Artifact Meta-Models for Composable Simulation

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

Keywords: artifact models, composable modeling, function hierarchy, attribute grammar

Abstract

This paper presents our artifact meta-models designed for facilitating artifact selection and composable simulation. In particular, meta-models incorporate function and form attributes to explicitly describe external characteristics and inter-connection requirements of artifacts. In addition, artifact meta-models are associated with behavior models, so that a user can choose artifacts with desired function and form features and then use their behavior models in simulation studies.

Our artifact-models are developed in an object-oriented fashion and are specified in extensible markup language (XML). To manage the complexity associated with a large number of artifact attributes, the function and form attributes are organized in a hierarchical fashion. The paper discusses our methodologies developed for designing artifact attribute hierarchy and the research issues that need to be addressed in future work.

1. OVERVIEW

Advanced computer modeling and simulation technology has been playing an increasingly important role in complex system analysis and design. With the rapid advancing computer technology, it is generally faster and cheaper to create, revise and test computer models than physical mockups. Recent research efforts aimed at further improving modeling efficiency have led to modular and object-oriented modeling paradigms and techniques, which use models of components to form system-level models. Such a compositional modeling paradigm distributes complexity associated with complex system modeling to individual components and facilitates reuse of high-fidelity component models, which are expensive and time-consuming to obtain in general.

For the success of compositional modeling paradigms, component models need to follow syntax suitable for model composition and include appropriate information required for component selection. Research in the first problem has made much progress, leading to international standards and various modeling methodologies and platforms in support of development and maintenance of composable models. The information included in component models has so far focused on what is needed for carrying out simulation studies such as behavior equations and geometric CAD models. This kind of technical information, however, is not informative for component selection, since the overall effects of components (such as component functions) cannot be easily inferred from behavior equations but are usually used for specifying requirements for desired components. Therefore, it is important to further augment component models with some high-level descriptions of component attributes suitable for component selection. This kind of information may also be used for organizing, indexing and retrieving components from component repositories.

Since component composition and selection may be conducted through both human interactions and automatic computer operations, it would be desirable to describe component attributes in an interpretable computer language. To manage the complexity associated with a potentially huge, if not infinite, number of component attributes, it is essential to organize component attributes in a structured fashion. This is feasible since component attributes have a naturally hierarchical structure. For example, an attribute “conversion efficiency” will exist for a component of function “converting energy” but not for a component of function “generating rotational motion”. Component attributes from multiple perspectives, such as function, form and behavior, are interdependent, and thus, this dependency information needs to be reflected in component attribute descriptions as well.

This paper will present component meta-models designed to address these issues. Briefly, a component meta-model

includes component attributes from various perspectives, with attributes organized into a hierarchical structure and inter-dependency between attributes explicitly expressed. The attributes are specified in extensible markup language (XML), which is accessible over the Internet and is interpretable for both humans and computers.

2. RELATED WORK

A physical artifact is typically characterized by its form, function, and behavior [1, 2]. The form is a description of the physical embodiment of an artifact, while function is the purpose of the artifact—the (external) effect that is achieved by the artifact through its behaviors. (While there are various definitions for artifact function, we will adopt the one that defines artifact function as a relation between the input and output of energy, material and information flows.) As is illustrated in Figure 1, the actual behavior of an artifact depends on its form and determines its function. More specifically, artifact behaviors are governed by physical laws, which can be expressed (possibly through simplification) as mathematical equations with certain parameters determined by the artifact's form; and the external effect of artifact behavior on its environment is summarized as the mapping between input flows and output flows, i.e. the artifact function.

With this three-facet view of artifacts, the design problem amounts to determining an artifact form that achieves required function through satisfactory behavior at input and output level. Design is generally conducted through a process of decomposition and composition. High-level functions are hierarchically decomposed into functions for subsystems; these sub-functions are then mapped to physical components that are in turn recomposed into a complete system. This is the so-called configuration design process, which leads to designs specified in terms of components and their interconnections with each other. Such a design representation parallels the hierarchical modeling paradigm of a system: models of components are connected to each other via interaction models (describing the dynamics of the component interactions) to form a model of the system. Both representations are based on hierarchical composition: composition of form in design and behavioral models in simulation [3, 4].

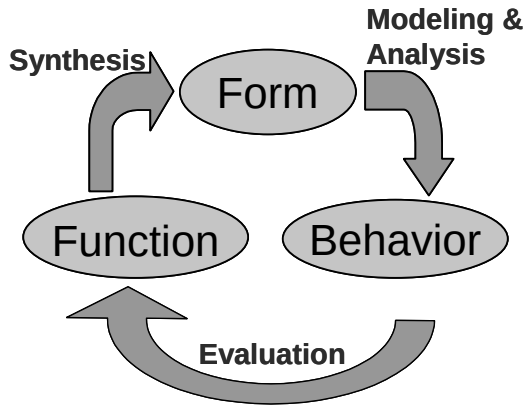

Figure 1. The relation between form, function, and behavior in the context of virtual prototyping

By taking advantage of the parallelism between composition in configuration design and composition in system modeling, a designer may simultaneously specify designs and create their models that can be used to verify proper functioning of the design alternatives. This is already common practice in several single-domain simulations such as electrical systems [5-7] and mechanical systems [8, 9]. The current trend is to extend this approach to multi-domain simulation [10-12].

The software design methodology of object-oriented programming has been applied to systems modeling as well, with the benefits of simplified model creation and maintenance [13-19]. An important principle of object-oriented programming is that of information hiding or encapsulation: only the public interface of an object affects its interconnections with other objects. The same principle can be applied to modeling by making a clear distinction between the physical interactions of an object with its environment (interface) and its internal behavior (implementation)[20, 21]. The advantage of encapsulation is that a system can be modeled by composing (connecting) the interfaces of its sub-systems, independently of the future implementations of these subsystems [4, 21, 22].

To take full advantage of composable simulation, it is also essential to equip composable artifact models with information suitable for artifact selection. As more researchers and engineers adopt composable modeling paradigms in their model development, more composable artifact models will become publicly available. To avoid re-inventing the wheel and reduce modeling costs, it is desirable to reuse developed artifact models whenever possible. While the artifact selection problem involves various information management and database issues such as artifact organization, storage, search, selection and retrieval, this paper will only consider the information representation issue: what information needs to be included and how the information should be organized so that it is easy to check whether an artifact provides a required function, form and/or behavior.

Note that artifact selection requirements are usually specified in terms of desired attributes for artifact function,

form and/or behavior. For example, an artifact selection task may be specified as finding an artifact that can covert electrical energy to mechanical energy with a conversion efficiency of more than 80% and weighing less than 0.5 pounds. Traditional artifact models, developed to support computer modeling, consist of geometric CAD models and mathematical behavioral models. These data-intensive CAD and behavior models, however, cannot be readily used in model selection, since it is very difficult to infer artifact information from them. This indicates the need for augmenting artifact models with explicit information about its function, behavior and form.

Engineering function and form [1, 23-32] have been a research topic in several fields such as AI and engineering design. While previous research in this area has led to inclusion of some additional information other than CAD and behavior models in artifact models, the additional information is usually specified in textual format, which is not convenient for search. Some other work studied information structures for artifact attributes, but usually did so without including modeling information. It is our objective to derive information representation schema that organize artifact attributes in a structured fashion and express inter-relationship between artifact attributes and simulation models.

3. COMPOSABLE BEHAVIOR MODELS

To facilitate simulation-based design, our group has been working on a composable simulation and design paradigm. Similar to traditional artifact models developed for computer simulation studies, our initial artifact models included only CAD and behavior models. The behavior models are designed in an object-oriented fashion with public behavior interfaces separated from their internal implementations. To support artifact selection, the initial artifact models have been expanded to artifact meta-models with artifact function and form attributes organized in a hierarchical structure.

Both our behavior models and artifact meta-models are defined in extensible markup language (XML) and are processed by Java codes. XML is similar in appearance to the Hypertext Markup Language (HTML), but allows for the development of user-defined tags, various types of references and other mechanisms. Its simplicity and flexibility has led to its widespread adoption in the information technology world and expected support in upcoming releases of several commercial web browsers, which will make XML-based models readily accessible across computer platforms. In addition, our meta-model design strategy follows the same strategy as for our behavior models.

There are three major types of identities in our composable behavior models: *behavior interface, behavior implementation and behavior instance*. The behavior interface of an artifact consists of ports and parameters. The parameters completely specify the information needed to model the artifact, and the behavior ports model the exchange of energy, mass, or signals between the artifact and its environment. While behavior interfaces provide information required for using artifacts in system modeling, behavior implementations specify physical laws underlying the artifact's internal behaviors in the form of mathematical equations and composition of behavior models. Since an artifact can be modeled with different levels of detail, a behavior interface can be mapped to multiple behavior implementations. A behavior instance is the binding of a behavior interface to a particular behavior implementation with behavior parameter values assigned.

Figure 2 shows the major schema definition for behavior models. Following the terminologies used for context free grammars and extensible markup language, the notations in the behavior schema can be explained as follows. An equation with assignment symbol ":=" defines a production rule for generating the element on the left hand side from its sub-elements specified between the parentheses on the right hand side. An suffix after a sub-element indicates the cardinality of its occurrence: "?" for 0 or 1 time, "*" for 0

```
behavior-interface := (port*, parameter*)
Attribute (behavior-interface)  = (name)
Attribute (port) = (name, type)
Attribute (parameter) = (name, type, default-value?)

behavior-implementation :=
    (behavior-instance*, equations)
Attribute (behavior-implementation) =
    (name, of-behavior-interface)

equations := (equation | connect)+
Attribute (equation) = (formula)
Attribute(connection) = (from-port, to-port)

behavior-instance := (bound-behavior-interface, bound-
behavior-implementation, parameter-binding)
Attribute(behavior-instance) = (name)
Attribute(bound-behavior-interface) = (name)
Attribute(bound-behavior-implementation) = (name)
Attribute(parameter-binding) = (name, value)
```

Figure 2. Behavior Model Schema

```
<behavior-interface name= “transformer-interface”>
   <port name= “input-port”, type = “energy-port”/>
   <port name= “output-port”, type = “energy-port”/>
   <parameter name = “transform-efficiency”,
                  type = “real”/>
</behavior-interface>
```

Figure 3. Behavior Interface: Transformer

or more times, and “+” for 1 or more times. The comma between sub-elements defines a required sequence of sub-elements, while the vertical bar “|” between sub-elements indicates that an author can choose between the sub-elements. Besides the equations for defining the production rules, the other type of equations in our model schema is used to define XML attributes associated with elements. For those readers who are familiar with XML, it is straightforward to convert our behavior model schema to an XML document type definition (DTD) where the (sub) elements and attributes will be defined as XML (sub) elements and attributes. *Both XML elements and XML attributes are attributes of artifact meta-models*. In the following discussion, we will use the term “XML attributes” explicitly to mean XML attributes, while the term “attributes” without attributive prefixes means meta-model attributes. Figure 3 includes a self-explanatory behavior interface for a transformer as an example of a valid behavior interface under our behavior model schema.

4. ARTIFACT META-MODELS

As discussed in previous sections, one major objective of developing artifact meta-models is to incorporate function and form information with behavior model. This allows users to choose artifacts of desired function and form attributes and then use their behavior models in simulation studies. Following the methodology used to define behavior models, our artifact meta-models are defined with the following three types of entities: artifact interface, artifact configuration, and artifact instance. An artifact interface describes the external effect of the artifact in terms of its function and form attributes as well as possible connections with other artifacts. A configuration describes sub-components and their interconnections used to build the artifact. Each artifact interface may be associated with multiple configurations, reflecting the possibility of achieving the same external effect with different sub-components and/or different inter-connections.

```
artifact-interface := (artifact-port*, function*, form*,
                       behavior-interface-binding*)
Attribute (artifact-interface)  = (name)

Attribute (artifact-port) = (name)

function  := {function-type, flows,
              appropriate-function-parameters}
…
form  := {geometry-type, location-information,
      material-type,  appropriate-form-parameters}
…
behavior-interface-binding := (bound-behavior-interface,
behavior-port-binding, behavior-parameter-binding)
Attribute(bound-behavior-interface) = (name)
Attribute(behavior-port-binding) =
   (artifact-port-name, behavior-port-name)
Attribute(behavior-parameter-binding) =
   (artifact-parameter-name, behavior-parameter-name)
```

Figure 4. Artifact Interface Schema

Since the attributes of artifact functions and forms are included in artifact interfaces, this paper will focus on artifact interfaces. Other related topics such as using artifact configurations to encode various types of design information, like function decomposition and physical decomposition, will be addressed in future papers.

4.1 Interfaces of Artifact Meta-Models

An artifact interface includes artifact ports, function and form attributes, as well as correspondence between artifact interface and behavior interfaces. Figure 4 shows definitions of major elements in artifact interfaces.

In contrast to behavior ports that have name and type attributes, artifact ports only have name attributes. This is because the function and form attributes of ports, which, roughly speaking, define artifact port types, are specified in the function and form parts of artifact interfaces. Artifact port names are mainly used as reference labels to establish the correspondence between ports and their attributes.

Artifact functions are specified in terms of function types, flows and related functional parameters, where flows are characterized by flow types and flow parameters. For example, the function of a transformer is to “transform” (function type) input energy flow to output energy flow with a transformation efficiency (functional parameter) of 0.8. In general, function types define appropriate flows and functional parameters. For example, a transformation function must have both input flow and output flow as well as a parameter for transformation efficiency, while an ideal sink function may have only an input flow. Moreover, function types and flow types can be further refined to their subtypes, which may introduce additional parameters. To help manage the huge number of function related attributes

```
<artifact-interface name= “transformer-interface”>
   <artifact-port name= “input-port”/>
   <artifact-port name= “output-port”/>

   <function>
     <transformation-function>
         <input-flow refers-to-port = “input-port”>
           <energy-flow>
            <max-power name= “max-power1”
            unit= “W” value= “2.0”/>
           </energy-flow>
         </input-flow>
         <output-flow refers-to-port = “output-port”>
           <energy-flow/>
         </output-flow>
         <efficiency name= “transform-efficiency”
            value= “0.8” type= “parameter”/>
     </transformation-function>
   </function>

   <form>
     <port-form refers-to-port = “input-port”>
        <shape>
            <round/>
        </shape>
     </port-form>
   </form>
   <behavior-interface-binding> …
   </behavior-interface-binding>
</artifact-interface>
```

Figure 5. Artifact Interface: Transformer

and to reflect their hierarchical structures, we have developed methodologies for organizing functional attributes and will discuss our methodologies as well as their XML implementation in the next sub-section.

Artifact form attributes include artifact geometry (shape and location, etc.) and material. Similar to artifact function attributes, form attributes are hierarchical. For example, an artifact of shape “round” has a form parameter “radius” while another artifact of shape “cubic” has a form parameter “length”. We have developed similar organizational methodologies for form attributes as those for functional attributes; but due to space constraints, we will not discuss organization schemes for form attributes in this paper.

In function and form attributes, flows and port forms are related to artifact ports: flows specify what type of entities such as energy, material, or signal the artifact will inter-change with other artifact through the ports; and port forms describe port-related form attributes. The XML attribute “refers-to-port” is used in flow and port-form attributes to indicate corresponding artifact ports. Consequently, the properties of artifact ports are specified by those flow and port-form attributes. For example, the input-port of the transformer in Figure 5 may accommodate input energy flow with up to 2.0 Watts of power and has a round shape.

In summary, an artifact meta-model organizes function and form attributes in hierarchical structures with the relationships between attributes of different perspectives explicitly represented. This makes a meta-model a graph instead of a tree. In addition to describing artifact properties, the hierarchical function and form structures can also be used to represent design specifications. Then, for a given design specification, an artifact may be selected if its function and form attribute graph includes the same pattern as that of the design specification. This essentially formulates the artifact selection problems as a graph-matching problem.

4.2 Function Attribute Hierarchy

To facilitate artifact indexing, selection and organization, artifact models need to use standard function and form terminologies and satisfy relationship constraints agreed upon by all relevant parties. There have been considerable efforts in developing standard function representations [31, 32]. While our function attribute hierarchy incorporates some results developed in this community, the main objectives of our work in this area are to develop methodologies for designing the hierarchical structures of function attributes and gain insights for future research in standard function representation.

As discussed briefly in the previous section, function attributes consist of function types, flows and functional parameters. Function types impose different constraints on flows. For example, the function “convert” should have input and output flows of different types, while the function “amplify” should have input and output flows of a same type. Assuming that functions and flows have a semantic attribute called “type” and functions have a semantic attribute called “valid”, the above observation can be written in the format of attribute grammar as follows.

function :=(“amplify” |“convert”|..., input-flow, output-flow)

(1)

```
valid (function) :=
((type (function)== “amplify”)  and
    (type(input-flow) == type(output-flow))) or
((type(function)== “convert”)  and
    (type(input-flow) != type(output-flow)))
```

Although XML is well suited for defining context-free grammar, it does not provide mechanisms for specifying semantic attribute grammar. Nevertheless, there are two ways to incorporate semantic information underlying the attribute grammar in our system. One way is to hand-encode this information in Java code used for processing XML files of artifact meta-models and then use the JAVA code to enforce the attribute grammar. The major drawback with this approach is that the Java code needs to be updated as additional attribute grammar rules are introduced, for instance when meta-modes are generalized to cover more aspects of artifact information such as cost and manufacturing process. The other way is to translate the attribute grammar to a context-free grammar. For example, the attribute grammar as listed above can be translated to the following:

function := (“amplify-function” |“convert-function”|...)
amplify-function := (elec-elec-flow-pair | mech-mech-flow-pair | ...)

convert-function := (elec-mech-flow-pair | mech-elec-flow-pair | ...)

Essentially, such an approach enumerates all valid choices under an attribute grammar, which can lead to a very large context-free grammar and corresponding XML DTD.

If neither of the above approaches is adopted, i.e., our function DTD just specifies the production rule as in formula (1) and the Java code does not include hand-code for attribute grammar, our system will not prevent the generation of semantically-incorrect artifact meta-models (such as an artifact that “amplifies” electrical energy to mechanical energy). This kind of semantically incorrect models, even if generated by mistakes, will likely not find similar models in artifact repositories. This can alert the modeling engineer that there is a possible mistake in the model definition. We are in the process of exploring the possibility of using more powerful markup languages for meta-models that provide mechanisms for specifying semantic attribute grammar, such as XML Schema or DAML.

As for function parameters, it is possible to define a generic parameter element where an author can specify parameter name, unit, value and other information. But there is no way to control generic parameter specifications in a function hierarchy, which may cause an inappropriate parameter value being associated with function or flow types. For example, a direct-current (DC) flow may be associated with a generic parameter that defines a “frequency” parameter, which is not proper for DC. So we propose to use as many explicit parameter tags as possible, but use generic parameter tags as a back up. Also we propose to move parameters to as high positions as possible in hierarchy, to reflect the natural hierarchy structures. For example, a “maximum power” parameter can be associated with an energy flow before the energy flow is further refined to electrical energy flow (which may have a parameter called “maximum current”).

Our function and flow type hierarchy follows NIST’s function and flow hierarchy [31, 32]. But our function and flow schema are different. For example, our functions and flows have specific parameters while NIST functions and flows only use generic parameters. In addition, for a compound artifact, the NIST schema associate flows with sub-components, from which the flows of compound artifacts need to be derived. But our flows are associated with artifact ports, which can be further associated with ports of sub-components. This approach frees us from the task of deriving flows for compound artifacts and makes artifact interfaces more informative without introducing dependencies on artifact configurations.

4.3 Behavior Interface Binding

In addition to specifying artifact ports and function/form attributes, artifact interfaces include correspondence between artifact ports/parameters and behavior ports/parameters. This correspondence ensures that a behavior model of an artifact can be instantiated and used in system modeling after the artifact is used in a system configuration.

While behavior parameters are explicitly defined as one group in behavior interfaces, artifact parameters are embedded in function and form attribute hierarchies. This is because parameters are an integrate part of function and form descriptions. Note that function and form parameters may be different from behavior parameters. For example, function parameters of a controller may be specified in terms of control stability criteria while behavior parameters are control parameters required for achieving the desired stability performance. Therefore, there usually exists a mathematical relationship between artifact parameters and behavior parameters. This relationship needs to be specified as part of behavior interface binding of an artifact interface and may be represented in various ways, such as through a Microsoft Excel spreadsheet. For simplicity, the data schema in Figure 4 only includes parameter mapping for the simplest case where there is a one-to-one correspondence between artifact parameters and behavior parameters.

As for ports, depending on the modeling strategies for artifact interfaces and behavior interfaces, there might exist a one-to-many and many-to-one mapping between artifact ports and behavior ports [33], an issue beyond the scope of this paper. Figure 4 shows again the simplest case for behavior-port binding where there is one-to-one correspondence between behavior ports and artifact ports. The artifact ports and parameters of the transformer shown in Figure 5 have the same names as their corresponding behavior ports and parameters in Figure 3; the behavior-interface-binding part is omitted in Figure 5

From an implementation point of view, the behavior-interface-binding part can be specified outside artifact interface and as a two-tuple mapping an artifact interface to a behavior interface. This will leave the artifact interface definition independent of the behavior interface development and allow artifact interfaces to be associated with behavior interfaces that are developed after artifact interfaces, without changing artifact interfaces.

The correspondence between artifact parameters and behavior parameters makes it possible to instantiate a behavior interface. With the correspondence between artifact ports and behavior ports, connections between two artifacts can be mapped to connections between their corresponding behavior models (with possibly additional operations for non-bijective port mapping). One subtle yet critical point to notice here is that inter-connections between artifacts usually involve non-trivial interaction behaviors such as rolling motion between two gears. Therefore, it is important to include appropriate interaction behavior models in the behavior modeling of connecting artifacts[19, 34]. Figure 6 shows a Motor-Pulley configuration and its behavior model with an appropriate interaction model between the motor and the pulley.

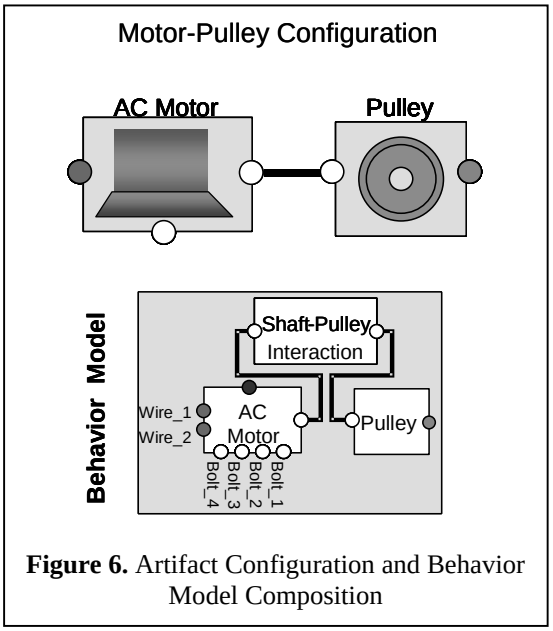

Figure 6. Artifact Configuration and Behavior Model Composition

5. SUMMARY

This paper has presented our artifact meta-models designed for facilitating artifact selection and composable modeling. In particular, meta-models incorporate function and form attributes to explicitly describe external characteristics and inter-connection requirements of artifacts. In addition, artifact meta-models are associated with behavior models, so that a user can choose artifacts with desired function and form features and then use their behavior models in simulation studies.

Our artifact meta-models are developed in an object-oriented fashion, with artifact public interfaces and internal configurations separated. This will allow association of multiple artifact configurations with one artifact interface reflecting the phenomena of achieving a same public interface (external effect) with different components and their inter-connections. Further, our artifact meta-models, as well as our composable behavior models, are defined in extensible markup language, which can be accessed across computer platforms and interpretable for both human beings and computer agents.

To manage the complexity associated with large number of artifact attributes, the function and form attributes are organized in a hierarchical fashion. We have discussed our methodologies used for designing artifact attribute hierarchy and the implementations of the hierarchy in extensible markup language. As discussed in the paper, XML is suitable for specifying context-free grammar; but function and form attributes follow attribute grammar. So we will explore other representation languages such as DAML. We are also interested in incorporating more aspects of artifact information into meta-models such as cost model, statistical model, and manufacturing/assembly process. Another part of ongoing research in our group is meta-model-based artifact indexing, organization and selection.

ACKNOWLEDGEMENTS

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

This research was funded in part by DARPA under contract ONR # N00014-96-1-0854, by the National Institute of Standards and Technology, by the National Science Foundation under grants # CISE/115/KDI 98 73005 and # EIA-97 29827, by the Pennsylvania Infrastructure Technology Alliance, and by the Institute for Complex Engineered Systems at Carnegie Mellon University.

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

[13] H. Elmqvist, S. E. Mattsson, and M. Otter, "Modelica: The new object-oriented modeling language," presented at The 12th European Simulation Multiconference, Manchester, UK, 1998.

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

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

[16] P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg, "ASCEND: An object oriented computer environment for modeling and analysis. 1 - The modeling language," *Computers and Chemical Engineering*, vol. 15, pp. 53-72, 1991.

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

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

[19] R. Sinha, C. J. J. Paredis, V.-C. Liang, and P. K. Khosla, "Modeling and Simulation Methods for Design of Engineering Systems," *ASME Journal of Computing and Information Science in Engineering*, vol. 1, pp. 84-91, 2001.

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

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

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

[23] W. Rodenacker, *Methodishes Konstruieren*: Springer, Berline, 1971.

[24] B. Chandrasekaran, "Functional representation: A brief historical perspective," *Applied Artificial Intelligence*, vol. 8, pp. 173-197, 1994.

[25] B. Chandrasekaran and Josephson, "An Explication of Function," presented at AAAI-96 Workshop on Modeling and Reasoning about Function, Portland, OR, 1996.

[26] P. A. Fishwick, "A functional/declarative dichotomy for characterizing simulation models," presented at

1992 Artificial Intelligence, Simulation and Planning in High Autonomy Systems, Perth, Australia, 1992.
[27] M. Pegah, J. Sticklen, and W. Bond, "Functional Representation and Reasoning About the F/A-18 Aircraft Fuel System," *IEEE Expert*, pp. 65-71, 1993.
[28] M. Schulte, C. Weber, and R. Stark, "Functional Features for Design in Mechanical Engineering," *Computers in Industry*, vol. 23, pp. 15-24, 1993.
[29] M. Sasajima, Y. Kitamura, M. Ikeda, and M. Mizogochi, "Representation Language for Behavior and Function: FBRL," *Expert systems with applications*, vol. 10, pp. 471-479, 1996.
[30] Y. Iwasaki, A. Farquhar, R. Fikes, and J. Rice, "A web-based compositional modeling system for sharing of physical knowledge," presented at International Joint Conference on Artificial Intelligence, 1997.
[31] S. Szykman, J. W. Racz, and R. D. Sriram, "The Representation of Function in Computer-Based Design," presented at 1999 ASME Design Engineering Technical Conferences - Design Theory and Methodology, Las Vegas, Nevada, 1999.
[32] S. Szykman, J. Senfaute, and R. D. Sriram, "The use of XML for describing functions and taxonomies in computer-based design," presented at 19th DETC/Computers and Information in Engineering Conference, Las Vegas, Nevada, 1999.
[33] L. Han, C. J. J. Paredis, and P. K. Khosla, "Object-Oriented Libraries of Physical Components in Simulation and Design," presented at 2001 Summer Computer Simulation Conference, Orlando, FL, 2000.
[34] R. Sinha, C. J. J. Paredis, and P. K. Khosla, "Modeling of Component Interactions in Configuration Design," Carnegie Mellon University, Pittsburgh, PA, Technical Report 2001.