "Smile – A Simulation Environment for Energy Systems", Proceedings of the 5th International IMACS-Symposium on Systems Analysis and Simulation, Sydow, A. (ed.), System Analysis Modelling Simulation series, Gordon and Breach Publishers, vol. 18–19, 503–506 (1995).

Mattsson, S.E., Andersson M. and Åström K.J., "Object–Oriented Modelling and Simulation", CAD for Control Systems, E.D. Linkens (ed.) Marcel Dekker Inc, New York, ISDN 9203500, Chapter 2, 31-69, (1993).

Mattsson, S.E., H.E. Elmqvist and J.F.Broenink (1997), Modelica™ – An international effort to design the next generation modeling language, *Journal A* 38, no.3 (Sept): 16-19.

Modelica, (1997), Modelica™ - a unified object oriented language for physical systems modeling, see http://www.modelica.org

Piela, P.C., Epperly, T.G., Westerberg, K.M. and Westerberg, .W., "ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis: the Modeling Language", Computers and Chemical Engineering, vol. 15 (1), 53-72 (1991).

Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, (1991), *Object oriented modeling and design*, Prentice Hall.

Sahlin P., Bring A. and Sowell E.F., "The Neutral Model Format for Building Simulation, Version 3.02", Department of Building Sciences, The Royal Institute of Technology, Stockholm, Sweden", (1996).

Viklund, L. and Fritzson, P., "ObjectMath – An Object-Oriented Language and Environment for Symbolic and Numerical Processing in Scientific Computing", Scientific Programming, vol. 4, 229–250, (1995).

BIBLIOGRAPHY

Jan F. Broenink received the M.Sc. degree in Electrical Engineering and Biomedical Engineering (1984) and the Ph.D. in Electrical Engineering (1990) from the University of Twente. His Ph.D. research was in the design of computer facilities for modelling and simulation of physical systems using bond graphs. He is presently Assistant Professor at the Control Laboratory of the Department of Electrical Engineering of the University of Twente, where he the project leader *software tools development*. His research interests include development of computer tools for modelling, simulation and implementation of embedded control systems.
His homepage is at: http://www.rt.el.utwente.nl/bnk
His e-mail is: J.F.Broenink@el.utwente.nl