Composable Models for Simulation-Based Design

Christiaan J. J. Paredis, Antonio Diaz-Calderon, Rajarishi Sinha, Pradeep K. Khosla

Carnegie Mellon University
Institute for Complex Engineered Systems
Department of Electrical and Computer Engineering
Pittsburgh, PA 15213
{cjp, adiaz, rsinha, pkk}@cs.cmu.edu

Abstract

This article introduces the concept of combining both form (CAD models) and behavior (simulation models) of mechatronic system components into component objects. *By connecting these component objects to each other through their ports, designers can create both a system-level design description and a virtual prototype of the system. This virtual prototype, in turn, can provide immediate feedback about design decisions by evaluating whether the functional requirements are met in simulation.*

To achieve the composition of behavioral models, we introduce a port-based modeling paradigm. The port-based models are reconfigurable, so that the same physical component can be simulated at multiple levels of detail without having to modify the system-level model description. This allows the virtual prototype to evolve during the design process and to achieve the accuracy required for the simulation experiments at each design stage.

To maintain the consistency between the form and behavior of component objects, we introduce parametric relations between these two descriptions. In addition, we develop algorithms that determine the type and parameter values of the lower pair interaction models; these models depend on the form of both components that are interacting.

This article presents the initial results of our approach. The discussion is limited to high-level system models consisting of components and lumped component interactions described by differential algebraic equations. Expanding these concepts to finite element models and distributed interactions is left for future research.

Our composable simulation and design environment has been implemented as a distributed system in Java and C++, enabling multiple users to collaborate on the design of a single system. Our current implementation has been applied to a variety of systems ranging from consumer electronics to electrical train systems. We illustrate its functionality and use with a design scenario.

1 Introduction and Motivation

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 through virtual prototyping (Haas and Jasnoch 1994). A virtual prototype enables the designers to test initially whether the design specifications are met by performing simulations rather than physical experiments. 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.

Virtual prototypes need to model the behavior of the equivalent physical prototype adequately accurately; otherwise, the predicted behavior does not match the actual behavior resulting in poor design decisions. But creating accurate models is a hard problem. Only recently has computing performance reached a level where high fidelity simulation models are economically viable. For instance, it is now feasible to evaluate dynamic simulations of finite element models for crack propagation (Swenson and Ingraffea 1988; O'Brien and Hodgins 1999). However, not always are the most detailed and accurate simulation models also the most appropriate; sometimes it is more important to evaluate many different alternatives quickly with only coarse, high-level models. For instance, at the early stages of the design process, detailed models are often unnecessary because many of the design details still have to be decided and accurate parameter values are still unknown. At this stage, the accuracy of the simulation result depends more on the accuracy of the parameter values than on the model equations; simple equations that describe the high-level behavior of the system are then most appropriate.

Equally important to accuracy is the requirement that simulation models be 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 article introduces such a paradigm, *composable simulation and design,* which is based on model composition from system components.

2 Composable Simulation and Design

To provide better support for simulation-based design of mechatronic systems, we have developed a simulation 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. Some other products have a modular product architecture allowing them to be customized for a particular application or mass-produced at low cost (Baldwin and Clark 2000). The design of these categories of products consists primarily of the configuration or assembly of existing components or modules.

The building blocks within our composable simulation and design environment are *component objects,* illustrated in Figure 1. These objects consist of a configuration interface (a list of ports), CAD model(s), behavioral model(s), and relationships between them.

The configuration interface of a component object consists of *ports*. A port defines an intended interaction between a component and its environment. For instance, the configuration interface of the AC motor in Figure 1 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.

The behavioral models in the component objects are also defined 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

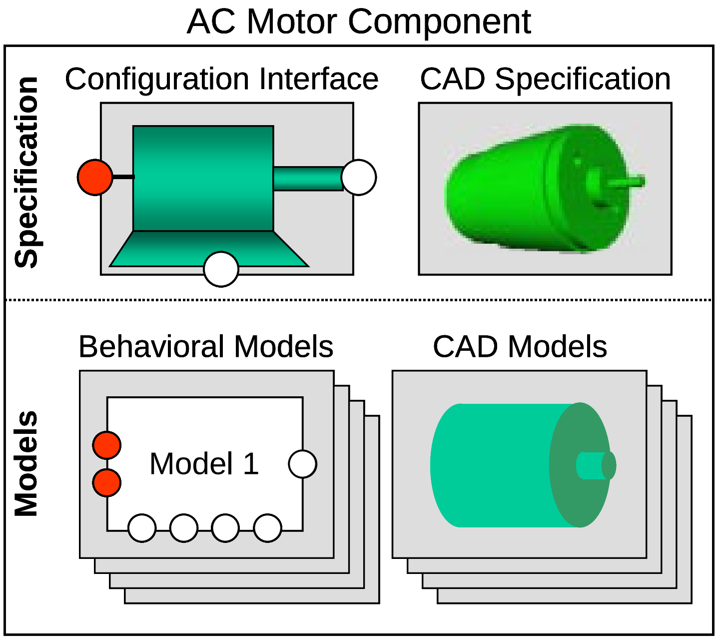


Figure 1: Component objects consist of a port-based interface for system configuration, combined with CAD and behavioral models.

AC motor corresponds to a mechanical energy port, while the AC plug is modeled as two electrical ports, one for each pin. We will describe port-based behavioral modeling in more detail in Section 4.

The CAD models in component objects serve a dual role. On the one hand, a CAD model is 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. Depending on the required accuracy of the analysis, these CAD models may be used to describe the component at different levels of detail. The component object also includes relationships between the ports and parameters in the configuration interface and certain form features and characteristics of the CAD model. This will be further explained in Section 6.

Multiple component objects can be configured into larger systems by connecting their ports. As is shown in Figure 2, the design prototype consisting of the pulley mounted onto the motor shaft can be represented by connecting the shaft port of the pulley to the rotor port of the motor. This configuration specifies the prototype completely: it specifies which components to use and how to configure these components.

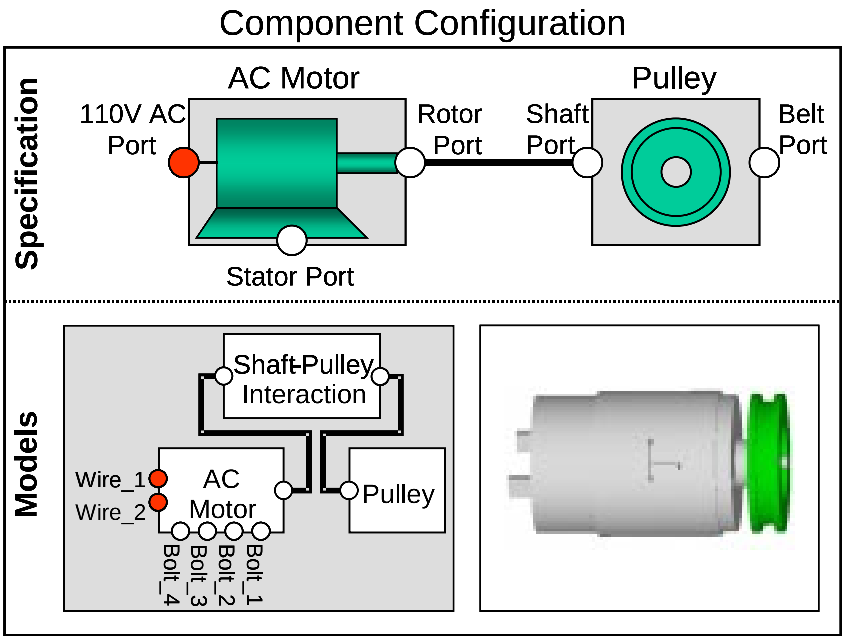


Figure 2: Component objects can be hierarchically configured into complex systems. At the same time, the behavioral and CAD models are configured also.

In their framework for System Design for Reusability (SyDeR), Feldkamp et al. (Feldkamp et al. 1998) provide an interface to hierarchically specify modular systems through port-based composition. Our approach goes beyond the specification of the design prototype, and further includes analysis capabilities by including CAD and behavioral models.

Because the *modeling* of systems described as component configurations can also be viewed as composition, we can obtain a system level simulation model by combining the behavioral models of the individual components. One important difference between the configuration of component objects and the configuration of their behavioral models is the inclusion of models that capture the dynamics of the *interactions* through the ports (friction, electro-magnetic interference, contact resistance, etc). The role of interaction models is further investigated in Section 6.2.

By taking advantage of the parallelism between composition in configuration design and composition in simulation modeling, our framework allows a designer to simultaneously design and model new artifacts. This is already common practice in electrical CAD software (Mentor Graphics 2000); when creating a chip layout, the instantiation of a transistor or logic gate creates the geometry for the silicon layers as well as the corresponding simulation model. In mechanical CAD, the integration between design and simulation is not as common. For purely mechanical systems, most commercial CAD packages do provide an optional module for multi-body simulation, but these modules do not support port-based configuration and lack sufficient support for multi-disciplinary systems. The main goal of our simulation and design environment is to extend these ideas to *simulation-based design of multidisciplinary systems* within an integrated software environment.

We believe that the concept of component objects is general and that the composition of port-based objects can be applied to many different application areas, energy domains, and levels of model accuracy. However, in our current research, we have applied this framework only to system-level modeling of mechatronic systems (Diaz-Calderon et al. 1999; Sinha et al. 2000); that is, modeling of computer-controlled electro-mechanical systems using differential algebraic equations (DAEs) (Ascher and Petzold 1998) and/or discrete event systems specifications (DEVS) (Zeigler et al. 2000).

Furthermore, the port-based modeling paradigm, as presented in this article, is limited to systems with lumped interactions. When an interaction is distributed in nature, as between a boat and the water on which it floats, it must be approximated by a large number of lumped interactions. The internal model of a component, however, may still be distributed. Consider, for example, a flexible beam attached to a structure by its two ends. A finite element model may describe the internal behavior of the beam, but, by defining a mapping between the lumped port variables and distributed boundary conditions of the finite element model, the interaction with the rest of the structure can still be captured with only two ports. For mechatronic systems, the primary interactions between components tend to be lumped, so that the port-based modeling paradigm is applicable. Only when more detailed models are required, may we have to consider phenomena, such as thermal interactions, that are distributed in nature. In the future, we plan to expand our modeling paradigm to different energy domains, and distributed interactions.

Our framework for simulation and design has the following characteristics, which we will address in detail in the subsequent sections:

A port-based modeling paradigm: To take advantage of the compositional nature of both design and modeling of mechatronic systems, we use a port-based modeling paradigm in which the user can compose system-level simulations from component models. By connecting the ports of the subcomponents, the user defines the interactions between them. This port-based modeling paradigm builds on object oriented modeling languages such as VHDL-AMS (IEEE 1999) and Modelica (Mattsson et al. 1998), and is explained in more detail in Section 4.

Reconfigurable Models: At each stage of the design process, the designer performs different simulation experiments to verify whether the design prototype meets the functional requirements. In the early, conceptual stage, these experiments may include quick trade-off analyses that require limited accuracy, while towards the end of the detailed design stage, the designer may decide to perform a comprehensive, detailed simulation. To accommodate simulations at different levels of detail without the need for remodeling the complete system, we develop the concept of reconfigurable models in Section 5. These models can evolve with the design prototype throughout the design process.

Simulation integrated with CAD: The building blocks in our simulation and design environment are *component objects*; they describe both the form and the behavior of system components. In Section 6, we describe how the CAD description of the form may be used to extract the lumped parameters of the behavioral models. In addition, we have developed algorithms that instantiate models of mechanical *interactions* based on the form of the interacting components.

A component library: The component objects are organized in a hierarchical component library. From this library, the designer selects the components that achieve the desired functionality within the system. We provide a detailed description of the component library and its implementation in Section 7.

3 Related Work

3.1 Modeling and Simulation

There exist already many modeling paradigms and commercial simulation packages. They can be characterized according to the following criteria: graph-based versus language-based, multi-domain versus single-domain, and declarative versus procedural modeling.

The best known of the graph-based modeling paradigms is Bond Graph modeling (Paynter 1961; van Dixhoorn 1980; Rosenberg and Karnopp 1983; Karnopp et al. 1990). It is based on energy-conserving junctions that connect energy storing or transforming elements with bonds; the bonds represent the energy flow between the modeling elements. Bond graph modeling has the advantage that it is domain independent and based on energy flow, but it is not very convenient for the modeling of 3D mechanics or continuous-discrete hybrid systems. Furthermore, beginning users find it counterintuitive that the topology of a bond graph is different from the topology of the corresponding physical system.

Linear graph models do reflect the system topology directly (Trent 1955; Branin 1966). They are also domain independent and can be easily extended to model 3D mechanics (Andrews et al. 1988; Richard et al. 1995; McPhee 1996) and hybrid systems (Roe 1966; Muegge 1996). The VHDL-AMS language, which we use for modeling, builds on the concepts of linear graph modeling, although it does not require an explicit graph representation (Christen et al. 1999; IEEE 1999).

The majority of modeling paradigms is not graph-based, but language-based. A large number of modeling languages are derived from the CSSL (continuous system simulation language) standard developed by the Technical Committee of the Society for Computer Simulation (Strauss et al. 1967). These languages have in common that they are procedural. A model is defined by a procedure that computes the derivatives of the state for a given state and time. A second group of modeling languages is equation-based or declarative: Modelica (Elmqvist et al. 1998), Easy5 (The Boeing Company 1999), Dymola (Dynasim AB 1999), Omola (Anderson 1994), and VHDL-AMS (IEEE 1999). Here, the model is defined by a set of equations that establishes relations between the states, their derivatives, and time. A model compiler is responsible for converting these equations into a software expression that can be evaluated by the computer.

The advantage of declarative languages is that the user does 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. Many of the declarative languages are also object-oriented and support multiple energy domains. This is the case for VHDL-AMS and Modelica, which have the additional advantage that they support both continuous time and discrete time systems simulation.

The modeling paradigm presented in this article builds on the current state-of-the-art modeling languages (Modelica and VHDL-AMS). The reconfigurable port-based models, introduced in Section 4 and 5, are compiled into either Modelica or VHDL-AMS models once the parameter values have been extracted from the CAD data and the user has specified the implementation bindings.

3.2 Simulation-based Design

Many companies are resorting to simulation tools to improve their design process. A well-publicized example of virtual prototyping is the design of the Boeing 777 airplane (Upton 1998). Boeing switched from a paper-based design process to a digital CAD representation, allowing them to perform some of the performance analysis (using CFD software) and assemblability analysis without the need for building physical prototypes. This resulted in a shorter design and testing period. A similar all-digital approach is also being adopted by car manufacturers (Bullinger et al. 1999).

Although the success of simulation-based design has already been demonstrated commercially (Upton 1998; Bullinger et al. 1999), many unresolved research issues remain to be addressed. Ongoing research includes model validation, automatic meshing and model creation, integration of simulation engines in different domains, architectures for collaboration, and visualization using virtual reality technology. In this article, we focus on simplifying the process of model creation, by integrating form and behavior into component objects.

Our approach is based on the characterization of a design prototype by its form, function, and behavior (Pahl and Beitz 1996; Shooter et al. 2000). The form is a description of the physical embodiment of an artifact, while function is the purpose of the artifact—the behavior that the designer intended to achieve. As is illustrated in Figure 3, the actual behavior does not depend on the function, but only on the form. During design or synthesis, we instantiate a form to satisfy a given function, while, during design verification, we derive the behavior from the form and

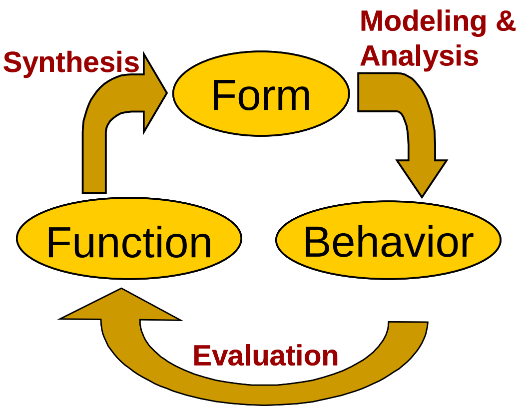


Figure 3: The relation between form, function, and behavior in the context of virtual prototyping.

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.

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 (de Vries and Breunese 1995; Shooter et al. 2000). During this process, the design team adds information and thus transforms the design representations. For instance, a needs assessment is transformed into design specifications and engineering requirements; engineering requirements, in turn, are converted into a family of solutions that are evaluated and compared (possibly using simulation) to iterate on the description of the artifact in terms of form, function, and behavior (Pahl and Beitz 1996). As a result, all representations evolve simultaneously from the initial high-level decompositions to increasingly detailed descriptions of the design artifact.

In the early stages of the design process, when only few physical details have been defined, 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 or form are included in design artifacts, these high-level models can be replaced gradually by more detailed behavioral models of the physical components. The modularity and encapsulation of our port-based modeling paradigm facilitates these model substitutions.

4 Port-Based Modeling Paradigm

To achieve composability of behavioral models, we have developed a port-based modeling paradigm. This paradigm is based on two concepts: *ports* and *connections* (Diaz-Calderon et al. 2000a; Diaz-Calderon et al. 2000c).

Ports correspond to the points where a component exchanges energy or signals with the environment. All energy is exchanged through ports. There is one port for each separate interaction point, and the type of a port matches the type of the energy exchange. For example, a DC motor has four ports, two electrical and two mechanical. The electrical ports correspond to the electrical connectors of the motor, the mechanical ports to the stator and the rotor—one port for each rigid body.

The energy flowing through a port is characterized by an *across* and a *through* variable, also called *effort* and *flow* variables in Bond Graph modeling (Paynter 1961; Rosenberg and Karnopp 1983). Examples of across variables are voltage in the electrical domain and velocity in the mechanical domain. They are measured across the port relative to a global reference. The corresponding through variables, electrical current and mechanical force, are measured through the port.

The interactions between component models are represented by *connections* between ports. Each connection imposes algebraic constraints on the port variables. These constraints are the equivalents of the Kirchhoff voltage and current laws in electrical circuits. One type of constraint requires that the across variables be equal, the other that the sum of the through variables be zero. A single energy connection in our framework is equivalent to two connections in block-diagram modeling languages such as SimuLink (The Mathworks Inc. 1999). In block diagrams, all interactions occur through signals. The user is responsible for determining which of these signals are dependent and which are independent, that is, the mathematical causality of the model.

Combining across and through variables in a single connection allows us to model components and ports as declarative equations rather than procedural assignments. Many recent simulation languages are declarative, including Modelica (Elmqvist and Mattsson 1997), VHDL-AMS (IEEE 1999), and Dymola (Dynasim AB 1999); SimuLink (The Mathworks Inc. 1999), on the other hand, is procedural. When solving a set of declarative differential equations, the solver must first determine the mathematical causality of the equations.

As a reflection of the underlying physics, the declarative representations of both energy connections and energy ports are undirected. An electrical resistor, modeled by $V = RI$, does not have an input and an output (no predetermined mathematical causality), and the energy through its ports can flow in either direction (no sign restrictions on the flow variable). Since the model is declarative, the solver may instantiate this single equation as either $V = RI$ or $I = V/R$, depending on how the resistor is used in the circuit. Even when considering the heat dissipated by the resistor, $Q = RI^2$, the user does not have to worry about the direction of the heat transfer or its causality as defined by the second law of thermodynamics. The solver will recognize that Q is always positive and that the only valid mathematical causality assignment is to compute the dissipated heat from the voltage and current; the opposite causality would impose simultaneous constraints on the voltage and current through the resistor, resulting in overconstrained equations for the electrical system.

All the ports combined form the interface of the model. This interface defines how the component can interact with the other components in the system, but does not contain any information about the internal behavior of the component. Instead, the interface *encapsulates* the implementation of the model, which defines the internal behavior of the component

As illustrated in Figure 4, a port-based model can be hierarchically defined when it consists of a composition of sub-models, resulting in a *compound component*. When the sub-models are also compound, multiple levels of hierarchy occur. The bottom of the hierarchy consists of *primitive models* that are defined only by their constitutive equations; these relate the across and through variables of a component model. For example, the constitutive equation for a resistor relates the voltage difference between the two ports with the current through the ports according to Ohm's law, $V = RI$. In general, the model equations may include a combination of both algebraic and ordinary differential equations.

In addition to ports and connections that model energy flow, we also consider signal ports and signal connections. No energy flows through signal ports, and the connections between them are directed. This reflects the physics of a low-impedance electrical output driving a high-impedance input; the signal can only flow from the output to the input, an operation that requires almost no power (Sedra and Smith 1997). Examples of systems with signals are computer networks, data buses, or embedded controllers; they can be modeled as block diagrams similar to SimuLink models (The Mathworks Inc. 1999). Signal components are defined by *procedures* rather than constitutive equations. Procedures differ from constitutive equations in that their mathematical causality is fixed (inputs are independent, and outputs are dependent variables). Most mechatronic systems contain both energy-based and signal components and are thus hybrid systems (Shetty and Kolk 1997).

5 Reconfigurable Models

Most object-oriented modeling languages have a concept similar to ports (sometimes called terminals, or connectors) (Anderson 1994; Sahlin 1996; Elmqvist et al. 1998; IEEE 1999), and it is possible to use these languages to describe the composable and hierarchical port-based models introduced in the previous section. However, these languages do not guarantee a clear separation

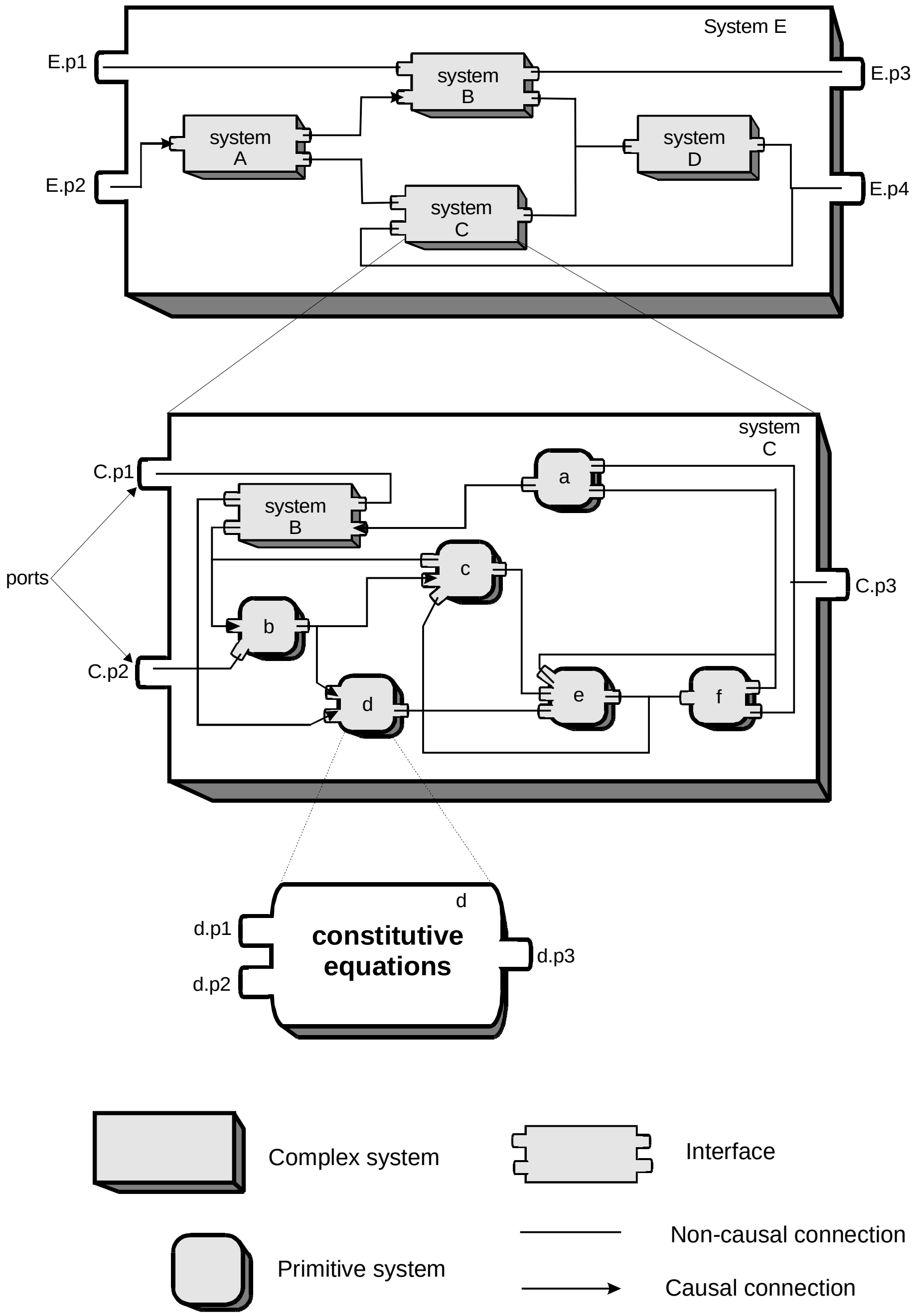


Figure 4: Port-based simulation models may be hierarchically defined.

between the interface of the model and the implementation of its behavior, often merging both concepts into a single modeling object. To create models that can evolve with the design, we need the capability to bind different implementations to the same interface, allowing the designer to select a more detailed behavior for a component, without having to remodel its interactions with the rest of the system.

In traditional object-oriented modeling languages, the behavior of a model can only be modified by changing the values of the parameters. In our approach, the *structure* of the model can be modified also, resulting in *reconfigurable models*. In a reconfigurable model, the interface of the model and the implementation of its behavior are defined separately. As is illustrated in Figure 5, each implementation has a corresponding interface, but a single interface may have multiple implementations associated with it.

In the definition of reconfigurable models, we consider two principles: *composition* and *instantiation*.

Through the principle of composition, a model implementation can be defined as a set of sub-component interfaces and the interactions between them, as in implementation A in Figure 5. At this point, the sub-components do not yet have any behavior; they are represented only by their interface. This allows us to define the interactions between sub-components independently of their internal behavior. One can think of an interface as the equivalent of an abstract class in object oriented programming; it defines the methods through which one can interact with the object, but it does not provide an implementation.

An abstract model becomes concrete by instantiating an implementation. Only

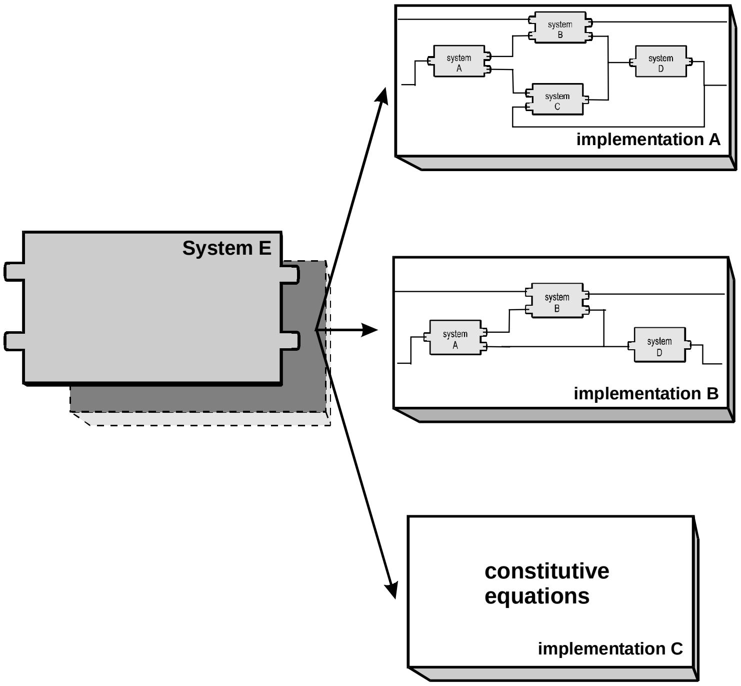


Figure 5: A reconfigurable model consisting of an interface and three implementations.

implementations that match the interface can be instantiated. Moreover, the semantics of the implementation must match the semantics of the interface. For example, the interface of an electrical resistor has the same ports as the interface of a capacitor. However, because the semantics of the two components are different, only resistor implementations can be instantiated for a resistor interface.

Like general port-based models, reconfigurable models can be hierarchical. Because compound implementations are a composition of abstract interfaces, they themselves are also abstract. The instantiation of a compound model, therefore, requires the recursive instantiation of all the interfaces of its subcomponents. The number of possible configurations of a compound model can grow very large when considering all possible combinations of implementation bindings. We call this set the model space of the component. The advantage of reconfigurable models is that all the elements of the model space can be instantiated without having to redefine the interactions with other system components because the interface remains the same.

The instantiation principle also allows the definition of families of components. In this case, the implementations for an interface do not represent different behavioral models for a single component, but instead represent models for a family of components that all share the same interface. For example, a family of DC motors may all share an interface consisting of two mechanical and two electrical ports. When designing a system, the designer can include this interface in the system model without having to select a particular DC motor. It may be possible to select the most appropriate DC motor, by performing a series of simulation experiments each with a different motor from the family; each new experiment only requires that a new implementation be bound to the DC motor interface.

As is illustrated in Figure 6, the set of implementations for an interface can be represented as an AND-OR tree (Diaz-Calderon et al. 2000c). The structure of the AND-OR tree and the principles of composition and instantiation are closely related. AND arcs point from an implementation to the interfaces from which it is composed. Similarly, OR arcs point from an interface to the implementations from which it can be instantiated. For example, the AND-OR tree in Figure 6 depicts a DC-motor model. The top-level interface has three different implementations associated with it, represented by three OR arcs. The electro-mechanical model implementation has three AND arcs, meaning that it is a compound model consisting of three interfaces: electrical, conversion, and mechanical. When instantiating a particular model in the

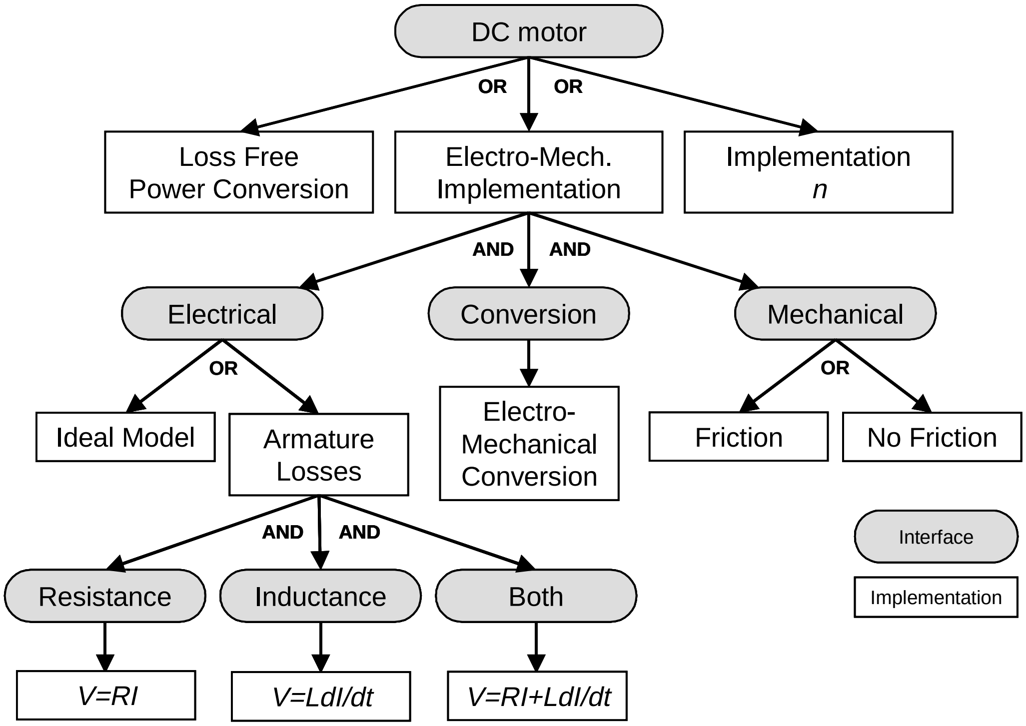


Figure 6: An example AND-OR tree representation of a reconfigurable model.

model space, we must bind implementations to interfaces. Working our way down from the top to the bottom of the AND-OR tree, we must first assign an implementation to the top-level interface and then recursively to each of the interfaces that constitute the selected implementations.

In general, a component object can contain multiple behavioral models, describing the component at different levels of detail. Sometimes it is possible to capture this set of behavioral models in a single reconfigurable model, as described above. This requires, however, that all behavioral models have the same interface, a condition that may not always be satisfied. For instance, if we decided to model the thermal losses in the DC-motor in Figure 6, the interface would have to be expanded to include a thermal port. Our current research is addressing the issues that arise when including behavioral models with different interfaces.

6 Relation between Behavior and Form

Composable simulation and design are based on the concept of component objects that combine form and behavior. By composing component objects into systems, a designer simultaneously designs and models new artifacts. The previous two sections introduced a modular modeling paradigm that supports such composition. In this section, we focus on maintaining consistency between these behavioral models and the corresponding form descriptions as represented by the CAD specification model.

In model compositions, we distinguish between two different types of behavioral models: models representing physical components, and models representing interactions between components. Examples of physical components are motors, screws, shafts, or controllers. Their component objects contain a description of both form and behavior. Interaction models, on the other hand, only occur when two component objects are connected to each other. They do not have associated form, but their model parameters can be extracted from the form of the two interacting components. Examples of interaction models are lower pairs that result from mechanical contact, contact resistance in an electrical switch, or magnetic forces between two magnets.

6.1 Form and Behavior of Component Families

A component object contains a specification of the form of the component as well as CAD models and reconfigurable models describing its behavior. The reconfigurable models may describe the component at different levels of abstraction or with respect to different energy domains, and provides, in this way, different views of the component. Similarly, multiple CAD models may provide different views, at multiple levels of detail, of the geometry of the component.

When a component object is defined as a composition of sub-components, both the form specification and the models are derived automatically by applying the compositional concepts described in the previous sections. When a component is not compound but primitive, however, a fair amount of work is required to define a CAD specification of the form, additional CAD models for visualization, behavioral models, and the relationships between them.

To facilitate the specification of primitive component objects, one can group them into families. A family of component objects is parameterized by one or more instantiation parameters that, when assigned particular values, completely specify the form of the component. All other parameters describing the geometry or behavior of the component can then be derived from these instantiation parameters.

For instance, as is illustrated in Figure 7, given the value for a single instantiation parameter (the model-type of a DC-motor), a lookup table provides all the parameters specifying the form of the configuration ports as well as the parameters of all behavioral and CAD models. This CAD geometry may simply be a high-level abstraction, capturing only the external geometry through which the motor can interact with other components.

In a second example, we can automatically generate behavioral models for component families specified by *parametric CAD models*. In a parametric CAD model, the designer establishes relationships between certain geometric dimensions or parameters. As a result, the form is completely defined by a limited set of characteristic parameters or features, the instantiation parameters. The parameters in the behavioral models can, in turn, be derived from the CAD parameters. As is illustrated in Figure 7, the flow resistance of a hydraulic pipe depends on its length, diameter, and bending radii. Although these dimensions may not be defined explicitly in the CAD model, they can be extracted through parametric relations captured as procedures (Shah and Mantyla 1995; Bettig et al. 2000).

Finally, in the most general case, behavioral models can be automatically derived for

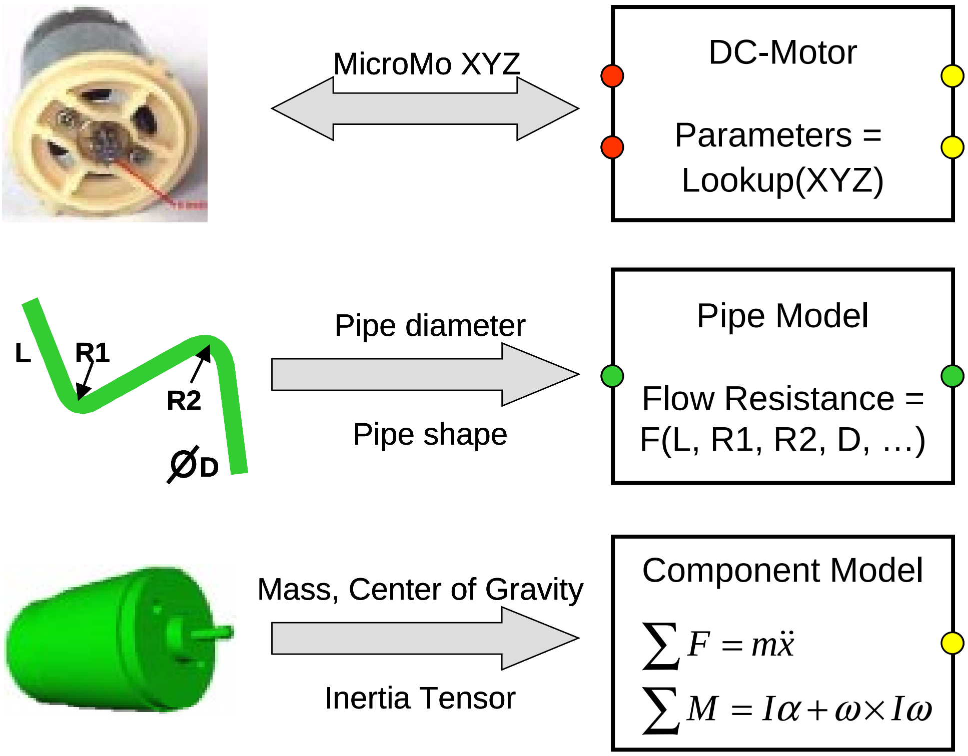


Figure 7: The relation between form and behavior parameters.

components specified by generic CAD models. This requires combining information about geometry and materials with knowledge of the physical phenomena occurring in the component. Creating such models automatically is too difficult in the general case, but can be achieved for certain classes of behavioral models. For example, as is shown in Figure 7, a rigid body model for a component with homogeneous material properties is completely defined by the mass and inertial parameters of the component. Most CAD software packages provide procedures that compute the inertial parameters from the density and the geometry of a part, as defined in a general CAD model. As a result, the behavior models of homogeneous rigid bodies can be derived automatically for any material and arbitrary geometry.

6.2 Form and Behavior of Component Interactions

In addition to the behavioral models of component objects, system models include models describing the *interactions between* component objects. For each pair of interacting component objects, there is an interaction model that relates the port variables of the two objects to each other.

Every interaction requires an interaction model. However, for the electrical domain, the interaction model is usually very simple. An electrical connection between two components is often modeled sufficiently accurately by constraining the voltage at the two connecting ports to be equal and the current through them to add to zero (Kirchhoff's voltage and current laws). Because this interaction model is so common, we allow it to be omitted as shorthand in our modeling paradigm, as is shown in Figure 8.

In the mechanical domain, the equivalent default model is a rigid connection between components (the positions of the reference frames are equal and the forces and torques add to zero). Besides rigid connections, other common mechanical interaction models are the lower pair kinematic constraints. We have developed algorithms to extract the type and parameters of a lower pair from the geometry of the interacting components (Sinha et al. 1998; Sinha et al. 2000). Previously, kinematic analysis was limited to geometry with only planar faces (Mattikalli et al. 1994). When approximating curved faces, which are common in engineering devices, with polygonal facets, these analyses may fail to recognize certain degrees of freedom.

In our work (Sinha et al. 1998; Sinha et al. 2000), we have extended these results to curved contacts, as is shown in Figure 9. When two rigid parts share a surface-to-surface contact, every

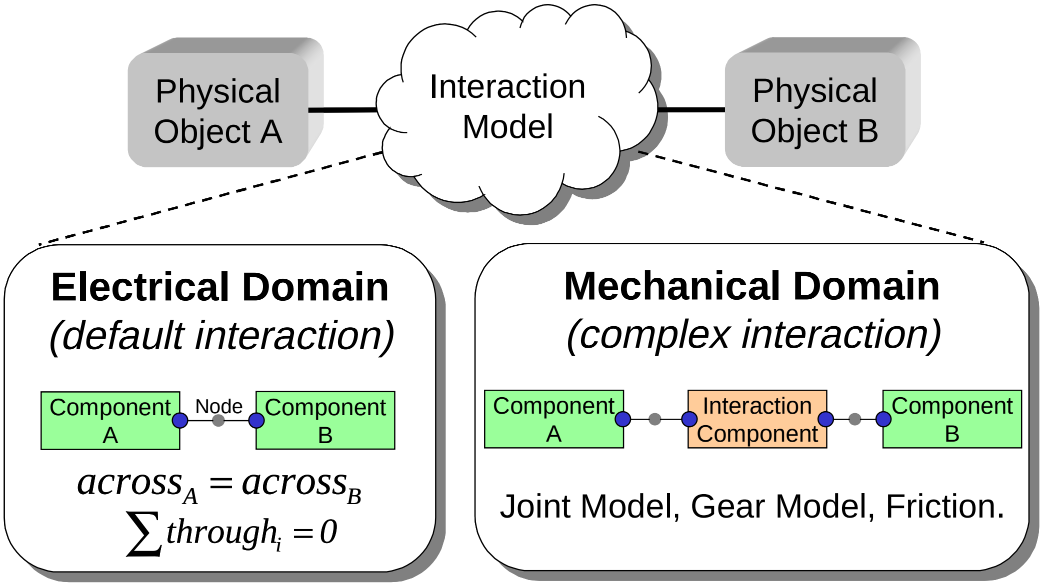


Figure 8: Modeling the nteractions between system components.

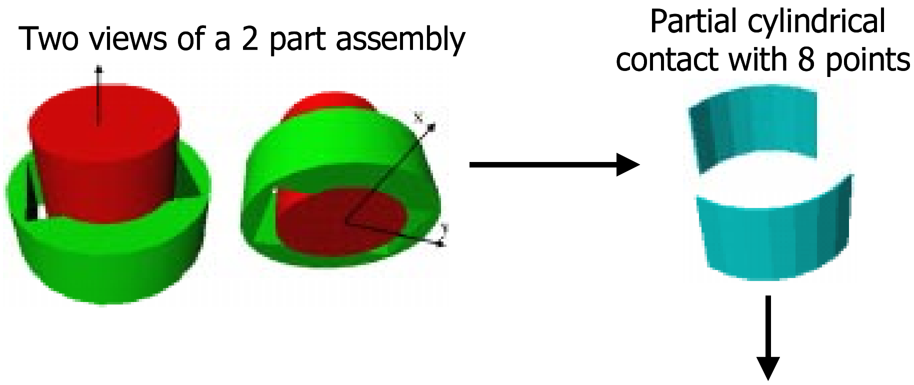
contact point is subject to a non-penetration condition. This condition requires that the instantaneous velocity between the two bodies does not have a component in the direction opposite to the surface normal at the contact point. We write this condition as a linear inequality of the form:

$$(\vec{v} + \vec{\omega} \times \vec{r}) \bullet \vec{n} \geq 0, \tag{1}$$

where $\vec{v}$ and $\vec{\omega}$ are the relative translational and angular velocities between the two bodies, $\vec{r}$ is the position of the point, and $\vec{n}$ is the normal to the contact surface. Imposing Equation (1) at every point on the contact surface is equivalent to imposing the constraint at the vertices of the convex hull (Sinha et al. 1998). For instance, the non-penetration conditions for the two bodies in Figure 9 result in eight equations, one for each of the eight corners of the two contact surfaces. In general, the analysis results in a linear relationship of the form:

$$J_{assembly} \begin{bmatrix} \vec{v} \\ \vec{\omega} \end{bmatrix} \geq 0, \tag{2}$$

where each row of $\boldsymbol{J}_{assembly}$ represents a non-penetration constraint, as in Equation (1). From the properties of the $\boldsymbol{J}_{assembly}$ matrix, we can determine the kinematic constraints between two interacting component objects. For example, the basis vectors of the nullspace of $\boldsymbol{J}_{assembly}$ define the contact-preserving degrees of freedom, as is illustrated in Figure 9.



$$\begin{bmatrix} -0.5\sqrt{2} & 0.5\sqrt{2} & 0 & 0 \\ -0.5\sqrt{2} & 0.5\sqrt{2} & -5\sqrt{2} & -5\sqrt{2} \\ -0.5\sqrt{2} & -0.5\sqrt{2} & 0 & 0 \\ -0.5\sqrt{2} & -0.5\sqrt{2} & 5\sqrt{2} & -5\sqrt{2} \\ 0.5\sqrt{2} & 0.5\sqrt{2} & -5\sqrt{2} & 5\sqrt{2} \\ 0.5\sqrt{2} & 0.5\sqrt{2} & 0 & 0 \\ 0.5\sqrt{2} & -0.5\sqrt{2} & 5\sqrt{2} & 5\sqrt{2} \\ 0.5\sqrt{2} & -0.5\sqrt{2} & 0 & 0 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_x \\ \omega_y \end{bmatrix} \le 0$$

$$v_x = v_y = \omega_x = \omega_y = 0$$

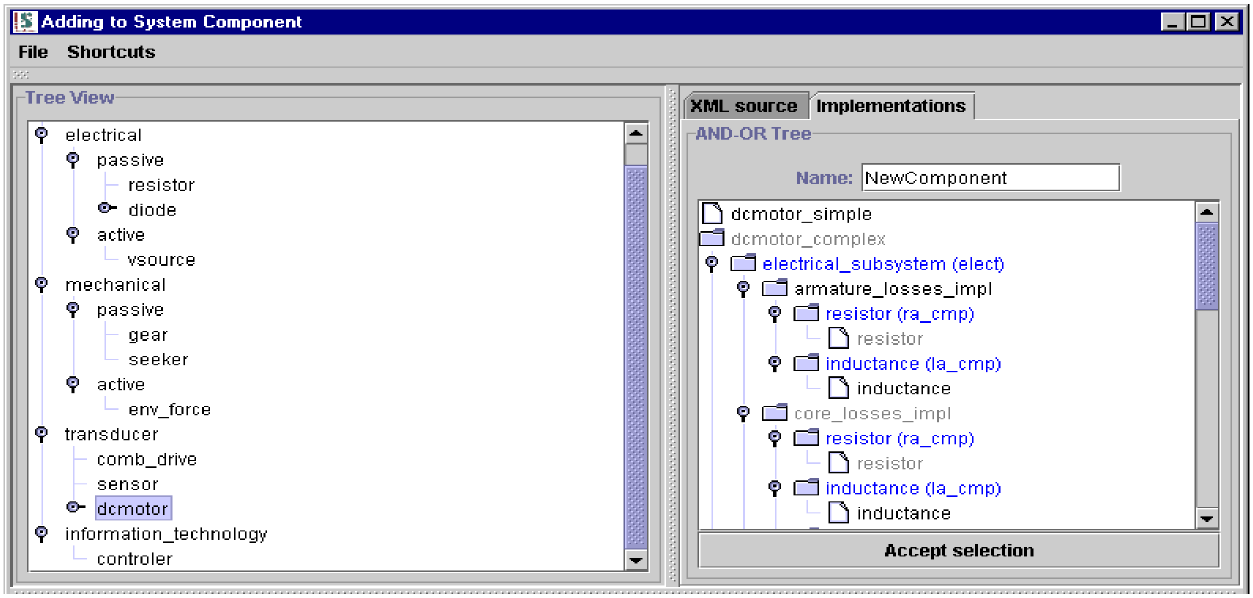
$$v_z, \omega_z \text{ unconstrained}$$

Figure 9: Extracting the type and parameters for lower pair interaction models.

Our method can infer behavior from devices with curved geometry, while at the same time resolving global, multi-part interactions. We have developed procedures that derive the $\boldsymbol{J}_{assembly}$ matrix directly from the CAD models, and from it determine the type and parameters of the interaction models (Sinha et al. 1998; Sinha et al. 2000).

7 Component Libraries

While the previous sections described the properties and characteristics of individual component objects, this section focuses on the organization of multiple component objects into libraries. By searching through the components in these libraries, the designer can locate the appropriate component (or system of components) for a particular desired function. In our current research, we are developing methodologies for assisting the designer in this search process. Such an intelligent synthesis assistant may search the component library based on queries regarding the component's behavior and form. When extending this idea even further, a component object could contain design rules or expert knowledge that allow it to adapt its form to meet the design specifications. Such "intelligent" components are introduced in (Susca et al. 2000).



Name: NewComponent

Figure 10: The component library browser with visualization of the corresponding reconfigurable models.

As is illustrated in Figure 10, we have organized the component objects in a hierarchical taxonomy instead of a flat organization. When moving from the top to the bottom of the hierarchy, the component objects become more concrete. At the top, the objects are abstract and represent families of components, such as the family of electrical two-ports or mechanical rigid bodies; at the bottom, the leaf nodes of the hierarchy represent completely specified physical components. A component can be completely specified for instance by identifying its manufacturer and part number—this allows a manufacturer to implement the design without ambiguity. However, the corresponding behavioral model(s) remain approximations of the actual physical behavior.

We call a component abstract when its implementation is not completely defined: It may not include any implementation, or its implementation may contain one or more unspecified parameter values.

A single component may appear in multiple locations in the taxonomy, depending on the viewpoint for its classification. For example, a DC-motor is an energy conversion component, but can also be considered as a structural element that implements a rotary joint. Conversely, each object in the library includes multiple behavioral views in the form of a reconfigurable model. Figure 10 shows the browser that allows the designer to navigate through the model space of a component, as defined by the AND-OR tree of model implementations.

When including a component into a larger system, the designer has to complete two steps: *component selection* and *model selection*. In the first step, the designer decides which component to use for the implementation of a particular function of the device. Initially, this may be an abstract object that represents a family of components and will later be replaced by a specific instance. For example, initially, the designer decides to use a DC-motor component, which represents the family of all DC-motors, and replaces it later by the specific component object for motor XYZ by company ABC Inc. In the second step, the designer selects the component implementation that is best suited for a particular simulation experiment. For example, the high-level "Loss Free Power Conversion" model of Figure 6 in the early stages of design and the more detailed model, including armature losses and friction, towards the end.

Both the hierarchy of the library and the individual entries are defined in XML format (extensible markup language) (W3C 1999). XML is a neutral and extensible format that can be easily parsed, searched, and shared over the Web. Our XML representation for component objects includes pointers to geometric models (ACIS or Pro/E), an interface definition of the behavioral model, and pointers to the corresponding implementations.

The definition of an implementation is also stored in XML format. The equations tags in primitive implementations are based on the VHDL-AMS standard (IEEE 1999). The component descriptions may also contain meta-knowledge capturing the semantics of the model: What are the assumptions? When is the model valid? Or, what is the meaning of the model? We anticipate using this meta-knowledge extensively when searching for components based on their function. Examples and a more detailed description of the XML model definition format is provided in (Diaz-Calderon et al. 2000b).

8 Software Architecture and Implementation

The implementation architecture of our simulation-based design environment is similar to the Open Assembly Design Environment (OpenADE) developed at NIST (Keirouz et al. 2000). As is shown in Figure 11, the core of our system is a central design database in which the representations for the current design are stored: function, behavior, product structure, and CAD data. Furthermore, the database contains the relationships between these representations; for instance, if a system component implements a particular function, the database will contain a

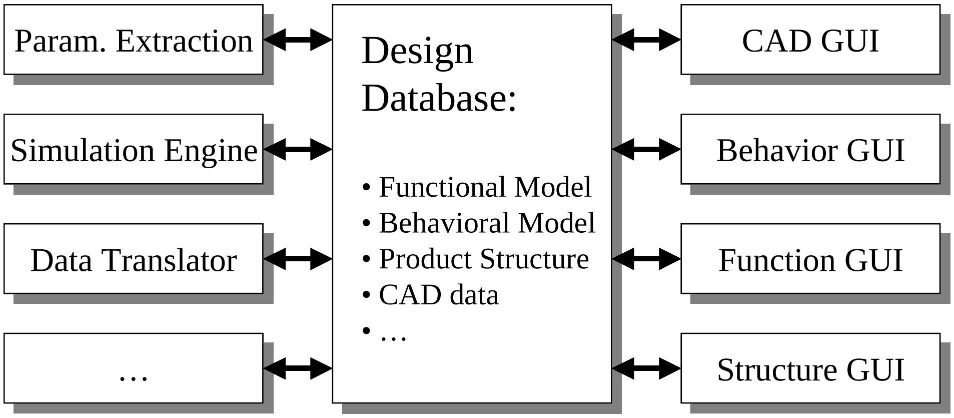


Figure 11: Java-Based GUI components and services interact through a shared design database.

“has_function” relation pointing from the object to its functional model, and an “implemented_by” relation from the model to the object.

During the design process, the information in the central database is continually transformed by autonomous software agents or by the designer—through graphical user interfaces, shown in Figure 12.

The main interaction between the designer and the database occurs through the 3D CAD GUI and the behavior model GUI. The 3D CAD GUI is implemented using the Java3D toolkit. It allows the user to view and manipulate the geometry associated with the system components, and to define mechanical interactions between components. It does not allow the geometry of individual components to be modified; we plan to provide that functionality in the future by integrating our framework with Pro/Engineer. The current Java-based 3D GUI will still remain useful for system-level interactions that do not require the design of new components.

The behavioral modeler provides a 2D view of the system. Each of the system components appears in this view as a port-based model.

In addition to the user interfaces, software agents interact with the design repository. These agents can act as design assistants, working in the background. The tasks performed by such agents include the following:

- the translation of CAD data to VRML format for rendering,
- the extraction and verification of mechanical component interactions,

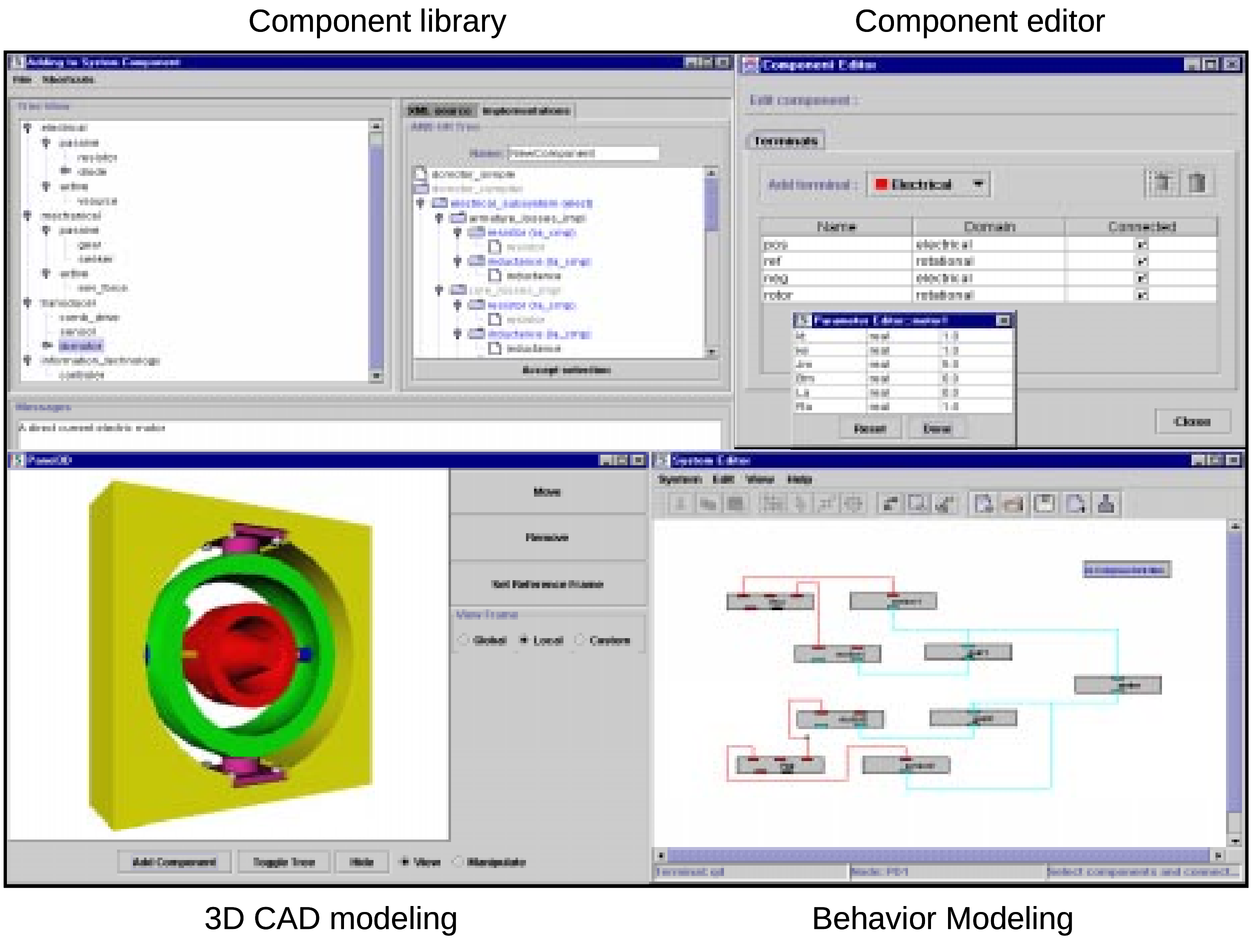


Figure 12: The CAD GUI and Behavioral Model GUI

- the compilation of behavioral models in XML format to VHDL-AMS simulation models.

The framework is implemented in a distributed fashion using Java and C++. The coordination between the distributed software components is event-based (Spell 2000). When a user or a software agent modifies a portion of the design representation, the design database broadcasts an event to all the subscribing agents and GUIs. If necessary, these components will then update their local cache to reflect the changes in the design database. This allows us to maintain consistency between the internal design data and its presentation to the user.

Because of its distributed implementation, our framework can also serve as a tool for collaboration. Multiple users can interact with the same design simultaneously, and design modifications introduced by one user can be propagated immediately to all other users.

9 Example Scenario

To illustrate the use of our composable simulation framework, we examine the design of a missile seeker (Cutkosky et al. 1996). It is not our goal here to present a detailed design case study, but to focus on the use of *modeling and simulation* during the design process.

The seeker is a device with two rotational degrees of freedom that allow it to scan a 2-dimensional area with its camera. Besides the articulated mechanism that realizes the desired degrees of freedom, the seeker consists of actuators, sensors, and embedded controllers for accurate positioning.

9.1 Kinematic Design

As mentioned in Section 3.2, one can think of design as the process of decomposing the function of an artifact, and transforming it into form, such that the form's behavior matches the function (Figure 3). By performing a functional decomposition of the missile seeker, the designer has decided to achieve the two desired degrees of freedom with a serial chain of two rotational joints. He specifies this kinematic function with a ball and stick model and a corresponding simulation model, as shown in Figure 13. This model reflects the intended behavior or function, but no specific physical components have yet been assigned to implement this intended behavior. Nevertheless, the designer can still use our simulator to verify whether these intended kinematics satisfy the design requirements.

9.2 Instantiation of the geometry

Next, the designer instantiates physical components to realize the kinematic structure. The revolute joints of the ball-and-stick model are replaced with DC-motors selected from the component library. Because the designer still needs to determine the dimensions of the motors, he instantiates them with a default parameter set. The corresponding behavioral model represents a complete family of DC-motors, from which he can later select a particular instance.

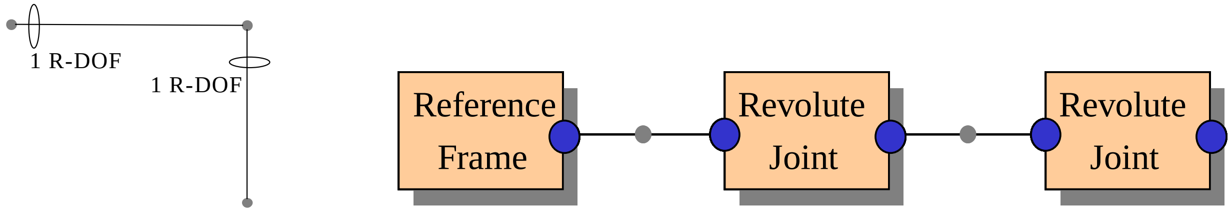


Figure 13: Kinematic model for the 2-DOF seeker.

To connect the motors physically, the designer creates the geometry of a gimbal ring in a CAD package linked to our design environment. This causes the corresponding rigid body model to be instantiated in the system-level behavioral model. From the CAD model, the geometric compiler automatically extracts the mass and inertial parameters, and applies them to the rigid body model. The designer also defines the configuration ports on the gimbal that correspond to the mounting locations of the motors and potentiometers. The resulting design configuration and simulation model is shown in Figure 14.

9.3 Motor Selection

For the next phase of the design, the mechanical engineer who has generated the kinematic structure of the seeker collaborates with a control engineer. From the component library, the control engineer instantiates simple PD controllers that control the position of the two degrees of freedom. Together with the mechanical engineer, he iterates on the selection of an appropriate DC motor. Our simulation framework provides the tools to verify the performance of this multidisciplinary system. The geometrical changes introduced by the mechanical engineer are reflected immediately in the corresponding behavioral models, so that the control engineer can

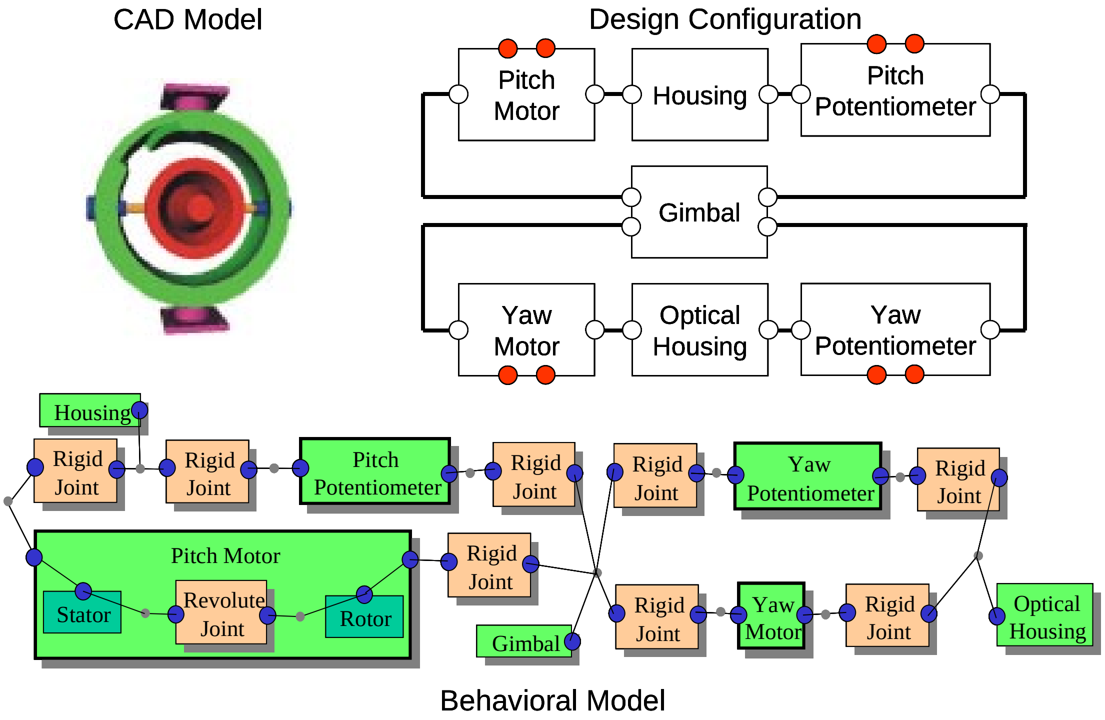


Figure 14: Form and behavior of an incomplete design prototype.

test the choice of controller with the most up-to-date dynamics models. The behavioral models in the different energy domains are combined into a system-level VHDL-AMS model that is evaluated using a commercial solver, as is shown in Figure 15.

9.4 Final Design Verification

For the final design verification, the designers decide to increase the level of detail of the model. The mechanical designer reconfigures the motor models to include nonlinear friction, while the control engineer replaces the analog implementation of the motor controller with a digital version that includes a PWM amplifier. The resulting system model requires significantly

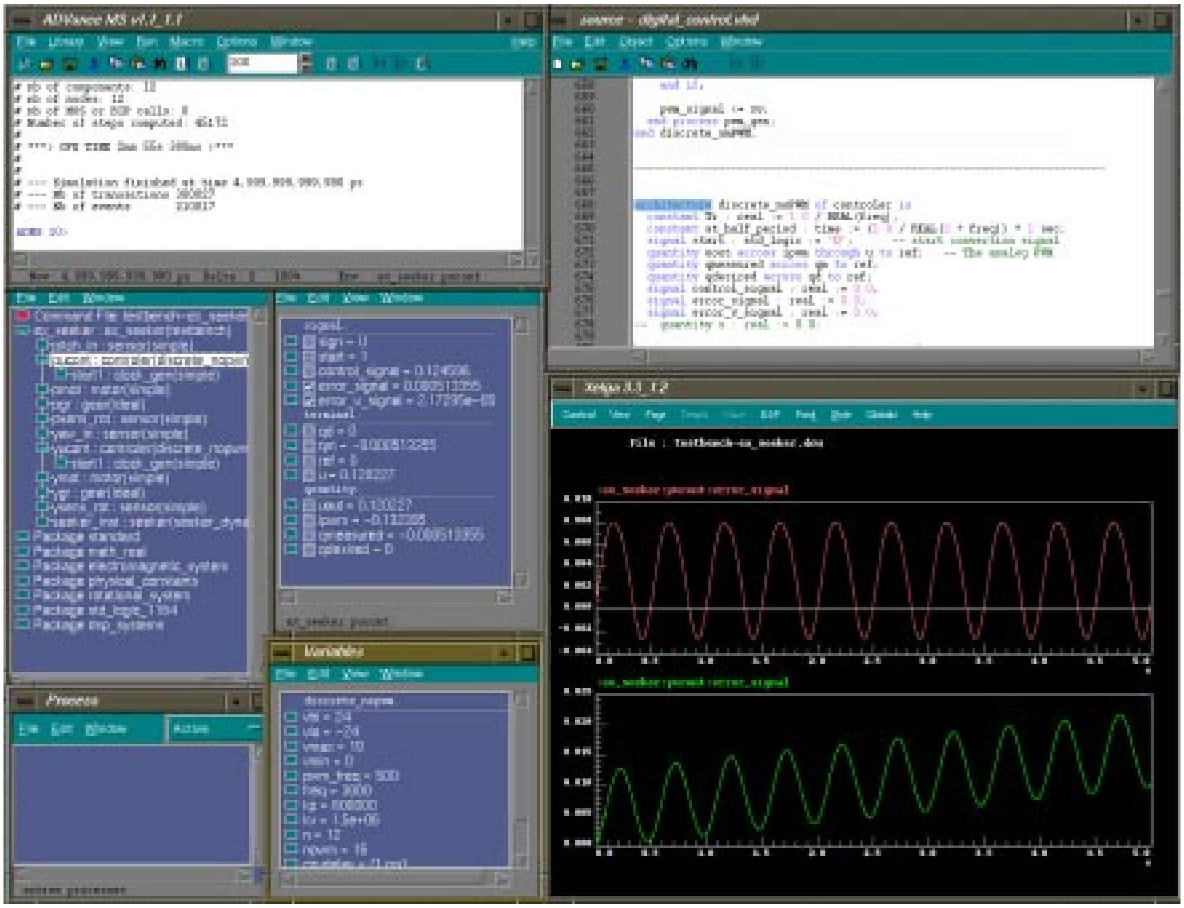


Figure 15: The VHDL-AMS simulation environment, ADVanceMS, by Mentor Graphics. This intermediate analysis shows an increasing position error in the control of the yaw motor for a 2 Hz sinusoidal input signal. The ADVanceMS environment lists the content and directory structure of the VHDL-AMS models, and provides access to all the variables that are defined in the models.

more time to evaluate, but increases the design team's confidence that the final design will perform as desired.

10 Summary and Discussion

To support simulation-based design, we have 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 enable this composition we have developed a modular port-based modeling paradigm that also facilitates the reconfiguration of models. The integration between form and behavior is further enhanced by defining relationships between CAD and behavioral parameters for component families. To extract the parameters of interaction models from the form of interacting components, we have developed procedures that automatically determine the type and parameters of lower pair mechanical interactions.

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 and fixed interfaces. We have successfully applied it to applications in the mechatronics area and have developed a system-level simulation for modular train systems in collaboration with DaimlerChrysler Rail Systems (AdtranzNA). However, to carefully evaluate its expected benefits in terms of component reuse and a faster, less expensive design cycle will require significant further research.

Additional research is also needed to expand the functionality of the framework. With respect to systems modeling, the aspect of *automatically instantiating interaction models*, given a component configuration, requires further investigation. We are currently developing taxonomies of ports and interaction models to address this need. The *selection of an adequate level of detail* for simulation models also requires further expansion of the capabilities of our framework. We currently provide the capability to include models at different levels of detail in reconfigurable models, but have not yet addressed the issue of aiding the user in selecting the most appropriate model for a particular simulation experiment—the model that has adequate accuracy and requires minimum computational resources. Finally, to allow very detailed analyses, *finite-element models* need to be included in our framework. Future research should focus on the interfacing between finite element models and lumped models so that we can includes models of distributed physical phenomena such as mechanical flexure, or complex electromagnetic and thermal behavior in system-level models.

Acknowledgments

We would like to thank Li Han, Werner Juengst, Veichung Liang, Simon Szykman, Ram Sriram, Eswaran Subrahmanian, and Art Westerberg for the insightful discussions, and Peder Andersen and Alex Cunha for their help with the implementation.

We would also like to thank the reviewers for their constructive comments; they greatly improved the accuracy and clarity of the ideas presented in this article.

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, and by the Pennsylvania Infrastructure Technology Alliance. Additional support was provided by DaimlerChrysler Rail Systems (AdtranzNA) and by the Institute for Complex Engineered Systems at Carnegie Mellon University.

References

Anderson, M. (1994). *Object-Oriented Modeling and Simulation of Hybrid Systems*. Ph. D. Thesis. Lund Institute of Technology, Department of Automatic Control. Lund, Sweden.

Andrews, G. C., Richard, M. J. and Anderson, R. J. (1988). “A General Vector-Network Formulation for Dynamic Systems with Kinematic Constraints.” *Mechanisms and Machine Theory*. 23(3):243-256.

Ascher, U. M. and Petzold, L. R. (1998). *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Philadelphia, Pennsylvania, SIAM.

Baldwin, C. Y. and Clark, K. B. (2000). *Design Rules: The Power of Modularity*, MIT Press.

Bettig, B., Summers, J. D. and Shah, J. J. (2000). "Geometric Examplars: A Bridge between CAD and AI." *The Fourth IFIP Working Group 5.2 Workshop on Knowledge Intensive CAD (KIC-4)*. Parma, Italy. 57-71.

Branin, F. H. (1966). "The Algebraic-Topological Basis for Network Analogies and the Vector Calculus." *Symposium on Generalized Networks*. Brooklyn, New York. 453-491.

Bullinger, H.-J., Breining, R. and Bauer, W. (1999). "Virtual Prototyping – State of the Art in Product Design." *26th International Conference on Computers & Industrial Engineering*. Melbourne, Australia. 103-107.

Christen, E., Bakalar, K., Dewey, A. M. and Moser, E. (1999). "Analog and Mixed-Signal Modeling Using the VHDL-AMS Language." *36th ACM/IEEE Design and Automation Conference*. New Orleans. IEEE.

Cutkosky, M. R., Tenenbaum, J. M. and Glicksman, J. (1996). "Madefast: Collaborative Engineering over the Internet." *Communications of the ACM*. 39(9):78-87.

de Vries, T. J. A. and Breunese, A. P. J. (1995). "Structuring Product Models to Facilitate Design Manipulations." *International Conference on Engineering Design*. Prague, Czech Republic. 1430-1436.

Diaz-Calderon, A., Paredis, C. J. J. and Khosla, P. K. (1999). "A Composable Simulation Environment for Mechatronic Systems." *SCS 1999 European Simulation Symposium*. Erlangen, Germany. Society for Computer Simulation.

Diaz-Calderon, A., Paredis, C. J. J. and Khosla, P. K. (2000a). "Automatic Generation of System-Level Dynamic Equations for Mechatronic Systems." *Journal of Computer-Aided Design*. 32(5-6):339-354.

Diaz-Calderon, A., Paredis, C. J. J. and Khosla, P. K. (2000b). "Organization and Selection of Reconfigurable Models." *2000 Winter Simulation Conference*. Orlando, Florida. IEEE.

Diaz-Calderon, A., Paredis, C. J. J. and Khosla, P. K. (2000c). "Reconfigurable Models: A Modeling Paradigm to Support Simulation-Based Design." *2000 Summer Computer Simulation Conference*. Vancouver, Canada. Society for Computer Simulation.

Dynasim AB (1999). *Dymola*. Lund, Sweden. http://www.dynasim.se.

Elmqvist, H. and Mattsson, S. E. (1997). "An Introduction to the Physical Modeling Language Modelica." *European Simulation Symposium*. Passau, Germany. Society for Computer Simulation.

Elmqvist, H., Mattsson, S. E. and Otter, M. (1998). "Modelica: The New Object-Oriented Modeling Language." *The 12th European Simulation Multiconference*. Manchester, UK.

Feldkamp, F., Heinrich, M. and Meyer-Gramann, K. D. (1998). “SyDeR—System Design for Reusability.” *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing.* 12(4):373-382.

Haas, S. and Jasnoch, U. (1994). "Cooperative Working on Virtual Prototypes." *IFIP Workshop on Virtual Prototyping*. Providence, RI. IFIP.

IEEE (1999). *1076.1 Working Group: Analog and Mixed-Signal Extensions for VHDL*, IEEE.

Karnopp, D. C., Margolis, D. L. and Rosenberg, R. C. (1990). *System Dynamics: A Unified Approach*. New York, John Wiley & Sons, Inc.

Keirouz, W., Shooter, S. and Szykman, S. (2000). "A Model for the Flow of Design Information in Openade. Gaithersburg, MD, National Institute for Standards and Technology.

Mattikalli, R., Barraff, D. and Khosla, P. K. (1994). "Finding All Gravitationally Stable Orientations of Assemblies." *IEEE International Conference on Robotics and Automation*. San Diego, CA. 251-257.

Mattsson, S. E., Elmqvist, H. and Otter, M. (1998). “Physical System Modeling with Modelica.” *Control Engineering Practice.* 6:501-510.

McPhee, J. J. (1996). “On the Use of Linear Graph Theory in Multibody System Dynamics.” *Nonlinear Dynamics.* 9:73-90.

Mentor Graphics (2000). *IC-Design Suite*. Wilsonville, OR. http://www.mentor.com/ic_design.

Muegge, B. J. (1996). *Graph-Theoretic Modeling and Simulation of Planar Mechatronic Systems*. MASc. Thesis. University of Waterloo, Systems Design Engineering. Waterloo.

O'Brien, J. and Hodgins, J. (1999). "Graphical Modeling and Animation of Brittle Fracture." *SIGGRAPH*. Los Angeles, CA, USA. ACM Press, NY 137-146.

Pahl, G. and Beitz, W. (1996). *Engineering Design: A Systematic Approach*. London, U.K., Springer-Verlag.

Paynter, H. M. (1961). *Analysis and Design of Engineering Systems*. Cambridge, MA, MIT Press.

Richard, M. J., Bindzi, I. and Gosselin, C. M. (1995). “A Topological Approach to the Dynamic Simulation of Articulated Machinery.” *Journal of Mechanical Design.* 117:199-202.

Roe, P. H. O. n. (1966). *Networks and Systems*. Reading, Massachusetts, Addison-Wesley.

Rosenberg, R. C. and Karnopp, D. C. (1983). *Introduction to Physical System Dynamics*. New York, McGraw-Hill.

Sahlin, P. (1996). "Nmf Handbook. Stockholm, Sweeden, Department of Building Sciences, Division of Building Services, Royal Institute of Technology.

Sedra, A. S. and Smith, K. C. (1997). *Microelectronic Circuits,* Oxford Univ Press.

Shah, J. J. and Mantyla, M. (1995). *Parametric and Feature-Based CAD/CAM: Concepts, Techniques, Applications*. New York, NY, John Wiley & Sons.

Shetty, D. and Kolk, R. (1997). *Mechatronics System Design*, Brooks/Cole Pub Co.

Shooter, S. B., Keirouz, W., Szykman, S. and Fenves, S. J. (2000). "A Model for the Flow of Design Information." *ASME DETC 2000, 12th International Conference on Design Theory and Methodology*. Baltimore, MD. DETC2000/DTM-14550.

Sinha, R., Paredis, C. J. J., Gupta, S. K. and Khosla, P. K. (1998). "Capturing Articulation in Assemblies from Component Geometry." *ASME Design Engineering Technical Conference*. Atlanta, GA. ASME.

Sinha, R., Paredis, C. J. J. and Khosla, P. K. (2000). "Kinematics Support for Design and Simulation of Mechatronic Systems." *The Fourth IFIP Working Group 5.2 Workshop on Knowledge Intensive CAD (KIC-4)*. Parma, Italy. 246-258.

Spell, B. (2000). *Professional Java Programming: Class Design, Threads, Event Handling, Layout Managers, Swing Components, JDBC, XML, Security, Javahelp, JNI, Performance, and Distributed Objects,* Mass Market Paperback.

Strauss, J. C., Augustin, D. C., Fineberg, M. S., Johnson, B. B., Linebarger, R. N. and Sanson, F. J. (1967). “The SCI Continuous System Simulation Language (CSSL).” *Simulation.* 9(6):281-303.

Susca, L., Mandorli, F. and Rizzi, C. (2000). "How to Represent "Intelligent" Components in a Product Model: A Practical Example." *The Fourth IFIP Working Group 5.2 Workshop on Knowledge Intensive CAD (KIC-4)*. Parma, Italy. 197-208.

Swenson, D. V. and Ingraffea, A. R. (1988). “Modelling Mixed-Mode Dynamic Crack Propagation Using Finite Elements: Theory and Applications.” *Computational Mechanics.* 3:187-192.

The Boeing Company (1999). *Easy5: Engineering Analysis System*. http://www.boeing.com/assocproducts/easy5/.

The Mathworks Inc. (1999). *Matlab/Simulink.* http://www.mathworks.com.

Trent, H. M. (1955). “Isomorphisms between Oriented Linear Graphs and Lumped Physical Systems.” *The Journal of the Acoustical Society of America.* 27(3):500-527.

Upton, J., Ed. (1998). *Boeing 777.* Airlinertech Series, Specialty Press.

van Dixhoorn, J. J. (1980). "Bond Graphs and the Challenge of a Unified Modeling Theory of Physical Systems." *Progress in Modeling and Simulation*. Ed.: Cellier. London, Academic Press. 207-245.

W3C (1999). *Extensible Markup Language (XML)*, World Wide Web Consortium. http://www.w3.org/XML.

Zeigler, B. P., Praehofer, H. and Kim, T. G. (2000). *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press.