RECONFIGURABLE MODELS: A MODELING PARADIGM TO SUPPORT SIMULATION-BASED DESIGN

Antonio Diaz-Calderon, Christiaan J. J. Paredis[1], Pradeep K. Khosla
Department of Electrical and Computer Engineering
Institute for Complex Engineered Systems
Carnegie Mellon University
Pittsburgh, PA 15213, USA
E-mail: paredis@cmu.edu

KEYWORDS

reconfigurable model, object-oriented modeling, model space, component libraries, VHDL-AMS.

ABSTRACT

In this paper, we define a modeling paradigm that provides simulation support for the design of mechatronic systems. The paradigm supports associating more than one simulation model with each system component. These models are organized in a *reconfigurable model* that represents the entire model space of the component. The elements of this space are models that vary from abstract (conceptual) to concrete (fully determined), thereby supporting the evolutionary nature of the design process. They allow the designer to work with high-level concepts that can be specialized at later stages in the design process.

INTRODUCTION

Because of the intense competition in the current global economy, successful companies must react quickly to changing trends in the market place. For example, the need for a new product can be triggered by the introduction of new technologies, changes in customer demands, or fluctuations in the cost of basic materials and commodities. To capitalize on these imbalances in the market, a company must conceive, design, and manufacture new products quickly and inexpensively. Because the design process consumes a significant portion of the total development time, a shorter design cycle provides a distinct competitive advantage.

The design cycle can be shortened significantly through virtual prototyping. A virtual prototype enables the designers to test whether the design specifications are met by performing simulations rather than physical experiments; a physical prototype is only needed for final testing. Not only does virtual prototyping make design verification faster and less expensive, it provides the designer with immediate feedback on design decisions. This in turn promises a more comprehensive exploration of design alternatives and a better performing final design. To fully exploit the advantages of virtual prototyping, however, simulation models have to be accurate and easy to create.

Creating high-fidelity simulation models is a complex activity that can be quite time-consuming. To take full advantage of virtual prototyping, it is necessary to develop a modeling paradigm that supports model reuse, that is integrated with the design environment, and that provides a simple and intuitive interface which requires a minimum of analysis expertise. This paper introduces such a paradigm: *composable and reconfigurable modeling*, which is based on model composition from system components.

Reconfigurable models allow the designer to model systems in a hierarchical fashion. Initially, a system can be described by high-level components and the interactions between them. The models of the high-level components can then be refined iteratively without having to modify the system-level model description. As a result, the system model can be incrementally adapted as more detailed design features become known. This is in contrast to most currently available modeling environments in which a small change in the system description may require a large change in the model structure.

In this paper, we consider simulation models of mechatronic systems. Mechatronics can be defined as

> "A technological field that combines mechanics with electronics and information technology to form both *functional interaction* and *spatial integration* in components, modules, products and systems." (Breunese 1996)

1. Corresponding author.

To provide simulation support to the mechatronic design approach, we need modeling tools that capture system behavior across energy domains. In recent years, a number of modeling languages have emerged that capture mathematical models of mechatronic systems. These languages are based on object-oriented principles, and include Dymola (Dynasim AB 1999, Elmqvist *et al*. 1993), OMOLA (Anderson 1994), NMF (Sahlin 1996), and—more recently—Modelica (Elmqvist 1998) and VHDL-AMS (IEEE 1999).

Although these modeling languages are object oriented in nature, they do not permit the model *structure* to be easily modified. Instead, only mechanisms for parameter reconfiguration are provided. Given the evolutionary nature of the design process, it would be desirable to accommodate reconfiguration of the model structure also.

In de Vries' work with polymorphic models (the MAX system), he suggests an approach to achieve structure configuration (de Vries *et al.* 1995). His polymorphic models are similar to our concept of reconfigurable models; however, they present the following limitations:

- An instance of a model is considered an implementation. This forces a new type to be defined for each new set of parameter values for an implementation.
- Models are represented by bond graphs, which limits their applicability to lumped parameter systems.

To overcome such limitations, we propose a system representation based on two concepts: *interface* and *implementation*. In this model representation, systems are described from a systems engineering point of view where subsystems interact with their environment through energy exchange. The interface of a system describes the interaction through a set of ports. The implementation, on the other hand, describes a system's internal behavior. The interface and implementation together define a complete model of a system. A direct consequence of this new representation is that it is possible to assign different implementations to the same system interface, thereby achieving reconfigurability of models. We call system models that are based on this modeling paradigm *reconfigurable models.*

Because reconfigurable models are an extension of port-based models, we will first describe the port-based modeling paradigm.

PORT-BASED MULTI-DOMAIN MODELING OF MECHATRONIC SYSTEMS

To provide better support for simulation-based design of mechatronic systems, we have developed a modeling and design paradigm based on composition. A wide variety of products, ranging from consumer electronics to cars, contain mostly off-the-shelve components and components reused from previous design generations; for instance, in cars, the portion of completely new components is often less than twenty percent. As a result, the design of such systems consists primarily of the configuration or assembly of existing components.

The modeling of such systems can also be viewed as composition. We can obtain a system level simulation model, by combining the component models with the models that define the interactions between the components. Assuming the models for individual components already exist in a component library, and that the physics of the interactions between the components have been modeled in a library of interaction models, a system level simulation model can be generated through the composition of existing component and interaction models. To support modeling through composition, we have developed a port-based modeling paradigm.

A port-based modeling paradigm

In our modeling paradigm, subsystems interact with each other through ports (Diaz-Calderon *et al*. 1998, 1999). As noted above, ports represent localized points on the boundary of the system where energy exchange between the system and the environment takes place. At a port, energy flows in and out of the system. Consequently, there is a port for each interaction point, and each port will belong to an energy domain. For example, consider an electric transformer with four terminals. Each terminal represents a port through which electrical energy flows in and out of the transformer. In this example, the ports belong to the electrical energy domain; this captures the flow of power in terms of the voltages and currents on the two sides of the transformer.

The energy flow through a port is represented by means of two variables: an *across* variable and a *through* variable. Across variables represent values measured between a global reference point and the port. For instance, the across variable at one of the ports in the transformer represents the voltage between the physical terminal and ground. Through variables represent values measured through the element; e.g., the current in the transformer example.

Connections between ports represent the interactions between different components. A connection between two ports represents the energy exchange between two subsystems and imposes algebraic constraints on the port variables involved in the connection. In general, these

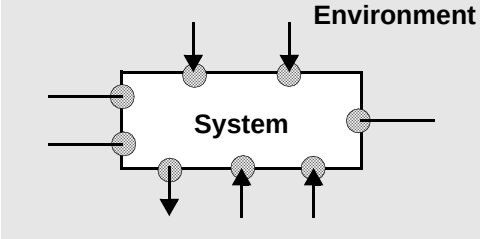


Figure 1: Model of an engineering system. Energy ports are represented by non-directed lines while signal ports are represented by arrows.

constraints take two forms: one form enforces the equality of the across variables, and the second enforces the sum of the through variables to be zero.

Physical interactions that represent energy exchange have no predefined direction. Therefore, we capture a physical interaction with undirected connections representing non-causal interactions. This approach to modeling reflects the physical interactions more accurately and relieves the modeler of specifying the input/output relations, as would be required in a modeling environment such as Simulink.

Besides ports and connections that model energy flow, we also consider signals and signal ports. No energy flows through the signal ports, and the interaction between signal ports is causal. That is, signal ports have a predefined input-output direction that constrains the signal flow between components. Signal and signal ports capture a system based on a block diagram description similar to Simulink.

The system's ports are collectively grouped into an interface, which defines the interaction points of the system with the environment. In this way, we can describe systems as self-contained entities, whose interactions with the environment can be described independently of the internal behavior of the system, as illustrated in Figure 1.

The port-based modeling paradigm can describe component interactions in any energy domain as long as the interaction is not distributed but lumped. Consider for example a flexible beam. A finite element model may describe the behavior of the beam. However, presuming that the interaction points of the beam are localized at the two ends, we can describe its interaction with the environment with two ports located at the two ends. Thus, our modeling paradigm is limited to interactions that are localized at a finite number of points on the boundary of the system.

As illustrated in Figure 2, the port-based modeling paradigm also supports a hierarchical model structure. The hierarchy

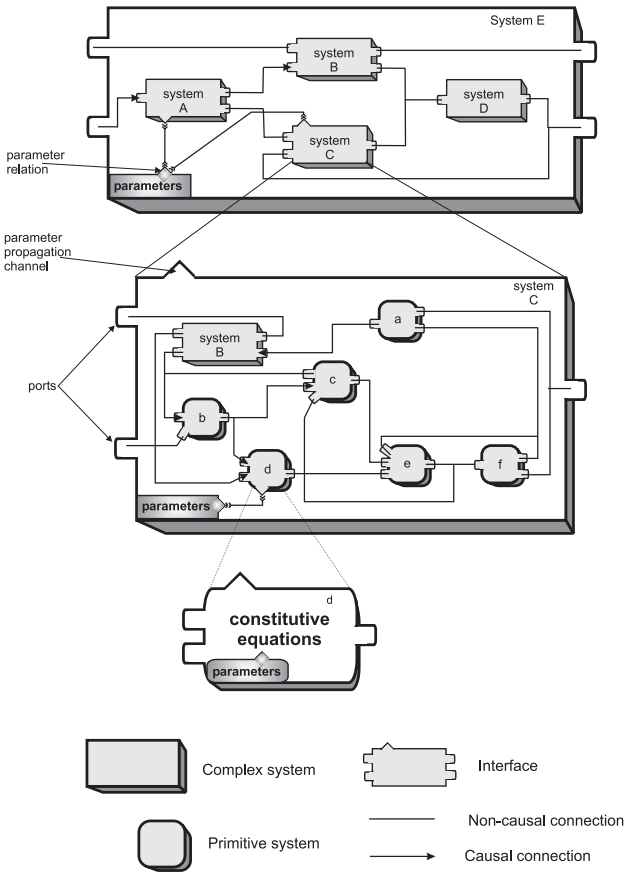


Figure 2: Hierarchical model structure

can have any number of levels; however, in order to transform it into an adequate simulation model, the hierarchy must be terminated with *primitive* systems, or systems that cannot be divided into smaller subsystems. *Compound* systems on the other hand are composed by connecting primitive models or compound models at a lower level in the hierarchy.

Equation-based modeling

The behavior of a primitive system can be specified using either of two formalisms, depending on whether the system is energy-based or not. Both formalisms represent the behavior of the system with a set of *constitutive equations.* The difference between the two types of systems lies in how the constitutive equations are specified and how the system's topology is described.

In the first formalism, the behavior of non-energy-based systems is described using a procedural approach. In this case, a signal quantity represents the quantity that is available to the environment through the interface of the system. Constitu-

tive equations of this kind are described as assignments (causal equations) that compute the value of a signal quantity based on the inputs to the subsystem.

In the second formalism, the behavior of energy-based systems can be represented using a graph-based model (Diaz-Calderon *et al*. 1999). Each edge in the graph has two associated quantities, called *branch quantities*: *across quantity* and *through quantity.* These quantities are analytic functions of time (i.e., they are piece wise continuous with a finite number of discontinuities). Across quantities, represent effort such as voltage, temperature, or pressure that are the result of a measurement taken across two energy ports of the system. Through quantities represent flow such as current, heat flow rate, or fluid flow rate that are the result of a measurement taken in series with the component. To illustrate this, consider an example of an electrical network. Here, the vertices of the graph represent equipotential nodes in the circuit, and edges represent branches of the circuit through which current flows. The measurement taken across two nodes defines the across branch quantity, while the measurement taken in series with the component defines the through quantity.

The constitutive equations are expressed by relating the across and through quantities of one or several branches—for example, a resistor has a single branch, and its constitutive equation (Ohm's law) relates the voltage across (the across quantity) and the current through (the through quantity) the resistor.

Constitutive equations define the behavior of a subsystem without specifying a particular causal direction. The causal direction emerges only when the equations are combined with constitutive equations of other subsystems, and the quantities that are external to the subsystem are specified. A quantity is external to the subsystem if its value is computed using a constitutive equation that is not part of the constitutive equations of the subsystem. For example, when the constitutive equation of a resistor is combined into a larger system of equations, either the voltage or the current will be defined externally; Ohm's law is causally oriented to reflect this: $v := iR$ or $i := v/R$. In the first case, the voltage v is computed from the current i, which is computed using an external equation. Similarly, in the second case, the current i is computed from the voltage v.

The equations appearing in the model can be of several kinds: *ordinary differential equations*, *algebraic equations*, or *differential-algebraic equations*. Ordinary differential equations (ODE) can represent the variation of quantities as a function of a single variable, such as time. Therefore, ODEs are used to represent lumped parameter models. Algebraic equations do not contain partial or total derivatives. When a model includes both ODE and algebraic equations, the resultant system of equations is called a differential-algebraic system of equations (DAE). In this type of system, algebraic equations represent constraints among the state variables defined by the set of ODEs.

Constitutive equations of both types (i.e., non-causal and causal) may include a combination of branch quantities and signal quantities. We call these types of models *hybrid models* since they describe the interaction of energy-based systems with non-energy-based systems. An example of this type of behavioral description would be an electromechanical system controlled by a digital controller.

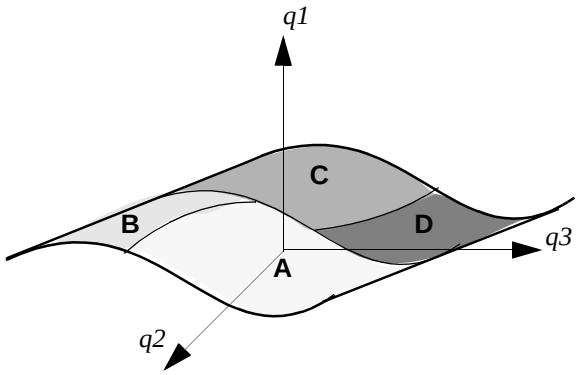
If the system is compound, its behavior is described according to the structural arrangement of subsystems, which in turn may be compound or primitive. This unambiguously defines the topological constraints among components. Consequently, a compound system can be reduced through a sequence of algebraic transformations into a primitive model that exhibits the same behavior.

Meta knowledge

Whether using a compound or primitive model to describe a system's behavior, constitutive equations do not provide sufficient information to reason about the properties of the system since they are based on implicit assumptions and approximations. In other words, the context in which a model of a system can be applied is not explicit. If such knowledge were explicit, one could not only reason about the applicability of the model to a given problem, but also decide when models having similar properties can be interchanged.

One kind of meta knowledge that we consider is the *operating region* of the model. The operating region defines the space of admissible values for the quantities of the model, which provide meaningful results. Outside this space, the model may provide erroneous results that invalidate its application. To ensure that a model is used within its operating region, we explicitly express its bounds through the *operating conditions* of the model. Operating conditions are conditional expressions on the quantities of the model and explicitly define the sub-domain for each quantity for which the equations stated in the model are valid.

When it is necessary to define multiple operating regions, for example to capture a large portion of the system's operating region, operating conditions are disjointed. Each element of the disjunction represents a valid operating region that has an




```
model process
quantities
  q1, q2, q3
equations
  if OC1(q1, q2, q3)
    RA(q1, q2, q3)=0
  elif OC2(q1, q2, q3)
    RB(q1, q2, q3) = 0
  elif OC3(q1, q2, q3)
    RC(q1, q2, q3) = 0
  elif OC4(q1, q2, q3)
    RD(q1, q2, q3) = 0
end process
```

Figure 3: Segmentation of the domain based on different operating regions

associated set of constitutive equations. In other words, operating conditions allow us to segment the domain of the quantities involved in the constitutive equations. Consequently, we can obtain a model that is applicable within a larger operating region of the system. For example, consider the operating region of the system shown in Figure 3. The domains of the three quantities define the solution space of the system. The operating conditions given as a function of the three quantities (represented by the functions OC_i) segment the space into four regions *(A, B, C,* and *D),* each of which has an associated set of equations (represented by the function R_j) that describe valid behavior under these conditions.

Current support for port-based modeling

Using object-oriented modeling principles, it is possible to describe a port-based model that is composable and hierarchical (Elmqvist 1998, Sahlin 1996, Anderson 1994, IEEE 1999). However, in an object-oriented modeling paradigm, there is not necessarily a clear separation between the interface of the model and the implementation of its behavior. Often both concepts are merged together into a single modeling entity. In this modeling approach, only parametric reconfiguration is allowed. However, to evaluate the design at different levels of detail, changes in structure should also be supported. This is achieved with reconfigurable models.

RECONFIGURABLE COMPONENT MODELS

In a reconfigurable model, the interface of the model and the implementation of its behavior are considered to be two separate concepts. By considering these two concepts independently, it is possible to associate different implementations with a single interface, achieving a structural modification of models, and consequently, creating a *reconfigurable model* (Figure 4). In addition to the traditional changes in parameter values, a reconfigurable model provides a mechanism to describe changes in the model structure by using the principles of *composition* and *instantiation*.

The composition principle denotes the mechanism by which the formal behavior of the component is described in terms of interfaces of subcomponents and their interactions. Since composition unambiguously represents topological constraints among components, a composed model given by the pair $\langle \Phi | \phi_c \rangle$ can be reduced to a primitive model $\langle \Phi | \phi_p \rangle$ which exhibits equivalent behavior, where $\langle \Phi | \phi \rangle$ represents the binding of implementation ϕ to interface Φ.

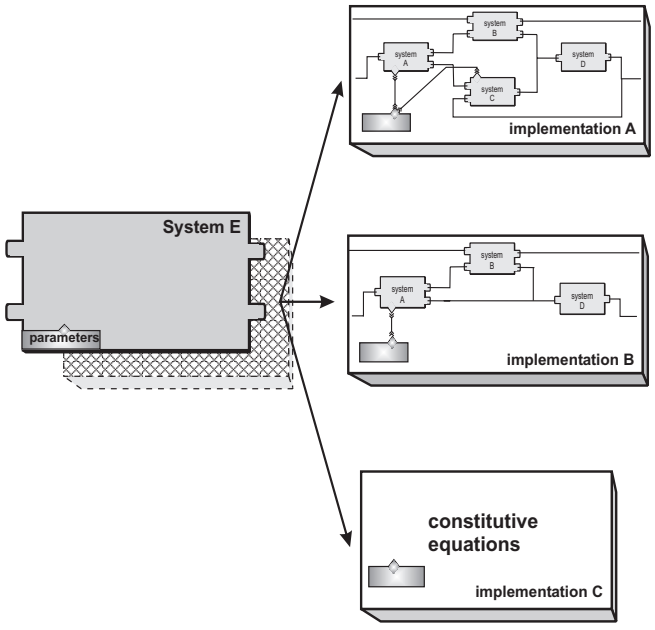


Figure 4: A reconfigurable system model.

As a result of the principle of instantiation, compound implementations are abstract. They are a composition of abstract interfaces, that still need to be bound to implementations to specify the behavior completely. Consequently, instantiating a compound component (i.e., a component with a compound implementation) requires the recursive instantiation of each interface in the component.

Reconfigurable models are hierarchical in nature. Based on the composition principle, we define self-contained implementations of a system in terms of the composition of subsystem interfaces; i.e., a compound implementation. However, a hierarchical system defined by reconfigurable models is not fixed. Rather, it changes as implementations are bound to the interfaces (model instantiation) that describe the compound implementation. Specifically, binding different implementations to an interface results in a different structural arrangement and thus a different hierarchical structure.

The second principle—the principle of instantiation—describes the mechanism by which the interface of a model is bound to its implementation. An implementation that meets the requirements of an interface, can generally be bound to it. However, the semantics of the resulting model must be consistent with the context in which the model is used. For example, consider the case of a resistor and a capacitor whose interfaces both include the same set of interaction points (two electrical terminals). In such a case, an implementation that satisfies the interface for the resistor will also satisfy that of the capacitor. However, the semantics of the resulting model will differ; hence they cannot be interchanged. In summary, we will allow bindings that produce models having the same semantic meaning.

There are two kinds of implementations that can be bound to an interface and maintain a consistent semantic interpretation of the model. These are implementations with different representations of equivalent behavior, and implementations with different behavior. Accordingly, if τ and κ are two implementations that satisfy interface Φ, then,

$$SEMANTICS(\langle\Phi|\tau\rangle) = SEMANTICS(\langle\Phi|\kappa\rangle) \qquad (1)$$

The instantiation principle also provides the basis for defining a family of systems (i.e., the model space of a component). An interface that can be bound to different implementations by the instantiation principle defines a family of systems. All members of the family will show the same interaction characteristics but with different formal behavior. This method of describing membership of an element in a set is referred to as a *type* in the theory of computational objects and can be phrased as follows (Abadi and Cardelli 1996):

> "The type of an object provides the semantic information that completely characterizes the object but not its behavior."

An interface defines a type, each member of which is a subsystem having a unique formal behavior. Based on this observation, it is possible to organize system models into a type hierarchy. This type hierarchy is derived from a type system that provides the notions of subtype and supertype (Abadi and Cardelli 1996).

Let the symbol <: represent a reflexive and transitive subtype relation between interfaces I and I', then:

Definition *Subtyping.* I' <: I if I' has the same ports and parameters as I and possibly more, and the following conditions pertain:

1. The types of the ports are subtypes of types of corresponding ports in I.
2. The types of the parameters are subtypes of the corresponding parameters in I.
3. The semantics of the parameters are equivalent to the semantics of the corresponding parameters in I.

Based on the definition of subtype (and its complement, supertype), two operations can be defined on the type hierarchy, namely, *specialization* and *generalization*. Specialization involves finding an interface for the same family of components that includes more detail, while generalization involves finding an interface that is less specific. These operations are carried out by traversing the hierarchy in either a downward direction (specialization) or an upward direction (generalization).

Definition *Specialization*. Let x and y be two interfaces. Interface y is a specialization of interface x, denoted $y \xrightarrow{S} x$, if and only if the type of y is a subtype of the type of x, and for any system S that contains x, y can be substituted for x while maintaining the same semantics. This can be stated formally as:

$$\begin{aligned} y \xrightarrow{S} x \Leftrightarrow\ & TypeOf(y) <: TypeOf(x) \wedge \\ & (\forall S|_x, S(x \ll y) \rightarrow \\ & SEMANTICS(S|_y) = SEMANTICS(S|_x)) \end{aligned} \qquad (2)$$

where the operator $S\langle x \ll y\rangle$ should be interpreted as "x is substituted by y in S" and $S|_x$ should be read "the system S

evaluated with interface *x*". Similarly, it is possible to derive the property of generalization based on the supertype relation.

Parameter handling

In addition to the ports, it is also important to include the parameters in the interface. Model parameters describe fundamental characteristics of the system. For example, inertia and torque constants specify the invariant properties of a system that represents an electric motor; they are constant quantities that do not change value throughout the entire simulation.

In defining parameters of lower level subsystems, two kinds of parameters can be identified: *formal* and *actual* parameters. A formal parameter is defined locally in the interface of the subsystem, and it is used by a bound implementation. An actual parameter contains the value of an argument that is related to a formal parameter in a call to the model, and it is defined by the environment. The value of the argument is the value of an expression defined in terms of the formal parameters in the current scope. Parameter composition ensures that the parameters of the system are propagated to all the subsystems that were incorporated into the compound implementation.

The principles of instantiation and composition together with parameter propagation provide the infrastructure required to define reconfigurable models.

SUMMARY

In this paper, we have described a modeling paradigm based on reconfigurable component models that supports the design of mechatronic systems. In this paradigm, mathematical models consist of two elements: interface and implementation. The interface defines the mechanism by which the model interacts with its environment, while the implementation describes the behavior of the component. Model reconfiguration is achieved when the model of a component is defined by binding an implementation to an interface.

ACKNOWLEDGMENTS

This research was funded in part by DARPA under contract ONR # N00014-96-1-0854, by the National Institute of Standards and Technology, by the NSF under grant # CISE/IIS/KDI 9873005, by the Pennsylvania Infrastructure Technology Alliance, and by the Institute for Complex Engineered Systems at Carnegie Mellon University.

REFERENCES

M. Abadi and L. Cardelli, *A theory of objects*. Monographs in Computer Science, Springer-Verlag, New York, 1996.

M. Anderson. "Object-oriented modeling and simulation of hybrid systems," Ph.D. Thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1994.

A. P. J. Breunese. "Automated support in mechatronic systems modeling," Ph.D. Thesis, Department of Electrical Engineering, University of Twente, Enschede, The Netherlands, 1996.

A. Diaz-Calderon, C. J. J. Paredis, and P. K. Khosla, "A modular composable software architecture for the simulation of mechatronic systems," presented at ASME Design Engineering Technical Conference, 18th Computers in Engineering Conference, Atlanta, GA, September 1998.

A. Diaz-Calderon, C. J. J. Paredis, and P. K. Khosla, "A composable simulation environment for mechatronic systems," presented at SCS 1999 European Simulation Symposium, Erlangen, Germany, October 1999.

Dynasim AB, "Dymola." Lund, Sweden, 1999.

H. Elmqvist, F. E. Cellier, and M. Otter, "Object-oriented modeling of hybrid systems," presented at European Simulation Symposium, Delft, The Netherlands, October 1993.

H. Elmqvist, S. E. Mattsson, and M. Otter, "Modelica: The new object-oriented modeling language," presented at The 12th European Simulation Multiconference, Manchester, UK, June 16-19 1998.

IEEE, 1076.1 Working Group, "Analog and mixed-signal extensions for VHDL," IEEE, 1999.

P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg, "ASCEND: An object oriented computer environment for modeling and analysis. 1- The modeling language," *Comput. Chem Engng*, vol. 15, no. 1, pp. 53-72, 1991.

P. Sahlin. "Modeling and simulation methods for modular continuous systems in buildings," Ph.D. Thesis, Department of Building Sciences, Division of Building Services, Royal Institute of Technology, Stockholm, Sweden, 1996.

T. J. A. de Vries and A. P. J. Breunese, "Structuring product models to facilitate design manipulations," presented at International Conference on Engineering Design, Praha, August 22-24 1995.