Proceedings of DETC'01
2001 ASME Design Engineering Technical Conferences
September 9-12, 2001, Pittsburgh, PA, USA

DETC2001/CIE-21285

INTERACTION MODELING IN SYSTEMS DESIGN

Rajarishi Sinha, *Student Member, ASME*,
Institute for Complex Engineered Systems,
Carnegie Mellon University,
Pittsburgh, PA 15213, USA.
Email: rsinha@cs.cmu.edu

Christiaan J.J. Paredis, *Member, ASME*,
Institute for Complex Engineered Systems and
Dept. of Electrical and Computer Engineering,
Carnegie Mellon University,
Pittsburgh, PA 15213, USA.
Email: paredis@cmu.edu

Pradeep K. Khosla, *Member, ASME*,
Dept. of Electrical and Computer Engineering and
Institute for Complex Engineered Systems,
Carnegie Mellon University,
Pittsburgh, PA 15213, USA.
Email: pkk@cs.cmu.edu

ABSTRACT

We present a simulation and design framework for simultaneously designing and modeling electromechanical systems. By instantiating component objects in our software and connecting them to each other via ports, the designer configures complex systems.

Interactions are the physical phenomena that occur at the interfaces between connected components. Most of the research in configuration design has focused on modeling components, with very little attention paid to the dynamics of the interaction phenomena. To obtain an accurate virtual prototype, the interaction dynamics must also be captured in behavioral models.

All interactions between components are mediated by ports. We introduce port and interaction model taxonomies, and provide a set-theoretic formalism that defines the algebra of port and interaction models. In addition, the formalism supports automatic instantiation of interaction models given the types of the connected ports, as well as the ability to replace one interaction model for another depending on the requirements of the desired simulation experiment. We illustrate our framework with an example.

KEYWORDS

Simulation-based design, dynamic interactions, CAD, Modelica, port-based modeling, component objects

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.

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. We are working on such a paradigm, *composable simulation and design*, which is based on model composition from system components [1].

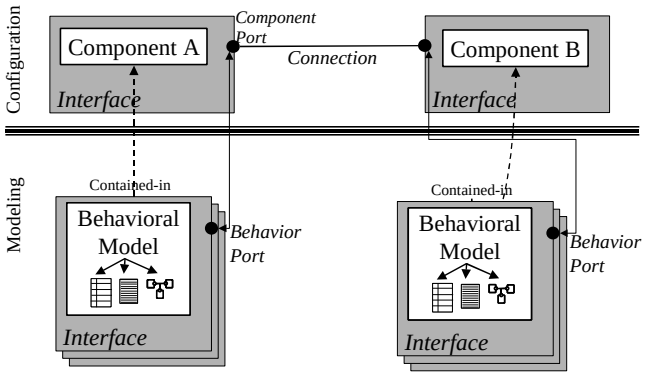

Figure 1. Design as a process of configuration of components and selection of behavioral models for the components and connections.

Many products have a modular architecture that is based on the selection and composition of off-the-shelf components and components reused from older designs. When the new design is created, components are selected and then connected together in a *configuration* (Figure 1).

In addition to the dynamics of the components themselves, physical phenomena occur at the interfaces between components. These interactions must also be modeled. This paper presents a framework that supports the following aspects of interaction modeling:

Model organization. Several models can represent a particular physical phenomenon. These models should be classified and organized so that the designer is not inundated with choices. We provide an interaction model taxonomy and a set-theoretic formalism to organize interaction models.

Reuse through standardized representation. All interactions between the component and its environment occur through the component's interface. Therefore, a library of interaction models can be indexed by their interfaces. Candidate interaction models can be selected by searching the library for models with interfaces compatible with the interfaces of the connected components. We standardize interfaces by utilizing two concepts: *ports* [1] and *instantiation parameters*. A port represents an intended interaction between a component and its environment. An instantiation parameter represents information that must be provided to define a component unambiguously.

Capturing component interaction dynamics. When two components are connected via their interfaces, the connection implies that there is an intended physical interaction between the two components. This interaction must be captured in a behavioral model. The interaction can be as simple as applying Kirchhoff's Laws to an electrical connection. Or it can be more complex such as applying a kinematic joint model to a mechanical connection.

In this paper, the focus is on characterizing, modeling and organizing the interactions between components. With this objective in mind, we model interactions as containers for a family of interfaces. We describe the port-based representation of interfaces and how the interface mediates interactions between components. We apply our insights to generating a system-level virtual prototype for a complex electromechanical device.

RELATED WORK

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

Configuration in Design and Software Engineering

Our framework is driven by configuration of components that contain analysis models, with rules imposed by a set-theoretic formalism. 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. [2] use port-based composition to describe hierarchical configurations of complex engineering design specifications. Motta and Zdrahal [3] look at solving parametric design problems using configuration. Gandhi and Robertson [4] describe a conceptual data model for the configuration of design specifications. We extend these ideas by incorporating analysis models in the configuration model.

Other researchers have looked at the configuration problem from a mathematical and set-theoretic perspective. Some of these results that relate to engineering design and simulation include Wache and Kamp [5] who have recognized that relationships in configuration problems can include "physical laws in a technical domain", part-whole taxonomies, and "procedures in the mechanical engineering domain". Burckert et al. [6] describe the role of formal domain knowledge in configuration problems. Sattler [7] defines a description logic for part-whole relations. We define a set-theoretic formalism that imposes rules on ports, connections between ports and interaction models.

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 [8-10]. Other researchers have used type systems to enforce rules governing software component composition [11]. 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.

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. In this paper, we define a taxonomy and an inheritance hierarchy of ports and interaction models.

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

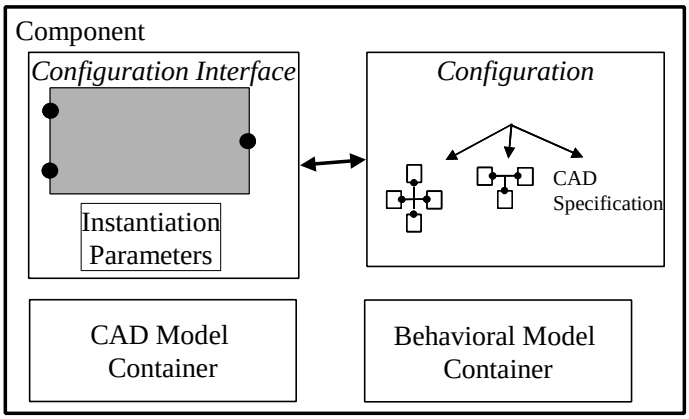

Figure 2. Components may encapsulate configurations of sub-components.

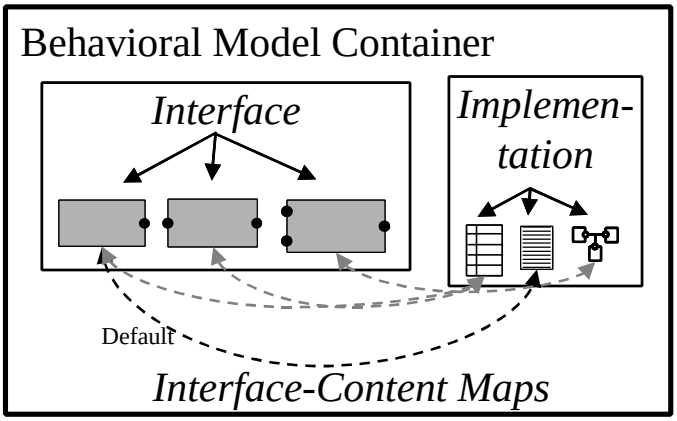

Figure 3. A behavioral model container containing behavioral models. Behavioral models describe the physical or informational behavior of a component.

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) [12-14]. 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, 14].

We build on these ideas by defining a taxonomy and an inheritance hierarchy of ports and interaction models. 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.

COMPONENTS

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 is a modular design entity with a complete specification describing how it may be connected to other components 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 2, a component is instantiated in the design by specifying *instantiation parameters* that describe its specification. Once instantiated, it is connected to other instantiated components 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 2). 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.

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.

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) [15]. Behavioral models can also be composed out of other behavioral models through the port-based modeling paradigm [13].

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 3). 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 [16].

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

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 (Figure 4). The connections represent physical or information-exchange phenomena that occur at the component interfaces.

The high-level component ports in the component interface are related to the ports of the behavioral model interfaces encapsulated in the component (Figure 1). For example, a gear component has a gear port in the configuration level that is related to the gear port of the behavioral model for the component.

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 4. 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

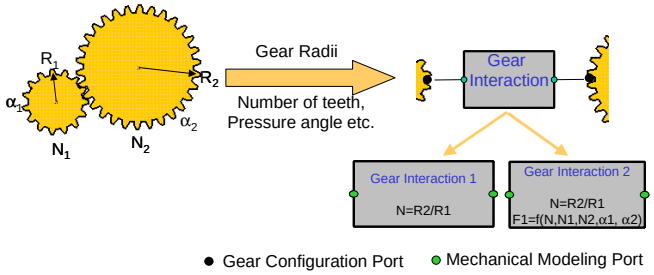

Figure 4. Interaction model as a container for a set of reconfigurable models. In this example, the container lists three possible behavioral models for this interaction.

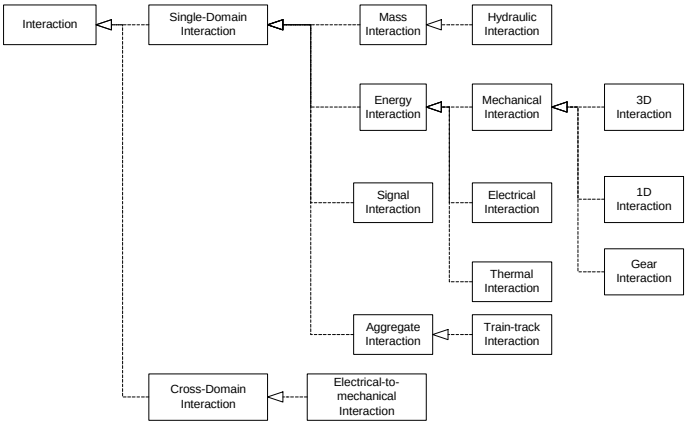

Figure 5. Interaction model taxonomy.

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 [17].

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 4, both the gear interaction models are closely related in that they have the same interface, but slightly different dynamics. To organize and maintain the space of all possible interaction models, and to support evolution of the design, we are working on taxonomies for ports and interaction models, as well as attribute grammars for ports. In this paper, we discuss the taxonomies for interaction models and ports. We have created a taxonomy of interactions (Figure 5). The taxonomy is organized by the number of participating energy and informational domains for the interaction (single and cross-domain interactions). Within each of these categories, models are classified by the physical domains that they represent. For example, the gear interaction model would be classified as a mechanical interaction model.

The choice of a particular model from the container depends on the nature of the simulation experiment that is being performed. In Figure 4, 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 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.

The ports on the models in the container are related via our port formalism and our port type taxonomy, which we present in the next section. A port on the one interaction model can be a supertype of a port on another model. 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.

As discussed in previous sections, the interface to a component is a description of the boundary of the component, where all interactions occur. Any information needed to instantiate a component in a configuration also resides in the interface. We define the interface to be represented by two major parts: ports and parameters.

Representation of Ports

A port is a descriptor for a discrete point on the boundary of a component where the component interacts with its environment. 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 [1]. 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 configuration port is related to its corresponding 3D mechanical modeling port.

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 base port to the stator. A gear component has a gear teeth port and a gear shaft port. 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.

The configuration port hierarchy (Figure 6) has aggregate ports and feature ports as the roots of the taxonomy. Aggregate port types are used in component interfaces to describe very high-level connections between components. We define a feature port as a port that exports a particular geometric feature. For example, the 110V AC plug component could have an interface containing a "110V AC plug pin" feature port that corresponds to the pin form feature in the CAD model, and to the mechanical port in the behavioral model. The port may contain the dimensions of the pin, so that a complementary feature port on an AC socket component can connect to this port.

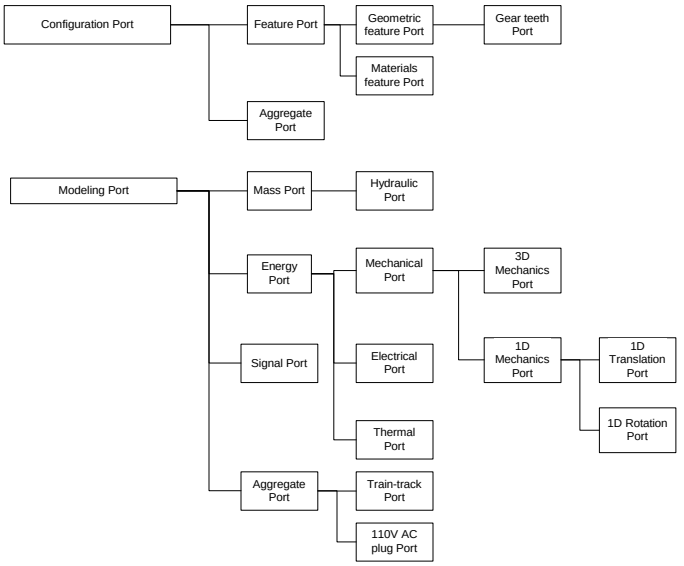

Figure 6. Port type taxonomy.

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 [1, 18]. For signal ports, the constraint equates the value of the signal at each end of the connection.

The roots of the modeling port hierarchy are *mass port*, *energy port*, *signal port*, and *aggregate port* (Figure 6). All other port types are subtypes of these base types. For example, a *mechanical port* type is derived from the energy port type, and a *1D translational port* type is derived from the mechanical port type. The aggregate port type is a container for a finite number of other port types. For example, a *110V AC plug port* type is an aggregate port type containing two *electrical ports* and a *3D mechanical port*. The ports derived from the mass, energy, signal and aggregate port base types are used in the interfaces to behavioral models.

Port Formalism

The designer can extend our taxonomy, provided he follows certain rules in defining new port and interaction model types. These rules are captured in a formalism for a port algebra that is similar to the theory presented by Salustri and Venter [19]. Our formalism is also based on Zermelo-Fraenkel set theory [20, 21]. We represent the port formalism using an XML DTD [22], and the port taxonomy as an XML document based on the DTD. The formalism imposes constraints on the types of ports that can be defined, as well as on the types of ports that can be connected [23]. For example, it requires that when an

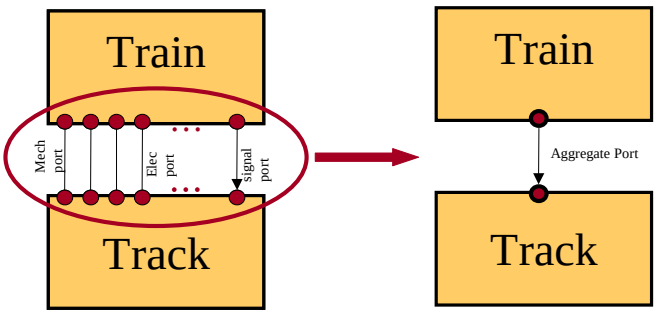

Figure 7. Abstraction of a train-track interaction. Each block represents a component in the configuration layer, and a circle represents a port on the component interface.

interaction model is replaced by another model, the ports still conserve energy. As an illustration, consider the interaction model between an electric train component and a track component. The model interface contains two ports that we call "train-track" modeling ports, one connected to the train component and the other connected to the track component. The train-track ports are defined as an aggregate port containing a 3D mechanical port to capture the mechanical contact, and an electrical port to capture electrical power transfer. This interaction model can be replaced by a model that relates two 3D mechanical ports, but not by a model that relates two control ports. This is because the train-track interaction incorporates an energy exchange that must be preserved in any substituted models. The structure and the magnitude of the interaction between the ports are determined by parameters in the connected components.

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 [18]. 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.

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

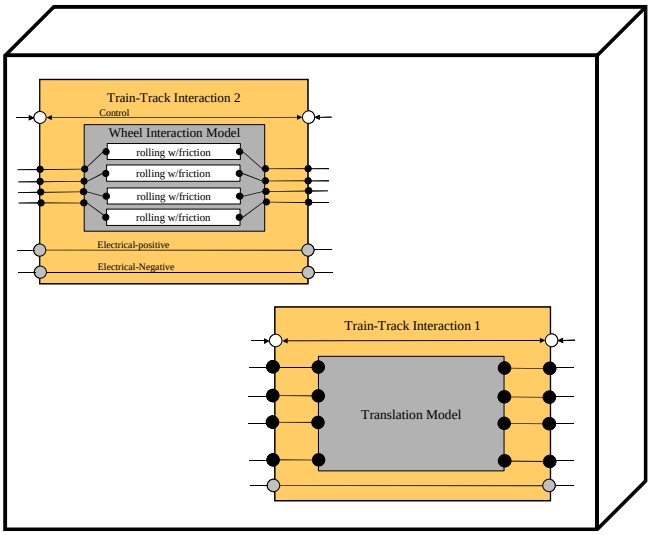

Figure 8. Train-track interaction model container that captures all the candidate interaction models that can model the connection between the train and the track.

interaction between the train and its track. We will use the port and interaction model taxonomies to demonstrate automatic selection, abstraction and design refinement of interaction models.

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.

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.

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 (Figure 9).

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.

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 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 10).

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 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.

Using the taxonomy, 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 Figure 11 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.

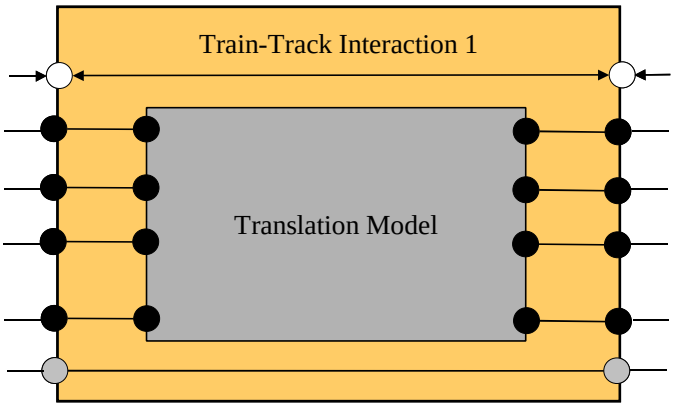

Figure 9. The train-track interaction model at the early design stage. The mechanical interaction has been disaggregated into the individual ports for the four wheels of the train.

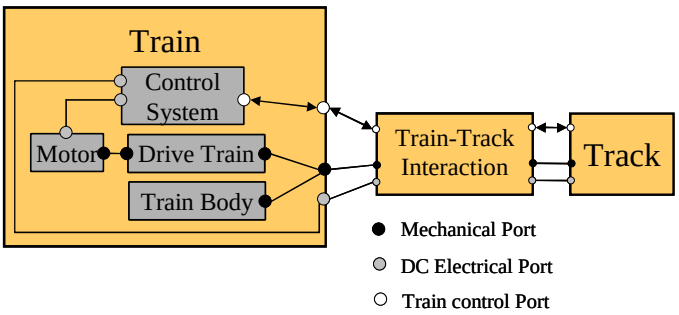

Figure 10. High-level component configuration for a single car train interacting with a track. Each block represents a component, and a circle represents a port on the component interface. Lines represent non-causal connections and arrows represent directed connections.

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 or she 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

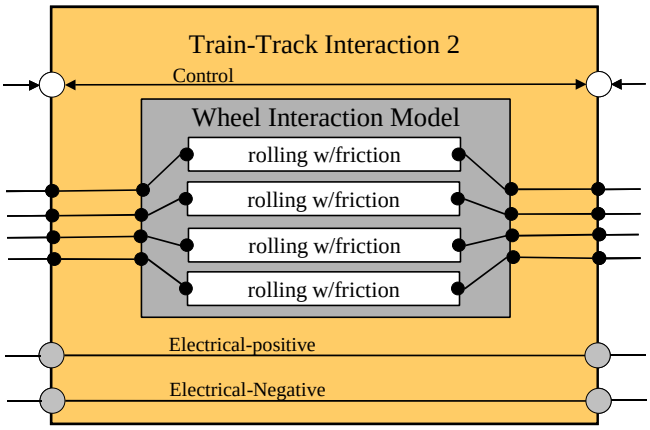

Figure 11. The train-track interaction model in the detailed design stage. The mechanical interaction has been refined to a rolling joint with rolling friction.

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.

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 and a taxonomy of interaction models. Our framework supports the designer throughout the design process by providing mechanisms for abstraction, automatic model selection and model reuse.

In our implementation, we use XML to represent the port and interaction model taxonomies and capture the corresponding dynamic models in Modelica. We illustrate the usefulness of the framework with an example of a complex electromechanical system.

In the future, we plan to extend our framework to incorporate distributed interactions and finite element behavioral models.

ACKNOWLEDGMENTS

We would like to thank the other members of the Composable Simulation team for sharing their insights in simulation-based design and virtual prototyping. We also acknowledge the comments provided by the reviewers; they improved the quality of the manuscript.

This research was funded in part by DaimlerChrysler Rail Systems (Adtranz), by the National Institute of Standards and Technology, 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] Paredis, C. J. J., Diaz-Calderon, A., Sinha, R., and Khosla, P. K., "Composable Models for Simulation-Based Design," *Engineering with Computers*, vol. in press, 2001.

[2] 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.

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

[4] Gandhi, M. and Robertson, E. L., "A Specification-Based Data Model," *11th International Conference on the Entity-Relationship Approach, ER'92*, Karlsruhe, Germany, pp. 194-209, 1992.

[5] Wache, H. and Kamp, G., "Using Description Logic for Configuration Problems," *WRKP'96: Knowledge Representation and Configuration Problems*, DFKI GmbH, Kaiserslautern, 1996.

[6] Burckert, H.-J., Nutt, W., and Seel, C., "The Role of Formal Knowledge Representation in Configuration," *WRKP'96: Knowledge Representation and Configuration Problems*, DFKI GmbH, Kaiserslautern, 1996.

[7] Sattler, U., "Description Logics for the Representation of Aggregated Objects," *14th European Conference on Artificial Intelligence*, Berlin, Germany, 2000.

[8] Magee, J., Dulay, N., Eisenbach, S., and Kramer, J., "Specifying Distributed Software Architectures," *5th European Software Engineering Conference, ESEC'95*, Barcelona, Spain, pp. 137-153, 1995.

[9] Erdogmus, H., "A Formal Framework for Software Architectures," Institute for Information Technology, National Research Council, Ottawa, Canada ERB 1047 / NRC 40136, December 1995.

[10] Allen, R. J. and Garlan, D., "Formalizing Architectural Connection," *16th International Conference on Software Engineering*, Sorrento, Italy, 1994.

[11] 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.

[12] Cellier, F. E., "Object-Oriented Modeling: Means for Dealing with System Complexity," *15th Benelux Meeting on Systems and Control*, Mierlo, Netherlands, 1996.

[13] Diaz-Calderon, A., Paredis, C. J. J., and Khosla, P. K., "Organization and Selection of Reconfigurable Models," *Proceedings of WSC 2000, Winter Simulation Conference*, Orlando, FL, USA, pp. 2 vol.(xl+xxiv+2114), 386-93 vol.1, 2000.

[14] 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.

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

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

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

[18] Sinha, R., Paredis, C. J. J., and Khosla, P. K., "Integration of Mechanical Cad and Behavioral Modeling," *Proceedings 2000 IEEE/ACM International Workshop on Behavioral Modeling and Simulation*, Orlando, FL, USA, pp. viii+119, 31-6, 2000.

[19] Salustri and Venter, "An Axiomatic Theory of Engineering Design Information," *Engineering with Computers*, vol. 8, pp. 197-211, 1992.

[20] Hayden, S. and kennison, J. F., *Zermelo-Fraenkel Set Theory*, 1 ed. Columbus, OH: Charles E. Merrill Publishing Co., 1968.

[21] Spivey, J. M., *The Z Notation: A Reference Manual*, 2nd ed. New York: Prentice Hall International, 1992.

[22] W3C, "Extensible Markup Language (XML),". World Wide Web Consortium, 1999.

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