Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

ORGANIZATION AND SELECTION OF RECONFIGURABLE MODELS

Antonio Diaz-Calderon
Christiaan J. J. Paredis
Pradeep K. Khosla

Department of Electrical and Computer Engineering
Institute for Complex Engineered Systems
Carnegie Mellon University
Pittsburgh, PA 15213, USA

ABSTRACT

This paper introduces the concept of reconfigurable simulation models and describes how these models can be used to support simulation-based design. As in object-oriented programming, a reconfigurable model consists of a separate interface and multiple implementations. An AND-OR tree represents which implementations can be bound to each interface. From the resulting model space, a designer can quickly select the simulation model that is most appropriate for the current design stage. We conclude the paper with an example that illustrates the XML-based implementation of reconfigurable models.

1 INTRODUCTION

Multi-disciplinary simulation is an important tool in the design of mechatronic systems. The demand for less expensive products that can be introduced quickly to respond to market demands requires that these products be designed with minimal prototyping, relying on simulation instead to verify design requirements.

To support such a simulation-based design paradigm, new simulation tools are required. These simulation tools should allow designers and analysts to combine models from different disciplines into integrated system-level models, allow models of sub-systems to evolve throughout the design process (from conceptual design to detailed design), and allow designers and analysts with expertise in different disciplines to collaborate in a open design environment.

To support the evolutionary nature of the design process, it should be easy to modify and refine the models of system components. The reconfigurable models described in this paper allow the designers and analysts to quickly select different models by navigating through an AND-OR tree. Through the domain independent model representation, and XML-based format, these reconfigurable models can also be easily shared among collaborating team members.

We begin the presentation by defining port-based and reconfigurable models. In Section 4, we introduce AND-OR trees to characterize the model space of a component, and use this representation to organize the models into a component library (section 5). Last, we define an XML-based markup language to describe reconfigurable models (section 6).

2 PORT-BASED MODELS

Multi-disciplinary and reconfigurable simulation models build on our *port-based modeling* paradigm (Diaz-Calderon et al. 2000). The port-based modeling paradigm is based on two concepts: *ports* and *connections* (Diaz-Calderon et al. 1998, 1999). As illustrated in Figure 1, ports represent localized points on the boundary of the system where energy exchange between the system and the environment takes place. Energy ports are represented by non-directed lines while signal ports are represented by arrows. A connection between two ports represents the energy exchange between two subsystems.

The port-based modeling paradigm cannot be applied to all systems; it 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

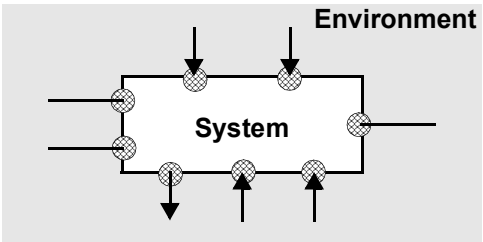

Figure 1: Port-Based Model of an Engineering System

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. Although a finite element model may describe the internal behavior of the beam, the interaction with 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 very detailed models are required, may we have to consider phenomena, such as thermal interactions, that are distributed in nature.

There exist several modeling languages that can describe port-based models. These languages are based on object-oriented principles, and include Dymola (Dynasim AB 1999, Elmqvist et al. 1993), OMOLA (Anderson 1994), NMF (Sahlin 1996), and—more recently—Modelica (Elmqvist 1998) and VHDL-AMS (IEEE 1999). Although these modeling languages can represent port-based objects, they do not permit the model *structure* to be easily modified. Instead, only mechanisms for parameter reconfiguration are provided. Given the evolutionary nature of the design process, it is also desirable to accommodate reconfiguration of the model structure. Therefore, we introduced a modeling paradigm called *reconfigurable models* (Diaz-Calderon et al. 2000).

3 RECONFIGURABLE MODELS

A reconfigurable model is a system representation based on two concepts: *interface* and *implementation*. The interface of a system describes the interaction through a set of ports, while the implementation describes a system's internal behavior. Together they define a complete model of a system. As is illustrated in Figure 2, it is possible to assign different implementations to the same system interface, thereby achieving reconfigurability of models. Our reconfigurable models are conceptually similar to the polymorphic models defined by de Vries (de Vries and Breunese 1995), but have a different instantiation mechanism. Additionally, de Vries represents models by bond graphs which limits their applicability to lumped parameter systems.

A reconfigurable component model provides a mechanism to describe changes in both structure and parameter values. Reconfiguration is based on *instantiation* and *composition.* The principle of instantiation refers to the binding of an interface to an implementation. It is represented by pairs of the form $\langle \Phi | \phi \rangle$ where Φ is an interface and ϕ is an implementation bound to Φ. The principle of composition refers to the description of the behavior of a component in terms of interfaces and interactions of subcomponents.

By binding all possible implementations to each of the interfaces of a reconfigurable model, we obtain the *model space* of a component. To facilitate the selection of a particular model from this space, we organize it as an AND-OR tree. In addition, we have developed a component taxonomy to organize the reconfigurable models in a component library.

4 COMPONENT STRUCTURE

In this section, we present a component structure based on an AND-OR tree that captures the entire model space for a component. The component structure is similar to the *system entity structure* (SES) developed by Zeigler (Kim et al. 1990). The difference is that the SES captures system architecture alternatives, while the component structure describes modeling alternatives for a single component in the system. To capture system architecture alternatives, the SES defines a set of labels that specify both coupling information and selection constraints imposed on the elements of the system.

In an AND-OR tree representation of the modeling space, each implementation of an interface generates an AND arc as illustrated in Figure 3. The *degree* of an AND arc is defined as the number of successor nodes. AND arcs of $degree = 1$ represent primitive implementations. AND arcs of $degree > 1$ represent compound implementations. Based on the principle of composition, an AND arc points to all the interfaces that constitute the composed model. The principle of instantiation is described by an OR arc pointing to all valid implementations for a given interface.

For example, the AND-OR tree shown in Figure 3 depicts part of the structure of a DC motor. The top-level interface can be bound to three different implementations.

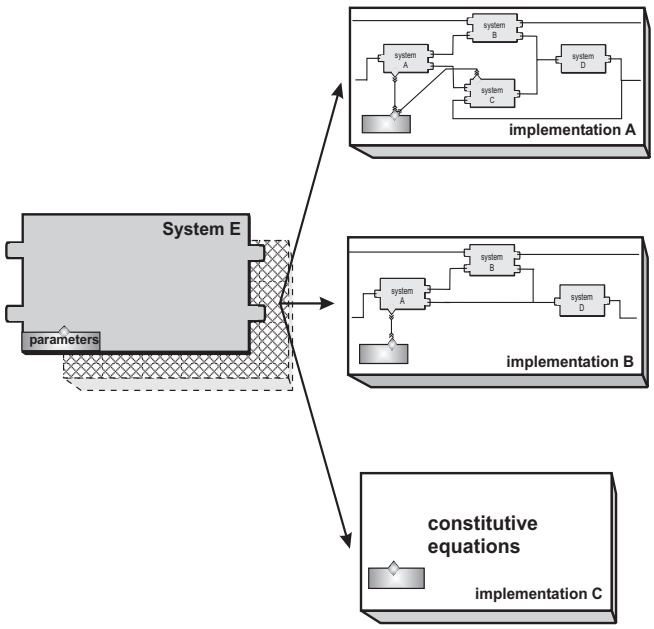

Figure 2: A Reconfigurable System Model

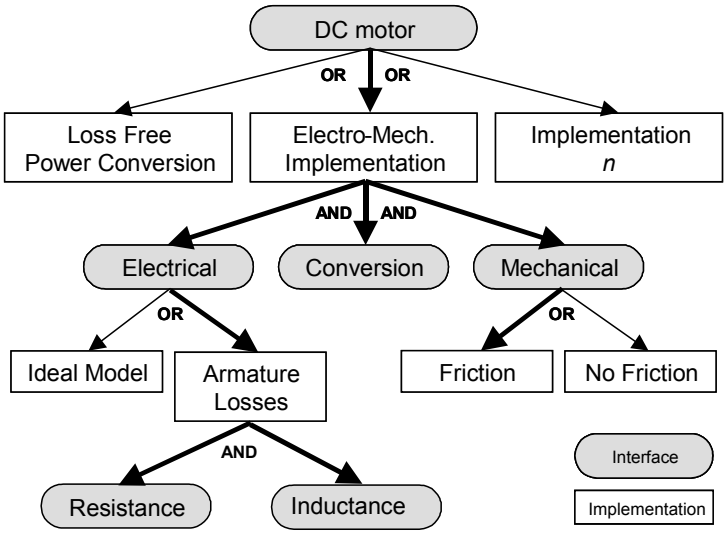

Figure 3: Component Model Structure Based on an AND-OR Tree

At the second level, the AND arc of *degree* = 3 indicates that the electro-mechanical implementation is composed of the interfaces *electrical, conversion* and *mechanical*. Each interface, in turn, generates an AND-OR tree that expands the possibilities in the selection of their respective implementations. For instance, the implementation "armature losses" of the interface *electrical*, spans an AND arc of *degree* = 2 with two interfaces *resistor* and *inductance*.

A particular model for a component is described by a sub-graph (an *induced* tree) of the AND-OR tree. The induced sub-graph on the component model structure has no remaining OR arcs and is defined by the bindings $\langle \Phi_i | \phi_k \rangle$ for $i = 1 \ldots N, k \in 1 \ldots n_i$ where N is the number of interfaces and n_i is the number of implementations for interface Φ_i. An example of an induced tree is indicated by bold arcs in Figure 3.

To fully define a concrete component, it is also necessary to define the parameter values of the model. This combined process of finding an induced sub-graph in the component structure together with the assignment of fixed parameter values for the component is called realization. Without assigning specific parameter values, a model represents a family of components, for instance, the family of all DC motors. To represent the hierarchical relationships between reconfigurable models in model families, we have organized the models in a component library.

5 COMPONENT LIBRARY

The component library presents to the user the set of available reconfigurable models in an organized fashion. It contains two kinds of models: *system component models* and *component interaction models*. System components include physical artifacts such as motors, gears, resistors, or microcontrollers. Component interaction models capture the dynamics of the interaction between two system components, but are not directly associated with any physical structure. For example, consider the model of a revolute joint illustrated in Figure 4. Although a joint appears in almost any mechanism, it does not have any physical structure. Instead it represents the kinematics and dynamics of the interaction between two components, and its parameter values depend on the physical structure of the interacting components.

Figure 4: Rigid Body Interaction

As is shown in Figure 5, component models in the library are organized in a type hierarchy that is represented by a directed acyclic graph. The vertices in the graph point towards more specific instances of components. The family of DC-motors, for example, is classified under the path *SystemComponents→Transducers→Motors→DC-motors*. Furthermore, a component (or subgraph of the library) may appear in multiple locations; for instance, the same DC-motor may be classified under *SystemComponent→Electro-Mechanical*. Although one can define different taxonomies for different applications, we have defined a taxonomy based on the energy domains and functions of the components.

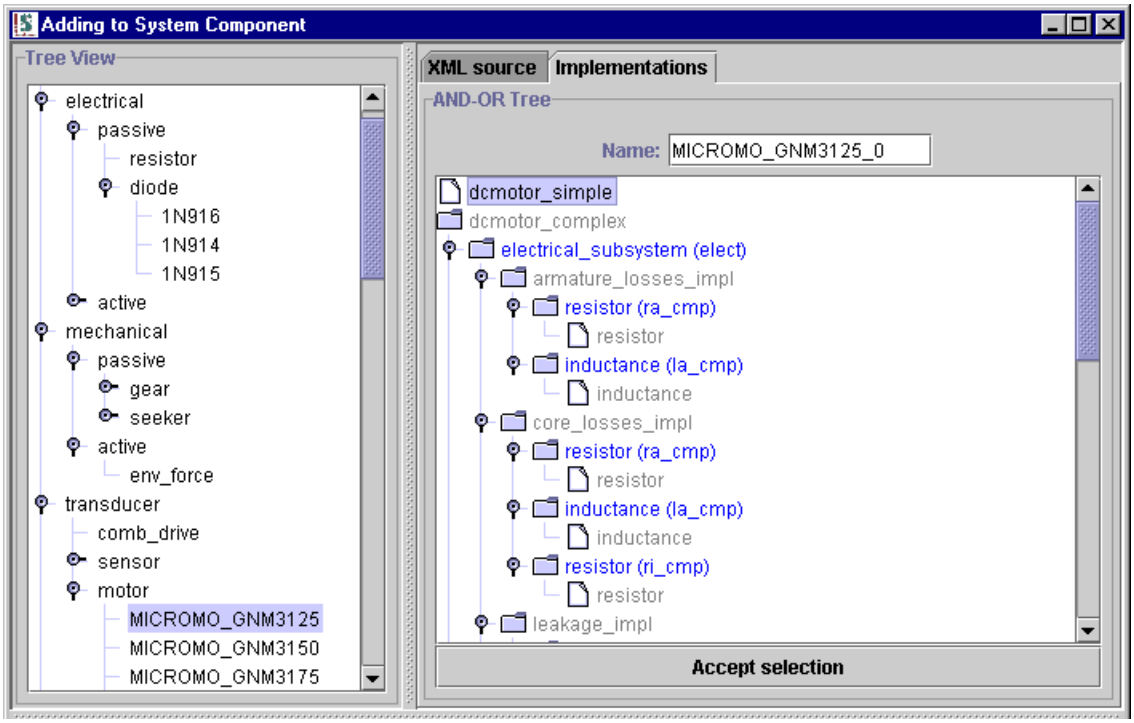

Figure 5: Component Library Browser

The component library may also represent the relation between a family of models and a particular instance of that family. A family of components is defined by an *abstract* model. Such a model has a fully defined reconfigurable structure, but its parameters have not yet been fixed. A *concrete* model, on the other hand, has a complete set of parameter values assigned. Concrete models may be derived from abstract models by assigning specific values to abstract model's parameters, or may be defined independently as a model without parameters. An example of a family↔instance relation in the component library is the MICROMO_GNM3125 motor model, which is derived from the abstract motor model by specifying the appropriate parameter values.

The organization of component models, as described above, supports the simulation-based design process. As is illustrated in Figure 6, one can think of engineering design as the process that transforms functional requirements into a physical structure or form, with a behavior that satisfies the desired function. Typically, high-level functional requirements are first hierarchically decomposed into sub-functions. These sub-functions are then mapped to physical structures that are ultimately re-composed into a complete system. Within the context of simulation-based design, simulation models are used to evaluate different design alternatives and to support the decisions the designer has to make. Since the designer is gradually adding detail to the design prototype, the corresponding simulation models have to follow this same evolution. Our component library and reconfigurable models aid the designer in this process.

For example, at a particular stage of the design process, the designer decides that the function of transforming electrical energy into rotational mechanical energy is best performed by a DC-motor. He can then replace the model for a generic electro-mechanical transducer with one of its children in the component taxonomy, namely, the DC-motor model. At this time, the DC-motor model still represents the family of all possible DC-motors, because its parameters have not yet been fixed. This allows the designer to evaluate different motor alternatives by performing simulations with different parameter values. Once and appropriate set of parameters has been determined, the designer can refine his choice and select a particular DC-motor from among the

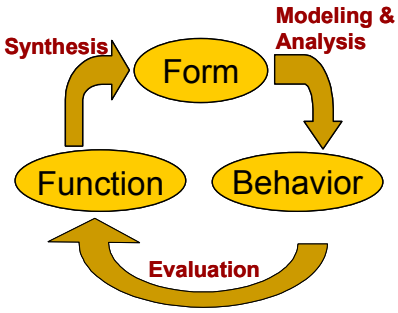

Figure 6: The Simulation-based Design Process

children of the generic DC-motor model in the component library.

In conjunction with the selection of a particular component from the component library, the designer may also select a particular model from the component's model space. For instance, after selecting a particular DC-motor type, the designer decides to replace the simple default model implementation with a detailed implementation that includes models for armature losses and mechanical friction.

6 COMPONENT MODELING MARKUP LANGUAGE

This section introduces the language for describing reconfigurable component models. The language is based on XML (extensible markup language) (W3C 1999) and captures the component model structure based on the AND-OR tree representation. Although most simulation languages do not support model reconfiguration, a particular model selection (induced tree) can be translated into a variety of languages: VHDL-AMS (IEEE 1999), modelica (Elmqvist 1998), ASCEND (Piela et al. 1991), OMOLA (Anderson 1994), or NMF (Sahlin 1996). Within our software framework, the XML description is translated into VHDL-AMS and then simulated with Advanced MS from Mentor Graphics. The design of our model description language, has the following characteristics:

- **Multi-disciplinary**: the language captures the interactions between components in multiple energy domains.
- **Declarative**: the language expresses the laws of physics without assigning causality.
- **Meta-knowledge**: the language represents knowledge that is implicit in the constitutive equations (such as assumptions and approximations).

6.1 Why XML?

XML provides a clear document structure and a context-free vocabulary. We further take advantage of XML's hierarchical document structures to capture the hierarchical nature of the component models. In addition XML provides the following benefits:

- **Document sharing**: designers can use standard XML tools to view and edit models.
- **Component search**: designers can use search tools to locate component models based on desired characteristics.
- **Expressiveness**: a rich internal structure and a rich vocabulary makes model knowledge clear.
- **Reuse**: a consistent document structure makes it easier to reuse document content and apply it to different problem domains.

6.2 The Markup Language

Component models are organized into a *document*, the internal structure of which captures the hierarchical structure of component models. All aspects of the component models (i.e., modeling constructs) are described with a rich vocabulary that translates into XML tags. A *document type definition* (DTD) describes the internal structure of a document and defines the symbols in the vocabulary. The use of this DTD ensures that the models will be *valid* (i.e., the document contains all the parts required by the DTD but no extraneous parts) and *well formed* (i.e., the document is syntactically correct).

The markup language defines the two basic modeling entities, interface and implementation, as the core of the internal document structure. Other tags represent the constitutive equations, subcomponents, interactions between components, and meta-knowledge. Our markup language describes both abstract and concrete system models.

The interface of the system includes required tags for ports and parameters. For example, the interface declaration for the DC motor in Figure 3 would be as follows:

```
interface DCmotor
  parameters
    ktau: real = 1.0e-14;
    km: real = 1.0;
    Ra: real = 1.0;
    La: real = 1.0e-3;
    Jm: real = 1.0e-14;
    Bm: real = 1.0e-5;
  ports
    pos, neg: electrical;
    rotor, reference: rotational;
end DCmotor;
```

Additional optional elements are used to introduce declarations that are common to all implementations of the interface, such as conditional statements that check the validity of parameters, and meta-knowledge statements that provide semantic information about the different implementations associated with the interface.

An implementation of an interface is described by a *declaration section* and a *statement section*. The declaration section defines quantities (or subcomponents in the case of a compound component) that are local to the body of the implementation. The statement section defines the behavior of the component with either a set of constitutive equations or a set of connection statements.

Compound components are described by the tags *component* and *connections*. Components declare the instances

used in the model, and connections define the interactions between subcomponents. A component declaration describes the induced tree in the AND-OR tree of the component. It captures hierarchically the parameter propagation and the interface-implementation bindings.

For the DC motor illustrated in Figure 3, the implementation composed of the three subsystems—electrical, conversion, and mechanical—is defined as:

```
implementation dcmotor-cmp implements DCmotor
  declarations
    elect-subsystem elect(Ra=10,La=0.1)
      bound-to armature-l-impl
        resistor ra(r=Ra) bound-to resistor
        inductance la(l=La) bound-to inductance;
    conv-subsystem conv(Km=km, Kt=ktau)
      bound-to conv-impl;
    mech-subsystem mech(Bm=1.0e-5)
      bound-to friction-imp;
  statements
    connections;
end dcmotor-cmp;
```

In this implementation, the paths *DCmotor-Electrical-[Resistor, Inductance], DCmotor-Conversion,* and *DCmotor-Mechanical-[Friction]*, shown in Figure 3, provide the subtree selected for the DC motor. The electrical system of the DC motor includes armature losses in the electrical subsystem and friction in the mechanical subsystem.

Binding the electrical subsystem to the implementation *armature-losses* requires also binding implementations for the components *ra* and *la*. These bindings are recursively specified in the declaration of the component *elect*. For example, component *ra* with interface *resistor* is bound to implementation *resistor*. The implementation *resistor* is defined by a set of constitutive equations and does not require bindings for any sub-components.

7 FUTURE DIRECTIONS

This research fits within a larger project, called "The Composable Simulation Project" (Composable Simulation, 2000). The goal of the project is to support the simulation-based design process with a new simulation framework. As indicated in Section 5, reconfigurable models and component libraries make a step towards this goal. At the same time, this research is only a starting point from which additional research efforts will emanate. Following is a list of a few future directions.

Reconfigurable models and component libraries support model evolution through model selection and component selection. However, both selection processes are performed completely by the user. In the future, we plan to automate these processes at least partially. For component selection, this means helping the user to decide which component has a behavior that best matches a desired function; for model selection, it requires deciding which model is most appropriate (adequate accuracy with minimal computational requirements) for a given experimental frame.

Solving these selection problems requires explicit knowledge about the semantics of the models. This information can be stored as meta-knowledge tags in our XML format. Semantic knowledge is also important for human users—a good designer will not use a model without understanding which physical phenomena it includes and which approximation have been made.

Another related research issue that needs to be addressed in the future is that of *model creation* and *maintenance.* Reconfigurable models contain a lot of information and require a lot of effort to create. These models are only practical if much of this effort can be amortized over a large number of models. One way to achieve this is by grouping models into parametrized model families. A single reconfigurable model can then be used to represent all the instances of the family. A second approach that we are exploring is the automatic generation of models by extracting model parameters from the form description, i.e., CAD data (Sinha et al. 2000). One can think of this approach as models that include a description of both the form and the behavior of a component; the parameters of the behavioral model are functions of the features and parameters of the form.

This brings us to the ultimate goal of the composable simulation project, namely, to interleave the design and analysis processes so that an up-to-date simulation model is automatically generated as the designer defines the design prototype within a CAD environment. The development of reconfigurable models and their organization in component libraries is a first step towards that goal.

8 CONCLUSIONS

In this paper, we introduced the concept of reconfigurable models and their organization into a component library. An AND-OR tree is used to capture succinctly all possible model alternatives (the model space) of a reconfigurable component model. These reconfigurable models can be logically organized into a library of components that supports the simulation-based design process through both component selection and model selection. Because reconfiguration of simulation models is rarely supported in current simulation languages, we have developed a neutral XML format to represent reconfigurable models. From this neutral format, simulation models in a specific language can be derived once a particular simulation model has been selected. Our current implementation uses VHDL-AMS as the target language.

ACKNOWLEDGMENTS

This research was funded in part by DARPA under contract ONR #N00014-96-1-0854, by the National Institute of Standards and Technology, by the NSF under grants #CISE/IIS/KDI 9873005 and #EIA-9729827, by the Pennsylvania Infrastructure Technology Alliance, by Adtranz, and by the Institute for Complex Engineered Systems at Carnegie Mellon University.

APPENDIX

An XML representation of a DC motor model.

```
<interface ident="DCmotor">
  <generics>
    <parameter semantics="torque_constant"
      default="1.0e-14" nature-type="real"
      ident="tau"/>
    <parameter semantics="motor_constant"
      default="1.0" nature-type="real"
      ident="km"/>
    <parameter semantics="armature_resistance"
      default="1.0" nature-type="real"
      ident="Ra"/>
    <parameter semantics="armature_inductance"
      default="1.0e-3"
      nature-type="real"
      ident="La"/>
    <parameter semantics="rotor_inertia"
      default="1.0e-14" nature-type="real"
      ident="Jm"/>
    <parameter semantics=”friction”
      default=”1.0e-5” nature-type=”real”
      ident=”Bm”/>
  </generics>
  <boundary>
    <terminal nature-type="electrical"
      name="pos"/>
    <terminal nature-type="electrical"
      name="neg"/>
    <terminal nature-type=”rotational”
      name=”rotor”/>
    <terminal nature-type=”rotational”
      name=”reference”/>
  </boundary>
</interface>

<implementation compound="true"
  of-interface="DCmotor" ident="dcmotor-cmp">
  <component interface-name="elect-subsystem"
    name="elect">
    <parameter-binding actual-part="10"
      formal-part="Ra"/>
    <parameter-binding actual-part="0.1"
      formal-part="La"/>
    <parameter-binding actual-part="10"
      formal-part="km"/>
    <bound-implementation
      implementation-name="armature-l-impl">
      <component interface-name="resistor"
        name="ra">
        <bound-implementation
          implementation-name="resistor"/>
      </component>
      <component interface-name="inductance"
        name="la">
        <bound-implementation
          implementation-name="inductance"/>
      </component>
    </bound-implementation>
  </component>
  <component interface-name="conv-subsystem"
    name="conv">
    <parameter-binding actual-part="32.0e-3"
      formal-part="tau"/>
      <bound-implementation
        implementation-name="conv-impl"/>
  </component>
  <component interface-name="mech-subsystem"
    name="mech">
    <parameter-binding actual-part="1.0e-5"
      formal-part="Bm"/>
      <bound-implementation
        implementation-name="friction-impl"/>
  </component>
  <concurrent-statements>
    <connect terminal-B="neg"
      terminal-A="elect.neg"/>
    <connect terminal-B="rotor"
      terminal-A="mech.load"/>
    <connect terminal-B="reference"
      terminal-A="mech.ref"/>
    <connect terminal-B="conv.elect"
      terminal-A="elect.conv"/>
    <connect terminal-B="conv.mech"
      terminal-A="mech.conv"/>
    <connect terminal-B="pos"
      terminal-A="elect.pos"/>
  </concurrent-statements>
</implementation>
```

REFERENCES

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

Composable Simulation. 2000. The composable simulation project. Carnegie Mellon University, Institute for Complex Engineered Systems, Pittsburgh, PA 15213.

de Vries T. J. A. and A. P. J. Breunese. 1995. Structuring product models to facilitate design manipulations. In *Proceedings of the International Conference on Engineering Design ICED '95*. Prague, Czech Republic.

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

Diaz-Calderon A., C.J.J. Paredis, and P. K. Khosla. 1999. A composable simulation environment for mechatronic systems. In *Proceedings of SCS 1999 European Simulation Symposium*. Erlangen, Germany.

Diaz-Calderon A., C.J.J. Paredis, and P. K. Khosla. 2000. Reconfigurable models: A modeling paradigm to support simulation-based design. In *Proceedings of SCS 2000 Summer Computer Simulation Conference*. Vancouver, British Columbia, Canada.

Dynasim AB. 2000. Dymola. Lund, Sweden.

Elmqvist H., F.E. Cellier, and M. Otter. 1993. Object-oriented modeling of hybrid systems. In *Proceedings of European Simulation Symposium*. Delft, The Netherlands.

Elmqvist H., S.E. Mattsson, and M. Otter. 1998. Modelica: The new object-oriented modeling language. In *Proceedings of The 12th European Simulation Multiconference*. Manchester, UK.

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

Kim T. G., C. Lee, B. P. Zeigler, and E. R. Christensen. 1990. System entity structure and model base management. *IEEE Transactions System Man and Cybernetics*, 20: 1013-1024.

Piela P.C., T.G. Epperly, K. M. Westerberg, and A. W. Westerberg. 1991. ASCEND: An object oriented computer environment for modeling and analysis. 1- The modeling language. *Comput. Chem Engng*, 15(1): 53-72.

Sinha R., C.J.J. Paredis, and P.K. Khosla. 2000. Kinematics Support for Design and Simulation of Mechatronic Systems. In *Proceedings of the 4th IFIP Working Group 5.2 Workshop on Knowledge Intensive CAD*. Parma, Italy.

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

World Wide Web Consortium. 1999. Extensible Markup Language (XML).

AUTHOR BIOGRAPHIES

ANTONIO DIAZ-CALDERON received the B.S.E.E and the M.S. in Computer Science from Monterrey Institute of Technology (Mexico) in 1987 and 1990 respectively, and the M.S. in Civil Engineering and Ph.D. in Electrical and Computer Engineering from Carnegie Mellon University in 1993 and 2000 respectively. His Ph.D. research focused on composable simulation to support simulation-based design of mechatronic systems. Dr. Diaz-Calderon is a Post-Doctoral Fellow at the National Robotics Engineering Consortium, The Robotics Institute at Carnegie Mellon University. His research activities include vehicle dynamics modeling and simulation, and the development of a lift truck rollover prediction system to reliably indicate proximity to unsafe states of motion. His email and web addresses are <adiaz@cmu.edu> and <www.cs.cmu.edu/~adiaz>.

CHRISTIAAN J.J. PAREDIS received the M.S. in Mechanical Engineering from the Catholic University of Leuven (Belgium) in 1988, and the M.S. and Ph.D. in Electrical and Computer Engineering from Carnegie Mellon University in 1990 and 1996 respectively. His Ph.D. research focussed on task-based design of modular systems, using agent-based evolutionary algorithms. He also developed a composable manipulator (the Reconfigurable Modular Manipulator System) that can be rapidly reconfigured for a different tasks. Dr. Paredis is currently research faculty at the Institute for Complex Engineered Systems at Carnegie Mellon University. His research activities include modeling and simulation of complex electro-mechanical systems, and the development of small distributed robotic systems. Additional information about his research can be found at <www.cs.cmu.edu/~paredis> and <paredis@cmu.edu>.

PRADEEP K. KHOSLA received B. Tech from IIT (Kharagpur, India), and both MS (1984) and Ph.D. (1986) degrees from Carnegie-Mellon University. He served as Assistant Professor of ECE and Robotics (1986-90), Associate Professor (1990-94), and Professor (1994 -), Founding Director (1/97-6/99) of Institute for Complex Engineered Systems (which includes the former Engineering Design Research Center - an NSF ERC). He is currently the Philip and Marsha Dowd Professor of Engineering and Robotics, and Head of Electrical and Computer Engineering Department at Carnegie Mellon. From January 1994 to August 1996 he was on leave from Carnegie Mellon and served as a DARPA Program Manager in the Software and Intelligent Systems Technology Office (SISTO), Defense Sciences Office (DSO) and Tactical Technology Office (TTO). Professor Khosla's research interests are in the areas of internet-enabled collaborative design and distributed manufacturing, collaborating autonomous systems, agent-based architectures for distributed design and embedded control, software composition and reconfigurable software for real-time embedded systems, reconfigurable and distributed robotic systems, integrated design-assembly planning systems, and distributed information systems. His research is multidisciplinary and has focused on the theme of "creating complex embedded and information systems through composition of and collaboration amongst building blocks." His email and web addresses are <pkk@ece.cmu.edu> and <www.ece.cmu.edu>.