Behavioral Model Composition in Simulation-Based Design

Rajarishi Sinha[1], *Student Member, IEEE*, Christiaan J.J. Paredis[1,2], *Member, IEEE*, and Pradeep K. Khosla[1,2], *Fellow, IEEE*
[1]Institute for Complex Engineered Systems, Carnegie Mellon University, Pittsburgh, PA 15213, USA
[2]Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA
Email: {rsinha, cjp, pkk}@cs.cmu.edu

Abstract

We present a simulation and design framework for simultaneously designing and modeling electromechanical systems. By instantiating component objects and connecting them to each other via ports, a designer can configure complex systems. This configuration information is then used to automatically generate a corresponding system-level simulation model.

The building block of our framework is the component object. It encapsulates design data and behavioral models and their inter-relationships. Component objects are composed into systems by connecting their ports. However, when converting a system configuration into a corresponding simulation model, the corresponding models for the component objects do not capture the physical phenomena at the component interfaces¾the interactions. To obtain an accurate composition, the interaction dynamics must also be captured in behavioral models.

In this paper, we introduce the concept of an interaction model that captures the dynamics of the interaction. When two ports are connected, there is an intended interaction between the two components. For composition of component objects to work, an interaction model must be introduced between each pair of connected behavioral models. We illustrate these ideas using an example.

1. Introduction and Motivation

The realization of new mechatronic devices is characterized by ever shortening times to market, along with increasing customer demand for improved quality. In this business environment, it is important for a company to be able to design and test the behavior of its products without having to resort to expensive and time-consuming physical prototyping.

A *virtual prototype*, on the other hand, enables the designers to test whether the design specifications are met by performing computer simulations rather than experiments on the physical prototype. 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 easy to create.

The mathematical modeling of virtual prototypes has evolved over time. Many early simulation languages were based on the Continuous System Simulation Language (CSSL) [12]. Models were written as sequential procedures, which implied a fixed mathematical causality. They were implemented as monolithic pieces of software with no separation between model and solver. Subsequently, object-oriented principles have been applied to systems modeling [2, 3, 6], with the result that models are easier to create, reuse and share. Causality assignment is performed automatically, and the solver is independent of the model. However, the product design methodology was not closely coupled with the modeling methodology.

We further the evolution towards a seamless integration of design and simulation by introducing the idea of a *component object* [10]. They contain their configuration information as well as behavioral models and design data. They are connected to other component objects via ports, as shown in Figure 1. In our framework, the virtual prototype is created once all the component objects are interconnected. We have implemented our framework in the Composition In Simulation and Design (COINSIDE) software.

Connecting component objects via their ports is not sufficient to create a complete system-level model. The behavioral models of each component object must also be connected to each other. However, as we will show in subsequent sections, merely making a connection between the modeling ports can result in an incorrect model in many cases.

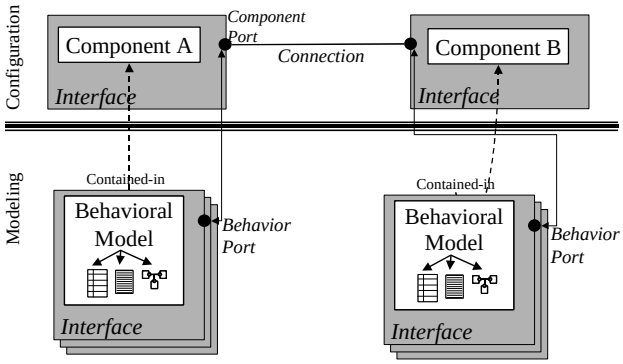

Figure 1. Design as a process of configuration of components and selection of behavioral models for the components and connections.

For a composition operation over component objects to be successful in generating a system-level virtual prototype, component interaction models are required. In this paper, we introduce the idea of *component interaction models* that connect the behavioral models of two interconnected component objects. Interaction models capture the physical dynamics at the interfaces between components. This paper presents a framework that supports the representation, modeling and organization of interaction s between components.

2. Related Work

The related literature can be classified into the following categories: configuration in design and software engineering, and port-based reconfigurable models.

2.1. Configuration in Design and Software Engineering

Our framework is driven by configuration of components that contain analysis models. At the present time, our framework does not incorporate optimization capabilities; we restrict ourselves to analysis of a single configuration at a time. Configuration has been studied in the context of design specification. Feldkamp et al. [5] use port-based composition to describe hierarchical configurations of complex engineering design specifications. Zeigler [14] has developed a DEVS framework for modeling and simulation of hybrid systems. Motta and Zdrahal [9] look at solving parametric design problems using configuration. Gandhi and Robertson We extend these ideas by incorporating analysis models in the configuration model.

Configuration also plays an important role in component-based software engineering. Components are used to describe specific software services, and ports are used to connect components together [1, 4]. Other researchers have used type systems to enforce rules governing software component composition [7]. We use the ideas of software ports to define feature ports for components, and we define type systems that govern ports used in engineering design and simulation.

2.2. Port-Based Reconfigurable Models

The software design methodology of object-oriented programming can be applied to systems modeling as well, with the benefits of simplified model creation and maintenance. An important principle of object-oriented programming is that of information hiding or encapsulation: an object can only be accessed through its public interface, which is independent of the underlying implementation. 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) [3, 13]. The advantage of encapsulation is that a system can be modeled by composing and connecting the interfaces of its sub-systems, independently of the future implementations of these subsystems [13].

In our framework, interaction models can be selected automatically, depending on the ports that are interacting. In addition, multiple interaction models can relate interacting ports, depending upon the desired level of abstraction.

3. Entities In The COINSIDE Framework

In this section, we introduce the design entities in our framework, namely ports, component objects, behavioral models, interaction models and parameters. We describe how interaction models are important in the creation of correct system-level models.

3.1. Ports

A port is a descriptor for a location on the boundary of a component where the component interacts with its environment (Figure 2). In Figure 2, ports 1 and 3 represent point interactions, whereas port 2 represents a distributed interaction that is lumped at the port. The types of interaction range from abstract descriptions of connection semantics, as is the case for ports in the configuration level, to exchange of mass, energy or information, as is the case in behavioral models [10]. There is one port for each separate interaction point, and the type of a port matches the type of the exchange.

There are two types of ports used in our framework: configuration ports and modeling ports. There are relations between a particular configuration port and its corresponding modeling ports. For example, a gear

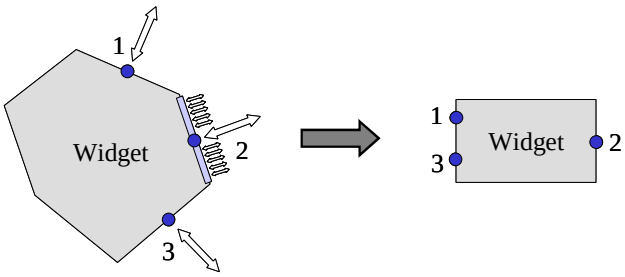

Figure 2. Ports on a component object.

configuration port is related to its corresponding 3D mechanical modeling port.

3.1.1. Configuration Ports. Configuration ports capture connection semantics between a component and its environment. For example, a DC motor component has four ports, two electrical ports, a shaft port and a stator port. The electrical ports correspond to the electrical connectors of the motor, the shaft port to the rotor and the stator port to the stator. A gear component has ports for its teeth and shaft. The gear shaft port is connected to the motor shaft port and is related to a mechanical modeling port that provides a transform for the rotational axis of the gear. The gear teeth port connects to another gear teeth port to form a gear pair, and provides information like the number of teeth and the gear pitch radius via a feature port. Configuration ports can be aggregated to form more complex ports.

Ports can be aggregated at higher levels of abstraction. Aggregate port types are used in component interfaces to describe very high-level connections between components. For example, the connection between a train component and a track component can be thought of as a connection between two aggregate ports that capture all the physical interactions between the two components.

3.1.2. Modeling Ports. The connections between behavioral models are represented by connections between modeling ports. Each connection imposes some constraints on the variables of the modeling port. For a connection between simple energy or mass ports (such as mechanical, electrical, thermal and hydraulic ports), these constraints are the equivalents of the Kirchhoff voltage and current laws in electrical circuits [10, 11]. For signal ports, the constraint equates the value of the signal at each end of the connection.

3.1.3. Port Representation. Ports form the basis of our framework. They are a part of the interfaces for component objects, component interactions, and behavioral models. In previous sections, we defined a port as a discrete point of interaction between a component and its environment. By this definition, a port is the spatial quantum of interaction in our framework.

At a specific, *physical* location on the interface, there can be exactly one port. Therefore *location* becomes the organizing principle of a port. The geometry and material properties at the location become important features of every port. We model the location using a CAD feature, and material properties by a set of defining characteristics such as name and physical properties.

Another important feature is the *intended use* of the port. This feature captures all the intended functions for the port as defined by the designer. The energy and informational domains of the port are also contained within this feature. In addition, any special compatibility constraints on any connections to this port are specified here.

3.2. Component objects

In many design processes, the target device is designed using predefined, modular parts. In such processes, these parts, called components, are selected, configured and assembled in such a way that the design specifications are met.

A component object is a modular design entity with a complete specification describing how it may be connected to other component objects in a configuration. For example, a DC motor component has a shaft to connect it to a drive-train, and bolts that fasten it to a platform. The shaft and the bolts collectively form the ports or interface to this component.

As shown in Figure 3, a component object is instantiated in the design by specifying *instantiation parameters* that describe its specification. Once instantiated, it is connected to other instantiated component objects via its ports. Before simulating the design, the designer selects b*ehavioral models* that describe its physical behavior, and *CAD models* that specify how it may be manufactured and visualized.

A configuration is created when two or more components are connected to each other via their interfaces. A component can itself encapsulate a configuration of components, thus allowing for the hierarchical description of systems (Figure 3). Multiple configurations can represent a particular component, and are bound to the configuration interface for this component. For example, a DC motor can be represented as a single component, or as a configuration of a stator and a rotor component. The candidate configurations are all equivalent specifications of the same component, and the choice of configuration is independent of the choices made for behavioral and CAD models.

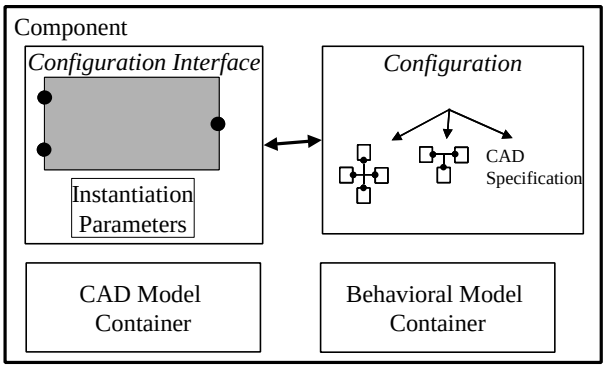

Figure 3. Components may encapsulate configurations of sub-components.

The component is connected to other components via configuration ports. For example, consider the configuration where a DC motor component is connected to a gear component. The DC motor component has ports for the rotor shaft and the stator, and the gear component has ports for the gear teeth and the gear shaft hole. The connection is established by connecting the "rotor shaft" port on the motor to the "gear shaft hole" port on the gear (*ports* are explained in the section on Representation). The configuration ports used in this example are defined in abstract terms, and no information is available about the semantics of the connection that they establish.

The configuration ports are related to modeling ports in the modeling layer. These modeling ports make up the interface of the behavioral models related to the component.

3.3. Behavioral models

Behavioral models capture the mathematical description of the physical and informational behavior of a component. For the scope of this research, we consider these models to consist of either differential-algebraic equations (DAEs) for continuous time phenomena, or discrete event systems specifications (DEVS) [14].

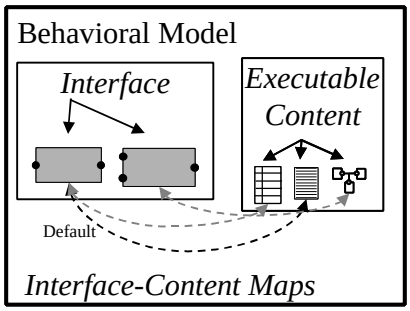

Figure 4. A behavioral model container containing behavioral models. Behavioral models describe the physical or informational behavior of a component.

Behavioral models can also be composed out of other behavioral models through the port-based modeling paradigm [10].

A component object can contain multiple behavioral models with different levels of detail. For example, a DC motor component can contain a family of mechanical behavioral models. One model could only capture the kinematic constraints between the rotor and the stator, while another could include non-linear friction models.

All of these behavioral models are stored in a behavioral model container. The container is separated into three parts: a family of interfaces, a family of implementations of particular models and a set of 2-tuples that enumerate the correspondences between the interfaces and implementations (Figure 4). One of these 2-tuples is the default map, and determines the default interface and implementation for the behavioral model. The implementation is typically a mathematical description of the DAEs and DEVS that make up the behavior of the component. Behavioral models in our framework are represented using the Modelica simulation language [8].

In addition to describing the internal component dynamics, behavioral models also describe the physical phenomena that act between components – the component interactions.

3.4. Interaction models

When a designer composes a system from components, he connects the configuration ports of components. By doing this, the designer explicitly indicates that there is an intended interaction between the connected components. The connections represent physical or information-exchange phenomena that occur at the component interfaces. In Figure 5(a), the resistor, capacitor and AC source have 2 pins each that are connected at A, B and C to form the circuit. There is a one-to-one correspondence between the circuit connections and the connections between the behavioral models. The correspondence is modeled using Kirchhoff's current and voltage network laws, resulting in a correct system-level model.

In the mechanical and other domains, merely connecting the corresponding behavioral models results in a rigid a connection between components. In Figure 5(b), the shaft is connected to the bearing. Applying Kirchhoff's network laws at the connection point A results in the positions $\vec{P}$ being equal and the generalized forces $\vec{F}$ summing to zero. This implies a rigid joint between the shaft and the bearing, which is incorrect.

Introducing an interaction model at A that captures the dynamics at the interface between the shaft and bearing solves this problem. Depending on the type of configuration ports that are connected, candidate interaction models are chosen automatically.

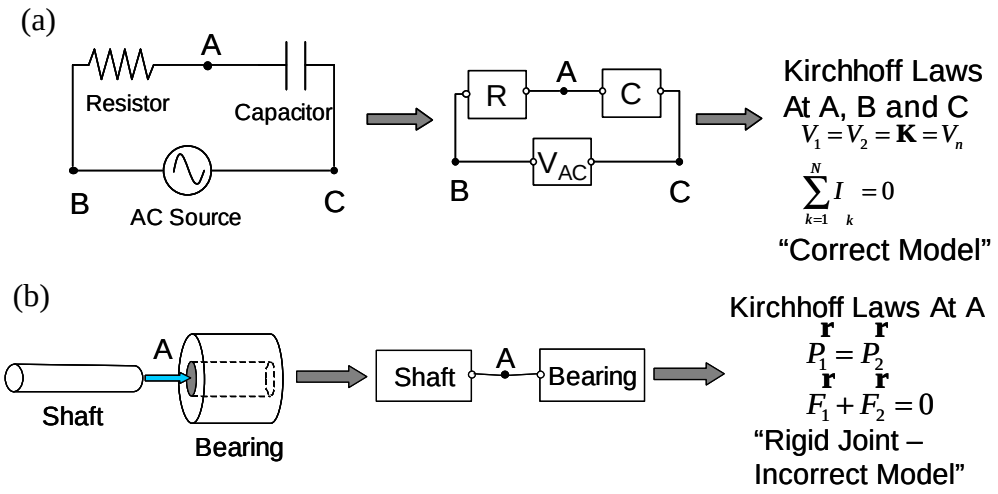

Figure 5. Composition of behavioral models in the electrical and mechanical domains. In (a) the composition occurs via the application of Kirchhoff's Laws. However, in (b), applying Kirchhoff's Laws results in an incorrect rigid joint.

There are a finite number of possible interaction model interfaces that can represent the connection between the configuration ports. In general, if there are m and n candidate behavioral ports for configuration ports 1 and 2 respectively, then the space of behavioral model interfaces representing the component interaction has an upper bound of $m \times n$. The interaction model then becomes a container for this set. For example, consider the two gear components in Figure 6. When the designer makes a connection between the two gear ports in the configuration level, a container interaction model is instantiated in the modeling layer. The container holds all the possible behavioral models that can be used to represent this interaction. Searching a library of interaction models populates the container. In this example, the possible models are two gear interaction models. The parameters of the interaction can be inferred by geometric reasoning on the CAD data in each component [11].

Potentially, a very large number of behavioral models can be present in the interaction model container, and two or more of these models may be closely related. In Figure 6, both the gear interaction models are closely related in that they have the same interface, but slightly different dynamics.

The choice of a particular model from the container depends on the nature of the simulation experiment that is being performed. In Figure 6, there are two gear interaction models in the gear interaction container. One is a simple kinematic gear interaction model with two 3D mechanical ports. Another is a complex gear interaction model with kinematics and frictional dynamics. The first model may be used in preliminary design, when a high-level simulation is needed. The second model would be

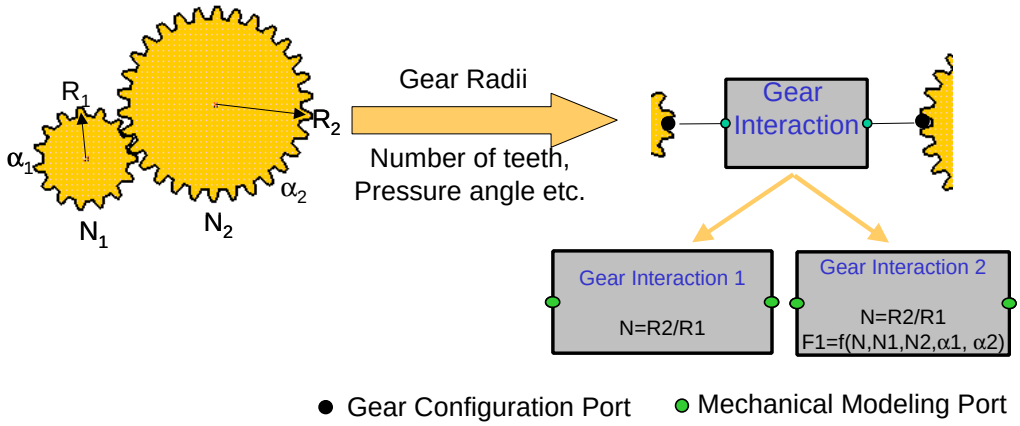

Figure 6. Interaction model as a container for a set of reconfigurable models. In this example, the container lists two possible behavioral models for this interaction.

used later in the design process, when a detailed simulation is performed. Both models are valid choices, and are presented as possible alternatives in the interaction model container for this connection.

This capability allows the designer to encapsulate ports within ports, and create multiple levels of abstraction for the interaction models in the design. At each level, he can work with ports and interaction models whose information richness is sufficient for the current abstraction level. This allows the designer to create and simulate virtual prototypes for complex, hierarchical devices by selecting and connecting components via their interfaces.

3.5. Representation of Parameters

A component from a family is completely specified when all its parameters are provided. For example, consider a family of resistor components that is parameterized by the *resistance* parameter. The value of the resistance parameter must be provided before the resistance component can be instantiated and used in a configuration. We call such parameters *instantiation parameters*.

The instantiation parameters are related to parameters that are used in the behavioral models [11]. Particular CAD features from the CAD specification of a component can be used as parametric input to the behavioral model. For example, a family of gear components can contain a parametric CAD model specification of the gear, and a parametric behavioral model. The CAD model has parametric teeth features (such as number of teeth and pressure angle) that are used to parameterize the behavioral model.

4. Example

To illustrate the concepts developed in the previous sections, we use an example of a complex electromechanical system – a train system. In the interest of brevity, and given that the focus of this work is on interactions, we will focus on the interaction between the train and its track.

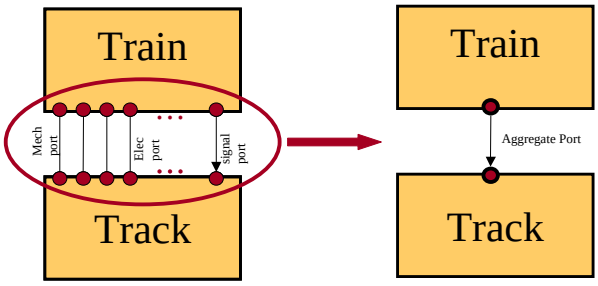

Figure 7. Abstraction of a train-track interaction. Each block represents a component in the configuration layer, and a circle represents a port on the component interface.

In a real-world train-track interaction, there may be hundreds of physical and informational interactions between the train and the track, such as mechanical interactions between the train and the track, electrical power flows, command signals, and sensor signals, as shown in Figure 7.

4.1. Configuration

Our framework supports the configuration of components in the virtual prototype by instantiating and connecting them. So the first step is to select the components that will constitute the virtual prototype. At the highest level of abstraction, we model the train-track interaction with the train component interacting with the track component (Figure 7). We select a train and a track component. In early design, no CAD models are available and the designer provides the parameter values for mass, moment of inertia, etc. The designer connects the train component to the track component via a "train-track interaction" aggregate port.

Once the configuration is complete and all component interfaces are connected, the designer proceeds to the modeling layer to select behavioral models.

4.2. Modeling

In the modeling layer, a high-level train-track interaction model is automatically selected and instantiated, based on the nature of the connected ports. This interaction model is a container for every behavioral model that can be used to describe the interaction between the train and the track (Figure 8).

The choice of behavioral model depends on the design stage and the requirements of the simulation. In this stage of early conceptual design, the particular behavioral model chosen is a simple Newton-Euler mechanical model; the train-track interaction port has only one sub-port (a 3D mechanical port), and the interaction model only considers mechanical translation of the train along the track (bottom right of Figure 8).

When modeling is complete, it is possible to simulate the virtual prototype by translating the behavioral models into Modelica and evaluating them in a commercial Modelica simulator.

4.3. Refinement: Configuration

To obtain a more realistic simulation, the designer decides that further refinement is necessary in the train component. This requires elaborating the train component

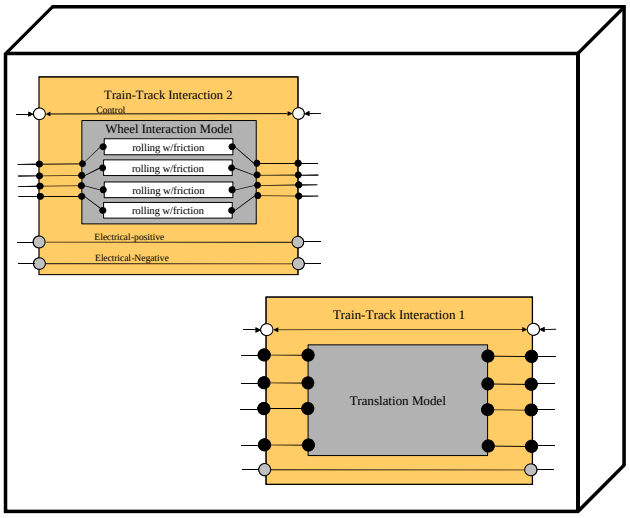

Figure 8. Train-track interaction model container that captures all the candidate interaction models that can model the connection between the train and the track.

in the configuration layer and selecting refined models in the modeling layer.

A new configuration is developed for the train component. The train is instantiated with a CAD model as a parameter. The track component is also instantiated with a CAD model parameter. The train component is now configured as a composition of sub-components: a DC motor, a drive train, a body component, and a control system component (Figure 9).

4.4. Refinement: Modeling

When developing the corresponding behavioral model, models are chosen for each sub-component in the train component, as well as for the track component.

In this stage of detailed conceptual design, the mechanical model of the body of the train is still a simple

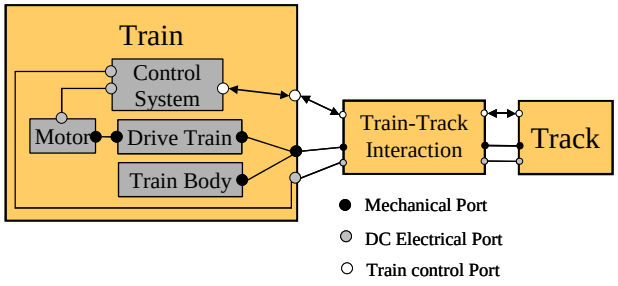

Figure 9. High-level component configuration for a single car train interacting with a track. Each block represents a component, and a circle represents a port on the component interface. Lines represent non-causal connections and arrows represent directed connections.

translational Newton-Euler model, but a DC motor model is added to convert electrical to mechanical energy, and a drive train model increases the torque output of the motor using a simple gear interaction model.

The train-track interaction port is refined to three sub-ports that are connected together on either high-level component. These ports are the control port, the mechanical port, and the DC electrical port. With the train being modeled in CAD as a detailed solid object, a train-track interaction model in the top left of Figure 8 can be automatically derived from the geometry. The parameter extraction engine examines the current CAD model for the train and track components, obtains the material properties and wheel geometry and uses a look-up table to obtain the friction coefficient for a coulomb friction model.

This provides all the necessary information to complete the system model and evaluate it in a Modelica solver.

4.5. Discussion

In this example, we have used abstraction, both in the configuration and in the modeling layer. Abstraction serves an important purpose: to reduce the amount of detail presented to the designer so that he can focus on high-level modeling decisions without dealing with small details.

Our framework supports automatic interaction model selection and instantiation. The automation allows the designer to focus on the more important tasks of configuration and CAD and behavioral model parameter assignment, while accurately capturing the intended interactions between components in the configuration. The automation also maintains consistency between the CAD parameters and behavioral representations.

Separation of the interfaces from the content of models (whether behavior or configuration) has the added advantage of encouraging standardization and reuse of these models in later design projects.

The port-based modeling paradigm imposes constraints on the types of models that can be defined. In particular, all interactions between component objects are limited to discrete locations on their interfaces. This works well when the energy exchange can be accurately modeled as being restricted to the interfaces. Our framework supports this type of interaction model.

Distributed interactions can also be captured within our framework. Instead of a discrete location, a surface on the interface is involved in the interaction. The entire surface is represented by an aggregate configuration port.

Field interactions (e.g. the gravity interaction) involve every physical location within one component object interacting with every physical location within the other component object. In this case, the interface extends to the

entire mass of the component object. When one of the component objects is decomposed into subcomponents, the interaction now involves each of the subcomponents. This is difficult to represent in our framework.

Certain interaction phenomena like mechanical collisions appear and disappear dynamically during the course of a simulation. The current port-based modeling paradigm restricts configurations to be static (i.e. unchanging for the duration of the experiment). In the dynamic case, one could capture every possible connection between the component objects (i.e. create a maximal configuration), and turn on and off only those connections that are active at each time step of the simulation; but this would possibly result in a very large number of interaction models.

5. Summary

We presented a framework where designers can create virtual prototypes of electromechanical systems by configuring components, while simultaneously selecting and assigning CAD parameters and behavioral (simulation) models.

To generate system-level behavioral models from component configurations, the behavioral models of the individual components need to be combined with behavioral models of the interactions between the components. We introduced a mechanism to extract such interaction models automatically based on the matching between component ports. Our framework supports the designer throughout the design process by providing mechanisms for abstraction, automatic model selection and model reuse.

Acknowledgments

This research was funded in part by Bombardier Transportation Systems, by the National Science Foundation under grant # CISE/115/KDI 98 73005, by the Pennsylvania Infrastructure Technology Alliance, and by the Institute for Complex Engineered Systems at Carnegie Mellon University.

References

[1] Allen, R. J. and Garlan, D., "Formalizing architectural connection," presented at 16th International Conference on Software Engineering, Sorrento, Italy, 1994.

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

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

[4] Erdogmus, H., "A formal framework for software architectures," Institute for Information Technology, National Research Council, Ottawa, Canada ERB 1047 / NRC 40136, December 1995.

[5] Feldkamp, F., Heinrich, M., and Meyer-Gramann, K. D., "SyDeR—System design for reusability," *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing*, vol. 12, pp. 373-382, 1998.

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

[7] Lee, E. A. and Xiong, Y., "System-level types for component-based design," University of California at Berkeley, Berkeley, CA ERL/UCB M 00/8, February 29 2000.

[8] Mattsson, S. E., Elmqvist, H., and Otter, M., "Physical system modeling with Modelica," *Control Engineering Practice*, vol. 6, pp. 501-510, 1998.

[9] Motta, E. and Zdrahal, Z., "Parametric Design Problem Solving," presented at 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96), Banff, Alberta, Canada, 1996.

[10] Paredis, C. J. J., Diaz-Calderon, A., Sinha, R., and Khosla, P. K., "Composable Models for Simulation-Based Design," *Engineering with Computers*, vol. 17, pp. 112-128, 2001.

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

[12] Strauss, J. C., Augustin, D. C., Fineberg, M. S., Johnson, B. B., Linebarger, R. N., and Sanson, F. J., "The SCI continuous system simulation language (CSSL)," *Simulation*, vol. 9, pp. 281-303, 1967.

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

[14] Zeigler, B. P., Kim, T. G., Praehofer, H., and Song, H., "DEVS Framework for Modelling, Simulation, Analysis and Design of Hybrid Systems," in *Hybrid II, Lecture Notes in CS*, P. Antsaklis and A. Nerode, Eds. Berlin: Springer-Verlag, 1996, pp. 529-551.