Object-Oriented Libraries of Physical Components in Simulation and Design

Li Han Christiaan J. J. Paredis Pradeep K. Khosla
Institute for Complex Engineered Systems and
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
{lihan,cjp,pkk}@cs.cmu.edu

Keywords: Composable modeling, design evolution, object-oriented libraries, simulation-based design, virtual prototyping.

Abstract

Our competitive global business environment promotes faster, better and cheaper product design. With the rapid advancing computational technology, virtual prototyping and simulation-based design have great potential to reduce design cost and improve design quality.

To support simulation-based design of mechatronic systems, our group has developed a simulation and design environment in which design and modeling are tightly integrated. This integration is based on *component objects* that combine descriptions of both form and behavior of system components. By composing component objects into systems, the design team simultaneously specifies design alternatives and creates their models.

To facilitate component reuse and organization, and to accommodate modeling of systems evolving throughout the design process, we have developed a hierarchical component library structure based on a function taxonomy. When moving from the top to the bottom of the hierarchy, the component objects become more specific. A single component may appear in multiple locations in the taxonomy, depending on the viewpoint for its classification.

We have also developed a mechanism that allows a component object to gain access to the high-level behavior models of its ancestors and to be replaced by its descendants with more detailed behavior models. This allows the virtual prototype to evolve throughout the whole design process and to achieve the accuracy and efficiency required for the simulation experiments at each design stage.

1 INTRODUCTION

Our competitive global business environment promotes faster, better and cheaper product design. In general, design is an iterative procedure with design alternatives evolving from abstract to specific. In order to verify proper functioning and provide design feedback, design alternatives need to be tested with physical mock-ups or virtual prototypes. With the rapid advancing computational technology, it becomes relatively fast and cheap to create, modify and test computer models. This renders substantial potential to virtual prototyping over physical prototyping in terms of improved design quality and reduced temporal and monetary design cost.

The design process is iterative and hierarchical in nature. To solve complex design problems, a design team typically considers the problem at different levels of abstraction, ranging from very high-level system decompositions to very low-level detailed specification of components. Accordingly, in the early stages of the design process, simulation models can capture the high-level, intended behavior of sub-systems, allowing one to use simulation to make important conceptual trade-offs. As more details of the actual embodiment are included in design artifacts, more detailed models of the physical components will gradually replace these high-level models. This indicates a need for a modeling paradigm to accommodate the modeling of systems evolving throughout the design process.

To support virtual-prototyping and simulation-based design, component objects and their models need to be organized in a fashion that facilitates their retrieval and subsequent reuse in evolutionary design and modeling process. This paper will present our component and component library schemata designed to address these issues. Section 2 discusses related work. Sections 3 and 4 introduce respectively our component object and component object library schemata, and discuss their usage in simulation and design. Section 5 concludes the paper with a summary and a brief discussion on future research.

2 RELATED WORK

A physical component is typically characterized by its form, function, and behavior[1, 2]. The form is a description of the physical embodiment of an artifact, while function is the purpose of the artifact—the behavior that is intended for the artifact. As is illustrated in Figure 1, the actual behavior does not depend on the function, but only on the form. While design or synthesis needs to determine a form to satisfy a given function, design verification needs to derive the behavior from the form and verify whether this behavior matches the function. In the context of virtual prototyping, the behavior is described by mathematical models and design verification is achieved by performing simulation experiments with these models.

In general, one can think of design as a process that consists of *decomposition* and *composition*. High-level functions are hierarchically decomposed into functions for subsystems; these sub-functions are then mapped to physical components that are in turn recomposed into a complete system. This is the so-called *configuration design* process, which leads to designs specified in terms of components and their interconnections with each other. Such a design representation parallels the hierarchical modeling paradigm of a system: models of components are connected to each other via interaction models (describing the dynamics of the component interactions) to reach a model of the system. Both representations are based on hierarchical composition: composition of form in design and behavioral models in simulation [3, 4].

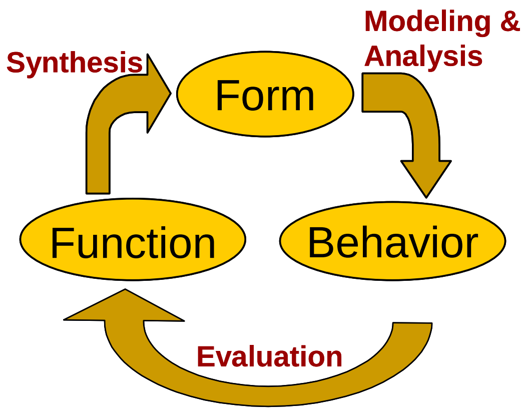

Figure 1: The relation between form, function, and behavior in the context of virtual prototyping

By taking advantage of the parallelism between composition in configuration design and composition in system modeling, a designer may simultaneously specify designs and create their models. This is already common practice in several single-domain simulations such as electrical systems[5-7] and mechanical systems[8, 9]. The current trend is to extend this approach to multi-domain simulation [10, 11].

One mechanism to facilitate model composition is equation-based modeling[7, 12], which uses a set of equations to establish relations between the states, their derivatives, and time. Declarative or equation-based modeling does not impose a fixed causality on the model, as in the case of procedural modeling that uses assignments to express a dependent variable as a function of independent variables (fixed causality). The simulation engine for declarative modeling is responsible for converting the equations into software procedures that can be evaluated by the computer. The advantage of declarative languages is that users do not have to define the mathematical causality of the equations, so that the same model can be used for any causality imposed by other system components

The software design methodology of object-oriented programming has been applied to systems modeling as well, with the benefits of simplified model creation and maintenance [13-15] [16-18]. An important principle of object-oriented programming is that of *information hiding or encapsulation*: only the public interface of an object affects its interconnections with other objects. The same principle can be applied to modeling by making a clear distinction between the physical interactions of an object with its environment (*interface*) and its internal behavior (*implementation*) [19, 20]. The advantage of encapsulation is that a system can be modeled by composing (connecting) the interfaces of its sub-systems, independently of the future implementations of these subsystems [4, 20, 21].

A second important principle of object-oriented programming is *inheritance:* objects that are derived from a parent class inherit its interface and data members. Similarly, in modeling, a model that derives from a parent model inherits the parent's interface and equations. The child model can be extended by including additional physical interactions (ports and parameters) in the interface or additional equations in the implementation [13, 14, 16]. Object oriented model design results in a hierarchical organization of component objects and simplifies the tasks of component and model reuse, maintenance, and organization.

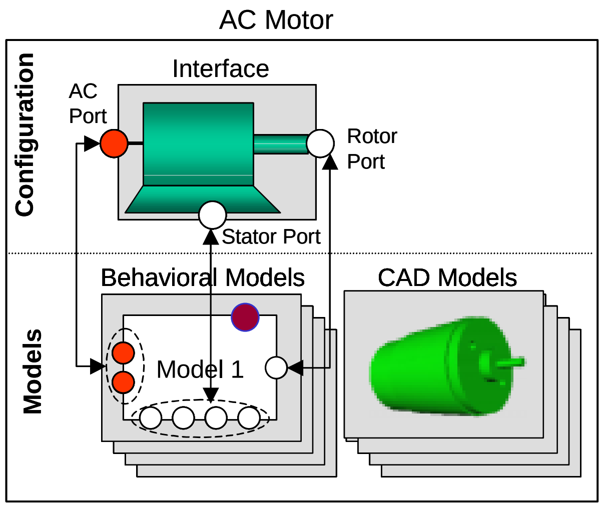

Figure 2: Primitive component objects consist of a configuration interface, CAD and behavior model(s).

The current design databases used in industry mainly contain geometry CAD information. It has been recognized that more information on design, such as functional descriptions, design rationales, manufacturing processes, mathematical models and animations, is needed to facilitate design knowledge exchange and reuse. This issue has been addressed in the study of several research groups[22-24]. A very comprehensive effort to organize and store design knowledge is part of the ongoing NIST design repository project [25, 26].

3 COMPONENTS IN COMPOSABLE SIMULATION AND DESIGN

To facilitate virtual prototyping and simulation-based design of mechatronic systems, our group has been working on a *composable simulation and design* paradigm, which is based on model composition from sub-components[27-31]. Our research so far has focused on developing mechanisms for component and model reuse, evolution and organization. In the following two sections, we will introduce the schemata for component objects and component object libraries. We will discuss their usages in simulation and design, and point out related technical issues.

3.1 Data Schema for Components

For the efficiency of system design and modeling, a component object needs to include information that reflects the characteristics of the modeled physical component. In our current implementation, a component object, as shown in Figure 2, consists of a configuration interface, configuration(s), CAD model(s), behavioral model(s), and relationships between them.

The *configuration interface* of a component consists of *ports and instantiation parameters*. The instantiation parameters completely specify the information needed to model the component, while the configuration ports define the intended interactions between a component and its environment. For instance, the configuration interface of the AC motor in Figure 2 has ports for the fastener holes in the stator, the shaft of the rotor, and the electrical connector. It is through its ports that a component is connected to and interacts with other components. In our system, a user will use a configuration interface to represent a component in a system specification and to gain access to the associated behavior models.

The *behavioral models* in the component objects are also characterized by port-based interfaces. However, here, the ports model the exchange of energy, mass, or signals between a component and its environment. Often there is a one-to-one mapping between the ports of the configuration interface and the ports in the behavioral interface but not always. For instance, the shaft of the AC motor corresponds to a mechanical energy port. But the AC plug is modeled as two electrical ports, one for each pin, and there is no configuration port corresponding to the thermal port in the behavior model, since thermal loss is not intended for an AC motor. Port-based behavioral models are further decomposed into *interfaces* (behavior ports and model parameters) and *implementations* (in the form of mathematical equations or composition of the behavior models). The port-based models are re-configurable, so that the same component can be simulated at different levels of details without having to modify the system-level model description. More information on our behavior models can be found in [20, 29, 30, 32].

The *CAD models* in component objects serve a dual role. On one hand, a default CAD model can be used as a *specification* of the form of a component: it provides nominal dimensions, tolerances, and material specifications—enough information for a third party manufacturer to manufacture the object. On the other hand, a CAD model is a mathematical representation of the geometry of an object. In this role, it can be used for visualization purposes, or as part of behavioral models[31], e.g. in collision check. Depending on the required accuracy of the analysis, multiple CAD models may be used to describe the component at different levels of detail.

The *configuration* of a component specifies subcomponents and their interconnections. In other words,

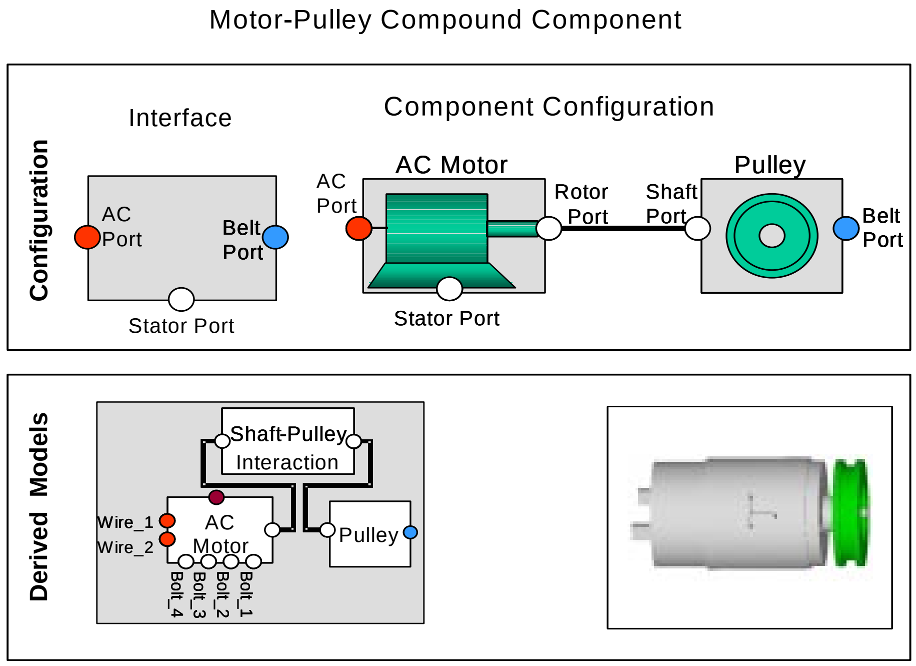

Figure 3: Compound components have configuration(s) associated with their configuration interface. Their CAD and behavior models can be derived from those of their sub-components.

it reflects the *physical decomposition* of the component and can be viewed as an *implementation* of a configuration interface. A configuration interface may be associated with multiple configurations to reflect different physical decompositions.

A *primitive component* is a component that does not include configuration information. The AC motor in Figure 2 is treated as a primitive component without elaborating its physical subcomponents, while the motor-pulley component shown in Figure 3 is a compound component. Notice that a compound component can derive its models from the composition of the models of its subcomponents, as shown in Figure 3. Therefore, it is allowed for a compound component not to include CAD and behavior models in its definition, which is not the case for a primitive component.

3.2 Use Case Scenario: Use Component Objects in Simulation

Assume that a user has created a design alternative through a composition of component objects. Then the following steps will be performed to create and simulate a behavior model of the design.

1. Connect the configuration ports of the component objects as dictated by the design alternative.
2. For each component object, choose a behavior model.
3. Connect the behavior models and choose interaction behavior models.
4. Simulate the overall system behavior model to see if the design alternative satisfies the design requirements.

In a first step, the compatibility of the connecting configuration ports is checked. Mismatches such as connecting a DC electrical port to a mechanical port will be caught and reported to the user. We use a configuration port hierarchy and port compatibility map to determine if two configuration ports can be connected [30].

Model selection, as performed in a second step, is a difficult research topic on its own[33-35]. The main concern is to choose a model that is accurate and efficient enough to meet the simulation requirements. Our reconfigurable model structure and the inheritance mechanisms introduced in the next section will provide a variety of behavior models with different levels of details from which a user can choose. We are studying various ways to assist users in the selection of models and even automate the model selection process.

The correspondence between configuration ports and behavior ports provides default behavior port connections, based on the configuration port connections specified in the first step. However, the user may still need to specify

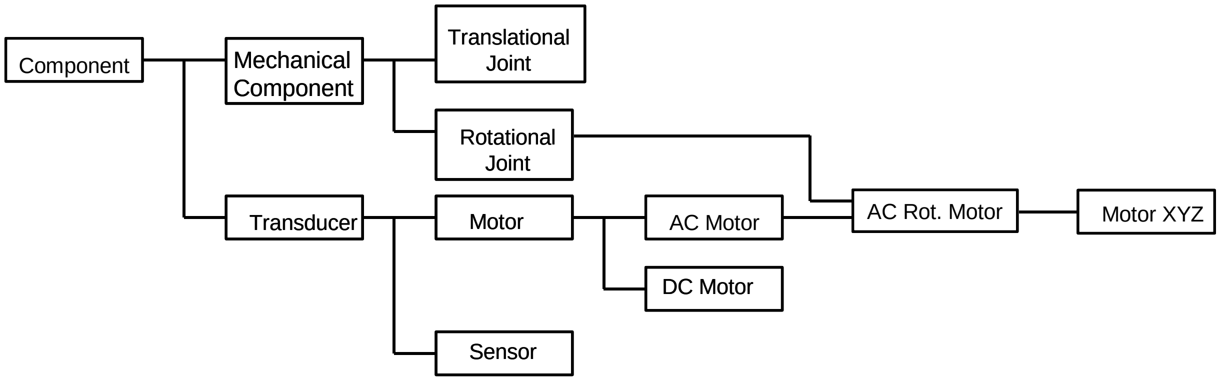

Figure 4: An example schema for a component library

connections for behavior ports that do not have a one-to-one mapping to configuration ports. Furthermore, the user needs to choose interaction models appropriate for simulation requirements[30].

4 COMPONENT LIBRARIES IN SIMULATION AND DESIGN

4.1 Data Schema for Component Libraries

For the success of composable modeling and simulation, it is imperative to develop a component library scheme, that stores components and their models in an organized fashion, that facilitates component indexing, search and reuse, and that accommodates the modeling of systems evolving throughout the design process.

To address these issues and to take advantage of the object-oriented modeling paradigm, we propose a hierarchical component library structure based on a function taxonomy, as illustrated in Figure 4. When moving from the top to the bottom of the hierarchy, the component objects become more specific. At the top, the objects are abstract and represent families of components sharing some functionality, such as the family of all motors which has the function of converting electrical energy to mechanical energy; at the bottom, the leaf nodes of the hierarchy represent completely specified physical components, such as AC motor XYZ manufactured by company ABC.

A single component may appear in multiple locations in the taxonomy, depending on the viewpoint for its classification. For example, an AC rotational motor is an energy conversion component, but can also be considered as a structural element that implements a rotary joint.

As a refinement of its parent component, a child will have a configuration interface that is a *subtype* of the configuration interface of its parent. In other words, a child will have more configuration ports and instantiation parameters, or have configuration ports and instantiation parameters that are subtypes of the parent ports and parameters.

Furthermore, we notice that models of a component object should be applicable to all of its child objects, due to the abstraction-refinement relationship between a parent and its children. To facilitate model reuse and model consistency between a child object and its parents, we introduce an object-oriented mechanism that would allow a child to inherit its parents' models, and recursively, to inherit all of its ancestors' models. With such an inheritance mechanism available, the modeling of a new component only needs to be focused on its specific properties that are not reflected by its ancestors.

In particular, we will establish the correspondence between the configuration interfaces of each child and parent pair, so that a child component may have access to the configuration interface of it parents, and thus, gain access to the behavior models of its parents. Figure 5 shows the correspondence of the configuration interfaces of a motor and an AC motor. In particular, the aggregated port of the stator port and the rotor port, and the AC port of the AC motor correspond to the mechanical port and the electrical port of the motor.

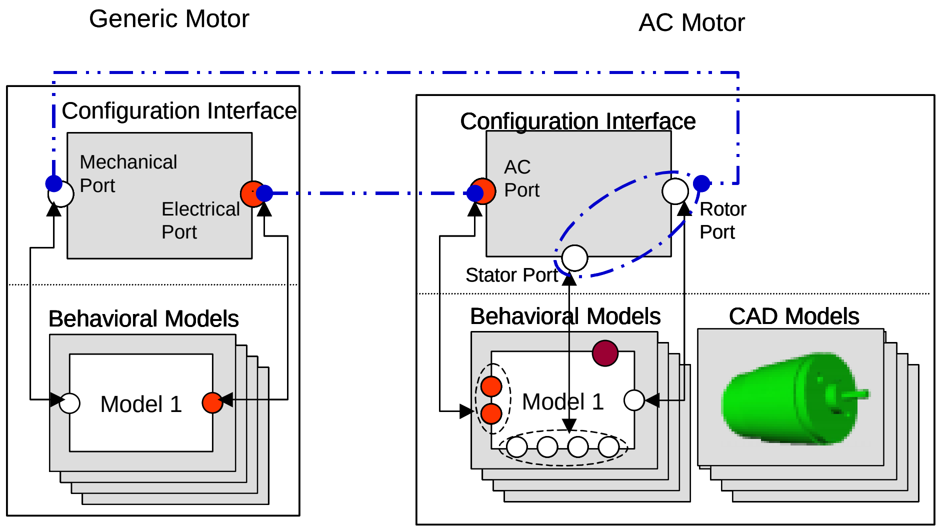

Figure 5: Correspondence of the configuration interfaces of a parent-child pair of component objects

In addition to the access to the models inherited from its ancestors or created for its own, a component object in a system virtual prototype can be replaced by its descendants with more detailed models. This allows the virtual prototype to evolve throughout the whole design process and to achieve the accuracy and efficiency required for the simulation experiments at each design stage.

4.2 Use Case Scenarios

Support Design and Simulation Evolution with Hierarchical Component Objects

Assume that component *P* is a parent of component *C*. Then the configuration interface of component *C* is a subtype of that of component *P*.

1. For design evolution: assume that component P is used in a design alternative specification and simulation, and that a designer wants to move to a more detailed design by replacing it with component *C*. Then the conventional subtyping mechanism would allow the replacement of the configuration interface of component *P* with that of component *C*. As a result, the behavior models of component *C* may be used in generation of a more detailed system model.
2. For the usage of a high-level model in simulation: assume that component *C* is used in a design alternative specification and simulation, and that a designer wants to use its high level model in the system simulation. (This requirement may happen in many situations, for example when component *C* is not the major concern of the simulation study and the designer wants to reduce the overall simulation time.) Then with the established correspondence between the configuration interfaces of components *P* and *C*, component *C* may get access to the configuration interface of *P* and use the high-level behavior models of *P* in the overall system simulation.

Add a New Component Object to a Library

Consider a library (possibly empty) and a component object that needs to be added to the library. To take advantage of the inheritance mechanism and maintain the functional hierarchical structure, the component can be modeled and inserted into the library in the following way.

1. Identify all of its parents and the appropriate position for it in the function hierarchy.
2. Establish the relationship between its configuration interface and those of its parents.
3. Model its specific behaviors that have not been reflected by its parents and ancestors.
4. Check if the newly added component can serve as a parent for other components. If so, establish the relationship between its configuration interface and those of its children, and possibly, simplify the behavior models of the children by removing the models included in the new component.

The first step will put the new component as a leaf node in the hierarchy. However, it is possible that a more specific component that can be its child has already been stored in the library. Step 4 is used to capture this kind of relationship.

5 SUMMARY

To support simulation-based design of mechatronic systems, our group has developed a simulation and design environment in which design and modeling are tightly integrated. This integration is based on *component objects* that combine descriptions of both form and behavior of system components. By composing component objects into systems, the design team simultaneously designs and models new artifacts.

To facilitate component reuse and organization, and to accommodate modeling of systems evolving throughout the design process, we have developed a hierarchical component library structure based on a function taxonomy. When moving from the top to the bottom of the hierarchy, the component objects become more specific. A single component may appear in multiple locations in the taxonomy, depending on the viewpoint for its classification.

We have also developed a mechanism that allows a component object to gain access to the high-level behavior models of its parents and to be replaced by its children with more detailed behavior models. This allows the virtual prototype to evolve throughout the whole design process and to achieve the accuracy and efficiency required for the simulation experiments at each design stage.

The research presented in this article is only an initial step towards an integrated framework for simulation-based design. Our current implementation is limited to component models with lumped interactions. To allow very detailed analyses and to more accurately model some physical phenomena, *finite-element models* need to be included in our framework. In addition, our current system only includes deterministic behavior models and needs to be generalized to deal with stochastic models. Furthermore, refined design strategies for component object and model libraries will be developed so that the libraries will have appropriate sizes and structures that will facilitate component selection and system level modeling.

ACKNOWLEDGEMENTS

We would like to thank other members of the Composable Simulation Group at Carnegie Mellon University, especially, Rajarishi Sinha, Vei-Chung Liang and Khaled H. Al-Ajmi, for sharing their insights in simulation-based design and virtual prototyping.

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 National Science Foundation under grants # CISE/115/KDI 98 73005 and # EIA-97 29827, by the Pennsylvania Infrastructure Technology Alliance, and by the Institute for Complex Engineered Systems at Carnegie Mellon University.

REFERENCES

[1] G. Pahl and W. Beitz, *Engineering design: A systematic approach*, 2nd ed. London, U.K.: Springer-Verlag, 1996.

[2] S. B. Shooter, W. Keirouz, S. Szykman, and S. J. Fenves, "A model for the flow of design information," presented at ASME DETC 2000, 12th International Conference on Design Theory and Methodology, Baltimore, MD, 2000.

[3] A. Sydow, "Hierarchical Concepts in Modeling and Simulation," in *Progress in Modeling and Simulation*, F. E. Cellier, Ed. London: Academic Press, 1982.

[4] G. Zhang and B. P. Zeigler, "The system entity structure: Knowledge representation for simulation modeling and design," in *Artificial Intelligence, Simulation and Modeling*, L. E. Widman, K. A. Loparo, and N. R. Nielsen, Eds. New york: Wiley, 1989, pp. 47-73.

[5] J. Keown, *Orcad Pspice and Circuit Analysis*, 4th ed: Prentice Hall, 2000.

[6] IEEE, *1076-1993 IEEE Standard VHDL Language Reference Manual*: IEEE, 1993.

[7] IEEE, *1076.1 Working Group: Analog and mixed-signal extensions for VHDL*: IEEE, 1999.

[8] A. Shabana, "Flexible multibody dynamics: review of past and recent developments," *Multibody System Dynamics*, vol. 1, pp. 189-222, 1997.

[9] W. Schiehlen, "Multibody system dynamics: roots and perspectives," *Multibody System Dynamics*, vol. 1, pp. 149-188, 1997.

[10] Aubert and Garcia-Sabiro, "VHDL-AMS, an unified language to describe multi-domain, mixed-signal designs, mechatronic applications," presented at FDL 1999: 2nd Forum on Design Languages, France, 1999.

[11] R. Lutz, R. Scrudder, and J. Graffagnini, "High Level Architecture Object Model Development and Supporting Tools," *Simulation*, vol. 71, pp. 401-409, 1998.

[12] H. Elmqvist, F. Boudaud, J. Broenink, D. Brück, T. Ernst, P. Fritzon, A. Jeandel, K. Juslin, M. Klose, S. E. Mattsson, M. Otter, P. Sahlin, H. Tummescheit, and H. Vangheluwe, *Modelica - A Unified Object-Oriented Language for Physical Systems Modeling*, 1997.

[13] H. Elmqvist, S. E. Mattsson, and M. Otter, "Modelica: The new object-oriented modeling language," presented

at The 12th European Simulation Multiconference, Manchester, UK, 1998.

[14] B. P. Zeigler, *Object-oriented simulation with hierarchical, modular models*: Academic Press, 1990.

[15] B. P. Zeigler, H. Praehofer, and T. G. Kim, *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, 2nd ed: Academic Press, 2000.

[16] 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," *Computers and Chemical Engineering*, vol. 15, pp. 53-72, 1991.

[17] P. A. Fishwick, "Integrating Continuous And Discrete Models With Object Oriented Physical Modeling," presented at 1997 Western Simulation Multiconference, Phoenix, Arizona, 1997.

[18] M. Anderson, "Object-oriented modeling and simulation of hybrid systems," in *Department of Automatic Control*. Lund, Sweden: Lund Institute of Technology, 1994.

[19] F. E. Cellier, "Object-oriented modeling: means for dealing with system complexity," presented at 15th Benelux Meeting on Systems and Control, Mierlo, Netherlands, 1996.

[20] A. Diaz-Calderon, C. J. J. Paredis, and P. K. Khosla, "Reconfigurable Models: A Modeling Paradigm to Support Simulation-Based Design," presented at 2000 Summer Computer Simulation Conference, Vancouver, Canada, 2000.

[21] B. P. Zeigler and C.-J. Luh, "Model based management for multifacetted systems," *ACM Transactions on Modeling and Computer Simulation*, vol. 1, pp. 195-218, 1991.

[22] H. C. Park and T. G. Kim, "A relational algebraic framework for VHDL models management," *Transactions of the Society for Computer Simulation International*, vol. 15, pp. 43-55, 1998.

[23] A. P. J. Breunese, J. L. Top, J. F. Broenink, and J. M. Akkermans, "Library of Reusable models: Theory and Application," *Simulation*, vol. 71, pp. 7-22, 1998.

[24] V. Devedzic, "A survey of modern knowledge modeling techniques," *Expert systems with applications*, vol. 17, pp. 275-294, 1999.

[25] S. Szykman, "Design Respositories: Engineering Design's New Knowledge Base," *IEEE Intelligent Systems*, vol. 15, pp. 48-55, 2000.

[26] S. Szykman, S. J. Fenves, S. B. Shooter, and W. Keirouz, "A foundation for interoperability in next-generation product development systems," presented at 2000 ASME Design Engineering Technical Conferences (20th Computers and Information in Engineering Conference), paper No. DETC2000/CIE-14622, Baltimore, MD, 2000.

[27] 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, 1999.

[28] A. Diaz-Calderon, C. J. J. Paredis, and P. K. Khosla, "Automatic generation of system-level dynamic equations for mechatronic systems," *Journal of Computer-Aided Design*, vol. 32, pp. 339-354, 2000.

[29] C. J. J. Paredis, A. Diaz-Calderon, R. Sinha, and P. K. Khosla, "Composable Models for Simulation-Based Design," *Engineering with Computers*, vol. in press, 2001.

[30] R. Sinha, C. J. J. Paredis, and P. K. Khosla, "Modeling of Component Interactions in Configuration Design," Carnegie Mellon University, Pittsburgh, PA, Technical Report 2001.

[31] R. Sinha, C. J. J. Paredis, and P. K. Khosla, "Integration of mechanical CAD and behavioral modeling," presented at Proceedings 2000 IEEE/ACM International Workshop on Behavioral Modeling and Simulation, Orlando, FL, USA, 2000.

[32] A. Diaz-Calderon, C. J. J. Paredis, and P. K. Khosla, "Organization and selection of reconfigurable models," presented at Proceedings of WSC 2000, Winter Simulation Conference, Orlando, FL, USA, 2000.

[33] C. Brodley, "Dynamic Automatic Model Selection," University of Massachusetts, Amherst, Technical Report 92-30, February 1992.

[34] Levy, Iwasaki, and Fikes, "Automated Model Selection for Simulation Based on Relevance Reasoning," *Artificial Intelligence*, vol. 96, pp. 351-394, 1997.

[35] N. Meade and T. Islam, "Technological Forecasting-Model Selection, Model Stability, and Combining Models," *Management Science*, vol. 44, pp. 1115-1130, 1998.