Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

A PORT ONTOLOGY FOR AUTOMATED MODEL COMPOSITION

Vei-Chung Liang

Institute for Complex Engineered Systems
Carnegie Mellon University
Pittsburgh, PA 15232, U.S.A.

Christiaan J.J. Paredis

Systems Realization Laboratory
G.W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology
Atlanta, GA 30332, U.S.A.

ABSTRACT

We study the concept of ports and we define an ontology for representing them. Ports define the locations of interaction at the boundaries of components or sub-systems; they can be used across different disciplines for both product modeling and simulation. They are therefore a convenient abstraction that allows simulation modelers to modularize and encapsulate their system descriptions such that configurations of port-based product models can be used to generate multiple simulation models at different levels of abstraction. However, to combine system models effectively across different disciplines, the representation of the ports needs to be unambiguous yet flexible, so that it can accommodate the differences in vocabulary and approach of all the disciplines. We provide an overview of how a port ontology, defined in the web ontology language, OWL, can capture both syntactic and semantic information such that automated modelers can reason about the system configuration and corresponding simulation models.

1 PORTS FOR AUTOMATED MODEL COMPOSITION

1.1 Ports

Ports constitute the interface that defines the boundary of components or sub-systems in a system configuration. As illustrated in Figure 1, a system can be represented as a configuration of components or sub-systems that are connected to each other through well-defined interfaces. The configuration interface of a component object consists of ports, which define the intended interaction between a component and its environment; interactions consist of the exchange of energy, matter, or signals (information). For instance, the configuration interface of the motor in Figure 1 has ports for the stator, the shaft of the rotor, and the electrical connectors.

It is through its ports that a component (sub-system) interacts with other components (sub-systems), as is indicated in the graph by the connection between ports. The fact that these interactions have been abstracted into ports does not imply that only components with standardized connectors can be defined in this fashion. When interfaces are not completely standardized (e.g. a weld between two structural elements), the interaction can still be abstracted into one of a relatively small set of general interactions types (e.g. a rigid mechanical connection).

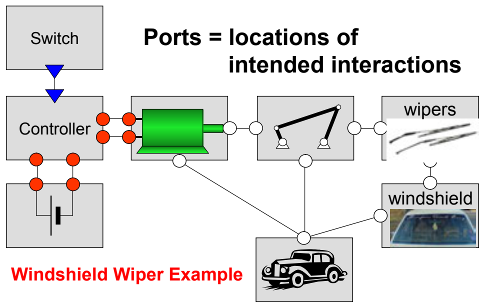

Figure 1: A System Configuration of Configuration Interfaces

1.2 Association Models

Let us now take a look at how the configuration interfaces can be useful beyond just representing the system architecture. Assume that each configuration interface is linked to an association model that establishes the relationships between geometric, functional, and behavioral models, as is illustrated in Figure 2. An association model may contain multiple simulation models at different levels of detail or from different disciplinary perspectives. For instance, a solar panel on a satellite can be modeled as a source of electrical energy, as flexible mechanical inertia, or both, depending on the analysis for which the simulation will be used. Regardless of the choice of model, however, the system configuration (the location and type of connection between the solar panel and the rest of the satellite system) remains the same.

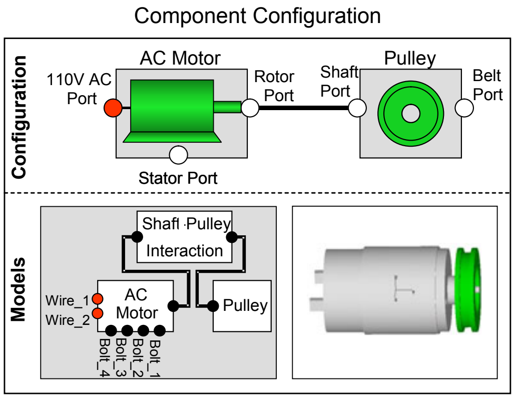

Figure 2: An Association Model Contains a Configuration Interface and Models with Different Product Perspectives

In addition to simulation and geometry models, the integrated component representations or association models also include the relationships between the configuration interface and the models. The relationship between the ports of the configuration interface and the ports in the behavioral interface is often, but not always, a one-to-one mapping. For instance, the shaft of the AC motor corresponds to a single mechanical energy port, but the AC plug configuration port is modeled as two electrical ports, one for each pin.

1.3 Model Composition

As a result of the mapping between configuration interfaces and the corresponding simulation models, the definition of a system architecture as a composition of component objects allows us to create a composition of simulation models that constitute the corresponding system-level model. As is shown in Figure 2, the design configuration consisting of the pulley mounted onto the motor shaft can be represented by connecting the shaft port of the pulley to the rotor port of the motor. The corresponding simulation model is obtained by connecting the simulation models of the motor and pulley through the corresponding ports. The ports in this scenario play an important role of providing and specifying interaction constraints between artifacts (shaft and pulley) and among representations (configuration interfaces, simulation models, and CAD models).

Through this mechanism, we could define a system (a configuration of component objects) and automatically compose the corresponding simulation model. This is already common practice in electrical CAD software (Mentor Graphics 2000); when creating a chip layout, the instantiation of a transistor or logic gate defines the geometry for the silicon layers as well as the corresponding simulation model. Similarly, Zeigler, Praehofer et al. (2000) has introduced a modular, hierarchical, port-based representation for discrete event simulation. These DEVS models can be associated with entity structures in the System Entity Structure/Model Base Framework, allowing for composition at the structural level. In mechanical CAD, the integration between design and simulation is not as common. For purely mechanical systems, most mechanical CAD packages do provide an optional module for multi-body simulation (Duckering 2000), but these modules do not support port-based configuration and lack sufficient support for multi-disciplinary systems. Our research aims to extend these ideas to *simulation-based design of multidisciplinary systems*.

To support automating the modeling and simulation process, it is essential that the relationships between the different aspects and perspectives of a component are represented explicitly. To determine which interaction models and corresponding simulation models need to be used for the simulation of a particular system, information is needed beyond what is contained in the configuration interfaces and simulation models themselves. This paper introduces representations for ports that form the bridge between component configuration and composition of the underlying models. To support automating the composition process, these representations need to be unambiguous, computer interpretable, and sufficiently broad so that they can provide access not only to configuration and simulation information, but also to function and form.

Our approach is based on semantically rich product models that include not only simulation models, but also function and geometry models. The approach takes advantage of recent developments in Information Technology, including semantic data formats, ontologies, and knowledge repositories. This is part of a general trend to move from data-centric to knowledge-centric representations, a trend that has been the focus of several ongoing research efforts related to engineering design (Eastman and Fereshetian 1994; Wood and Agogino 1996; Counsell, Porter et al. 1999; Susca, Mandorli et al. 2000; Szykman, Sriram et al. 2000), but could also benefit the simulation area. A good overview is provided in (Benson and Terpenny 2001). Unfortunately, the representation of ports has received little or no attention in this context.

2 WHY A PORT ONTOLOGY?

In order for product models to be useful for knowledge representation, the information encoded in product models needs to be unambiguously understood by all analysts, independent of their perspectives, physical locations, and times. Ambiguity may arise when multiple terms are used to mean the same thing, or when one term is used with multiple meanings. For example, a design concept may have multiple descriptions: a hinge can also be called a rotary joint or a groove can also be called a notch.

There are two general approaches to support unambiguous computable representations: labels and metadata. Giving a port a unique label or a name is a common implementation in computer aided design applications. The benefit of using labels is that it is easy to create by designers. However, the label approach requires extra effort to effectively sort and retrieve synonyms and maintain relations among the same terms used for different design concepts.

On the other hand, the metadata approach assigns primitive and compound attributes to the terms used to define concepts. For example, a hinge can be defined as a connection that cannot resist the external moment around the hinge axis. If a rotary joint is defined with the same degree of freedom attribute, then a computer application or engineer can infer that a hinge is also a rotary joint. The metadata approach is superior to the label approach since it compares not only the syntax but also the semantics.

However, the implementation of the metadata approach is not straightforward. The definition language must have the capability to define not only the syntax productions but also the semantic rules for the design concepts. Defining a port taxonomy in the Extensible Markup Language (XML) (Bray, Paoli et al. 2000) has been proposed by Sinha, Paredis et al. (2001). XML is similar to the Hypertext Markup Language (HTML), but allows user-defined tags and various types of references. Its simplicity and flexibility has led to widespread adoption in the Information Technology world. While it provides an important solution for making Internet information computer interpretable, XML by itself has a limited expressiveness for describing the relationships (schemas or semantics) between concepts. XML regulates only syntactic and structural relationships among tags. Most of the semantics of the tags (other than "has-a" and "one-of" relations) have to be hard coded within the parsing modules of the applications. It would be good if one could incorporate the semantics in the representation itself. Our approach for providing such representations is based on ontologies. An ontology can be informally defined as a description of concepts and relationships that are used in a specific knowledge domain.

3 RELATED WORK ON ONTOLOGIES

Ontology representations convey and encapsulate both syntax and semantics, allowing computer programs to share, exchange, extend, reuse and translate information. The representations can be based on either frame-based logic or description logic (Fensel 2000). In the frame-based language, Ontolingua (Farquhar, Fikes et al. 1997), the knowledge domain is described using frames and slots. On the other hand, the description logic languages provide declarative statements to describe the relations between concepts and relations. These statements are collected and processed later by the reasoner to create a complete terminology network. Representative description logic systems includes CLASSIC (Borgida, Brachman et al. 1989), FaCT (Horrocks 1998), and RACER (Haarslev and Moller 2001).

Two ontology languages, the DARPA Agent Markup Language (DAML) (Hendler and McGuinness 2000) and DAML+OIL (Ontology Inference Layer) (Fensel, Horrocks et al. 2001; Fensel, van Harmelen et al. 2003) have recently been merged and extended into the Web Ontology Language (OWL) as a W3C working draft (Dean, Connolly et al. 2002). Both frame-based logic and description logic reasoners can be used to handle OWL.

So far, these languages have been used to build ontologies mostly in the areas of computer science and social sciences. We propose to study the applicability of these ontology tools for product representations in the context of system simulation and design. Our work will focus on the representation of engineering ports, which has not yet been addressed in the ontology literature. A few other research efforts are underway towards ontologies: the PHYSSYS engineering ontologies for engineering modeling, simulation and design (Borst, Akkermans et al. 1997), the Collaborative Device Modeling Environment (CDME) developed by Iwasaki, Farquhar et al. (1997), a LEGO assembly ontology (Kopena, Peysakhov et al. 2002), and taxonomies for function representations (Szykman, Racz et al. 1999; Stone and Wood 2000; Hirtz, Stone et al. 2001).

4 OWL CLASSES AND PROPERTIES

In this section, we explain the principles and OWL constructs needed to design and define the port ontology. For a more in-depth overview of the OWL language, refer to (Dean, Connolly et al. 2002).

Ontologies in OWL consist of *concepts* and *relations*. The concepts of the knowledge domain are defined as OWL *classes*, while the relations are defined as *properties*. For instance, a generic port is defined by the OWL class expression:

```
<owl:Class rdf:ID="port"/>
```

Several other OWL constructs are provided to define necessary and/or sufficient axioms for a class. For instance, we can define a LEGO-port subclass using the "subClassOf" axiom:

```
<owl:Class rdf:ID="LEGO-port">
    <rdfs:subClassOf rdf:resource="#port"/>
</owl:Class>
```

Similarly, the "equivalentClass" and "unionOf" axioms can be used to express that a LEGO-male-port is the union of a stud port and a pin port:

```
<owl:Class rdf:ID="LEGO-male-port">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf df:parseType="Collection">
        <owl:Class rdf:ID="#LEGO-stud-port"/>
        <owl:Class rdf:ID="#LEGO-pin-port"/>
        </owl:unionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
  <owl:Class rdf:ID="LEGO-stud-port">
    <rdfs:subClassOf
      rdf:resource="#LEGO-male-port"/>
  </owl:Class>
```

A third way to describe a class uses the "disjointWith" axiom to disallow, for instance, a port instance to belong to both the pin and stud classes simultaneously:

```
<owl:Class rdf:ID="LEGO-pin-port">
  <rdfs:subClassOf
    rdf:resource="#LEGO-male-port"/>
  <owl:disjointWith
    rdf:resource="#LEGO-stud-port"/>
</owl:Class>
```

Once the classes are defined, the corresponding port instances are instantiated as follows:

```
<LEGO-stud-port rdf:ID="stport1"/>
<LEGO-pin-port rdf:ID="pinport2"/>
```

In addition to concepts defined as classes, OWL includes *properties* that describe relationships between concepts. One can think of properties as directed links between concept nodes. The starting node is the *domain element*; the ending node is the *range element*. For instance, a port (domain) may be related to a feature (range) through the property "hasFeature."

There are two types of properties: *object properties* and *datatype properties*. For object properties, the range element is an instance of a class, while for datatype properties, the range element is a primitive data type such as a byte, date, or number:

```
<owl:ObjectProperty
rdf:ID="hasFeature"/>
<owl:DataTypeProperty
rdf:ID="ManufactureDate"/>
```

To impose constraints on the types of classes that can be related by a particular property, one can use the "domain" and "range" constructs:

```
<owl:ObjectProperty rdf:ID="hasFeature">
  <rdfs:domain rdf:resource="#port" />
  <rdfs:range  rdf:resource="#feature" />
</owl:FunctionalProperty>
<owl:DatatypeProperty
    rdf:about="#manufacturDate">
  <rdfs:domain rdf:resource="#artifact"/>
  <rdf:range rdf:resource="&xsd;dateTime"/>
</owl:DatatypeProperty>
```

In this example, we specify that the range element of the Datatype property, "manufactureDate," is the XML schema "dateTime" type. Once we add these constraints, the "hasFeature" property can only relate features to ports, while the "manufactureDate" property is limited to relating "dateTime" instances to artifacts.

Similar to the "subClassOf" axiom, OWL also provides the "subPropertOf" axiom. For example:

```
<owl:ObjectProperty
    rdf:about="conveyMechanicalEnergy">
  <rdfs:subPropertyOf
    rdf:resource="#conveyEnergy"/>
  <rdfs:domain rdf:resource="#port"/>
  <rdfs:range rdf:resource="#mechanical"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="conveyEnergy">
  <rdfs:domain rdf:resource="#port"/>
  <rdfs:range rdf:resource="#energy"/>
</owl:ObjectProperty>
```

Here we define the "conveyMechanicalEnergy" property as a sub-property of "conveyEnergy". This states that any port conveying mechanical energy can be treated as a port conveying energy. If we want to search for all the ports conveying energy, then the inference engine will return all the port instances that have relations with either energy or mecahnical energy.

OWL also provides the "inverseOf " axiom that allows instances to be associated with each other. For example:

```
<owl:ObjectProperty rdf:ID="hasArtifact">
  <owl:inverseOf
       rdf:resource="#hasInterface"/>
</owl:ObjectProperty>
```

When a "hasArtifact" property individual <interface1, artifact1> is instantiated, a reversed instance pair <artifact1, interface1> will also be instantiated and added as a property individual of the "hasInterface" property.

To improve the readability of the ontology statements, we will henceforth use a graphical representation rather than OWL statements, as shown in Figure 3, where nodes represent classes or individuals and directed labeled links represent properties.

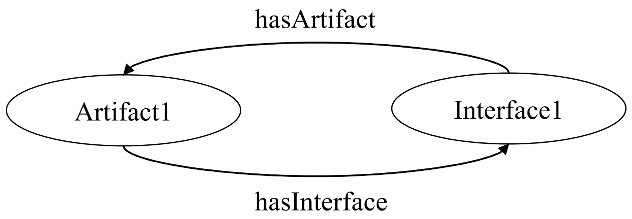

Figure 3: The Graphical Representation of OWL Statements

5 DESIGN AND DEFINITION OF THE PORT ONTOLOGY

As pointed out in the previous section, an ontology consists of classes and properties. In the port ontology, the classes include the ports themselves as well as the attributes that allow us to define the ports. These classes are a subset of the artifact ontology, which can not only describe the interface but also the internal characteristics of components and sub-systems.

5.1 Attribute Classes

The attributes are lower-level concepts for defining ports. We have divided the attributes into three main categories: form, function and behavior. The form attributes describe the structural, geometrical, topological, and part-whole information of an artifact. In this context, attributes are often referred to as features. There exists already a large number of concepts for defining form from which we can borrow (ISO 1994). However, it is often useful to introduce new form attribute classes for specific standardized port geometry. For example, rather than defining the detailed form features of an RJ-45 connector every time one is used, one could refer to the entire geometric specification for such a connector with one label, for instance, *RJ-45-male*. In the next section, we will use the same approach for standardized LEGO features.

In addition to form, ports are defined by function attributes. These attributes describe the intended use of the port. Artifact functions have also been researched extensively, and here also, we will leverage the concepts defined by others (Hirtz, Stone et al. 2001). Since ports, by definition, refer to locations of intended interaction, the functions that can apply to ports are limited to different types of interaction, such as:

- transfer (of energy, material, or signals)
- connect (fasten or attach)
- support (secure and position).

Finally, ports are characterized by behavioral attributes. Again due to limited range of functions that can be performed by ports, their behavioral attributes are also limited to characterizations of energy flow, material flow, or signal flow. For example, a port that is intended to establish a rigid connection with another port can be characterized by vectors for position and orientation combined with vectors for forces and torques. For the definition of behavioral attributes, we build on the Modelica simulation language (Elmqvist and Mattsson 1997).

5.2 Port-Attribute Properties

The relationships between ports and attributes are expressed as properties in OWL. In general, the relationship can be expressed as:

```
<owl:ObjectProperty
    rdf:about="#has-attribute">
  <rdfs:domain rdf:resource="#port"/>
  <rdfs:range rdf:resource="#attribute"/>
</owl:ObjectProperty>
```

However, it would be confusing to use the same OWL property, "has-attribute," to define the relationships between all possible ports and attributes. Instead, we use OWL sub-properties to define specific categories of relationships such as *has-function*, *has-form-feature*, and *has-behavior-connection*. The general "has-function" property can then be used to refer to any function attribute or its sub-classes, for instance, either "transfer", "transfer-energy", or "transfer-signal" (Figure 4). We define the "transfer" function as a subclass of the function attribute with two child classes: transfer-signal and transfer-energy. In addition to these general properties, more specific properties are defined if the restrictions for a particular relationship are different. OWL allows both restrictions on the cardinality and the value of a property.

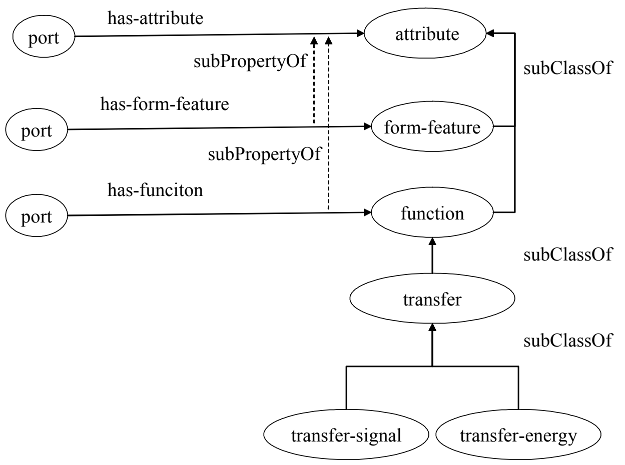

Figure 4: Partial Port-Attribute Properties and Attribute Hierarchy

5.3 Port Classes and Hierarchies

Once the attributes for form, function, and behavior are defined, they can be used to describe all the ports. As with attributes, ports can be defined at different levels of detail, in a hierarchical fashion. Although one could describe this hierarchy explicitly using the "subClassOf" property, such a representation has the disadvantage that it duplicates the information that is already captured in the attributes. Through a process of subsumption, one can consider Class A to be the child of Class B if A contains all the attributes of B and possibly more. When adding attributes to a port class, the class definition becomes more detailed resulting in a child class. Through the process of subsumption, it is possible to derive the inheritance relationships from the attributes of the port classes without having to specify the parent-child relationships explicitly.

6 LEGO PORTS: AN EXAMPLE

LEGO Ports are standardized with fixed dimensions and compatible shapes. We have identified 24 common ports some of which are illustrated in Figure 5. The circled areas are the ports on the LEGO parts: (a) rail-port, (b) stud-port, (c) circular-hole-port, (d) TECHNIC-stud-port, (e) TECHNIC-tube-port, (f) axle-hole-port, (g) channel-port, (h) tube-port, (i) friction-pin-port, and (j) axle-port. Most of them can be connected to each other by snapping together male and female ports. For example, studs (male ports) at the top of a LEGO brick snap into tubes (female ports) at the bottom of another LEGO brick. Some LEGO ports are compatible with multiple other ports. For example, a cross-shaped shaft can fit into a cross-shaped hole, but can also fit into a circular hole with the same circumscribed radius.

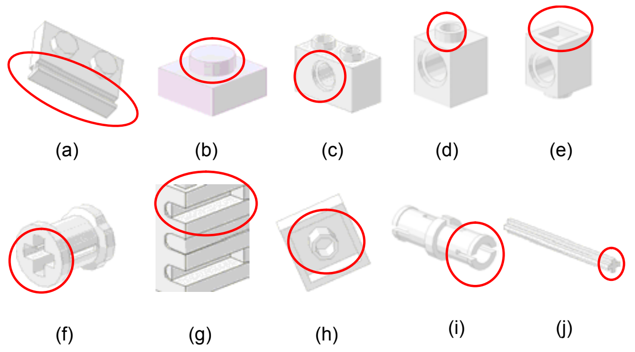

Figure 5: LEGO-Ports

As pointed out earlier, one could represent the hierarchy of ports explicitly starting with a top-level generic port class from which every port class inherits, as is shown in Figure 6. One could further refine this top-level LEGO-port into a snapping-port and two disjoint subclasses, male-port and female-port, followed by a geometric and functional classifications (pin-ports with or without friction). However, without also defining the attributes of each of these ports, such an explicit hierarchy is of limited use. By considering the attributes, one can, through subsumption, create many different taxonomies based on the order in which the attributes are considered. For instance, compare Figure 6 and Figure 7, in which each classifier contains the unique set of form-feature attributes, which group ports with similar features together.

7 USE OF THE PORT ONTOLOGY

In this section, we demonstrate the usage of the LEGO port ontology. We assume that the target users will be system designers who want to integrate a set of product models (subsystems). They integrate the subsystems at the product model level instead of at the behavior model level in order to make sure that the system satisfies not only behavior constraints but also function or form constraints. The port

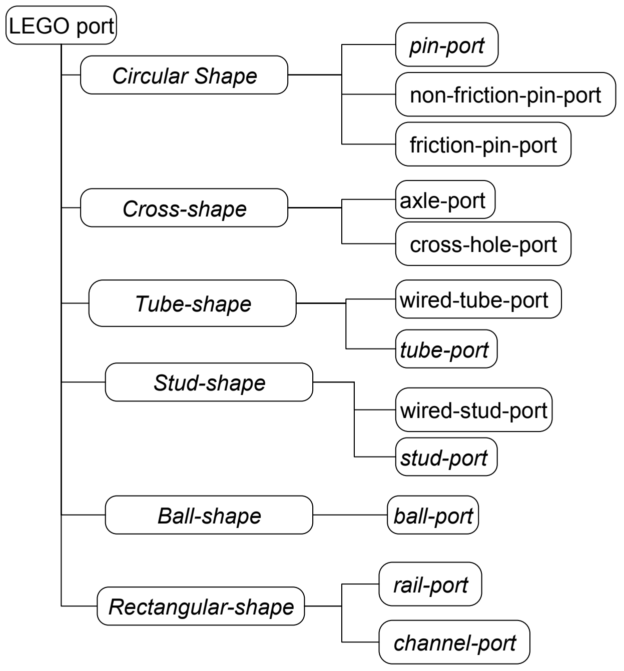

Figure 6: A Different LEGO Port Taxonomy Obtained by Using the Subsumption Mechanism

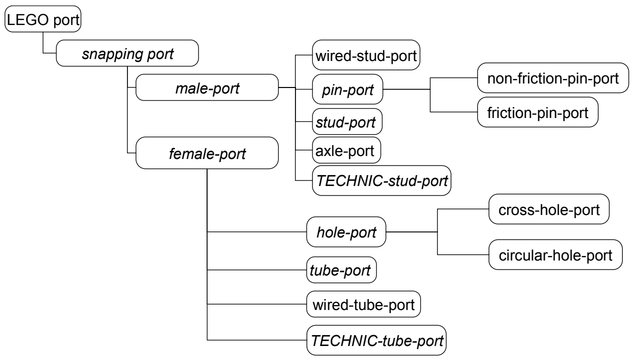

Figure 7: A Partial View of One of Many Possible LEGO-port Taxonomies

ontology is used in a design environment to represent the ports of the product model and to constrain the connections between ports. The first example shows how one can use the port ontology to represent and verify compatibility between the ports in a connection. The second example illustrates how one can reason with port ontologies to select interaction models automatically. Both compatibility checking and interaction model selection are prerequisites for automated model composition.

Please, note that this is ongoing work. The ideas presented here are our vision of how the field of simulation can benefit from semantically-rich port representations.

7.1 Port Compatibility in Connections

A system can be defined as a configuration of components by connecting the components at their ports. However, to be connected two ports need to be compatible: a 110V plug does not fit in a 220V outlet, or a square plug does not fit a round hole. In this section, we illustrate how a port ontology can be used to define general rules for port compatibility.

The port ontology that we have defined so far does not include the concept of compatibility. One could include a property "is-compatible-with" to identify the port types that are compatible with each other. In Figure 8, for instance, we define the compatibility for a circular-hole-port in LEGO. The rule explicitly specifies that only the axle-port and pin-port can connect to the circular-hole-port.

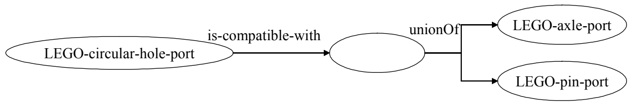

Figure 8: A Compatibility Rule for Circular-Hole-Ports

This compatibility rule is solely based on port *names*. The disadvantage of using only port names is that when a new port class is added to the port ontology, many compatibility rules also need to be updated. Even adding a port with the exact same usage but a different name will require updating the compatibility rules.

A more general approach is to use attributes to describe the compatibility constraints. A circular-hole-port can be connect to all ports with certain geometric features. One could express this rule using low-level geometric constraints on the type and dimensions of port features. However, in the case of LEGO, with its standardized port geometry, we can use the names of form-attributes instead.

In Figure 9(a), we restate the compatibility of the circular-hole-port in terms of form-attributes. Figure 9(b) list several ports that satisfy the compatibility constraint. The unlabeled classes in Figure 9(a) are called anonymous classes; they match any class for which the specified properties hold. When adding a new port type such as a LEGO-TECHNIC-pin-port, we only need to specify that it has a LEGO-pin-shape form-feature to define its compatibility properties. Compatibility checking occurs when two product models are connected. The OWL description of the ports can be processed by the description logic reasoner. The reasoner will verify that all the attributes of the two ports satisfy the compatibility requirements specified in the port ontology. Consider the example in Figure 9. In a description logic reasoner, the port definitions and compatibility rules are stored in the T-box (Donini, Lenzerini, et al. 1996). The T-box is a collection of axioms describing the true conditions of the port connection domain. When a port connection is established, the system queries the reasoner to verify that the connected port instances satisfy all the axioms in the T-box. For example, the axiom in Figure 9

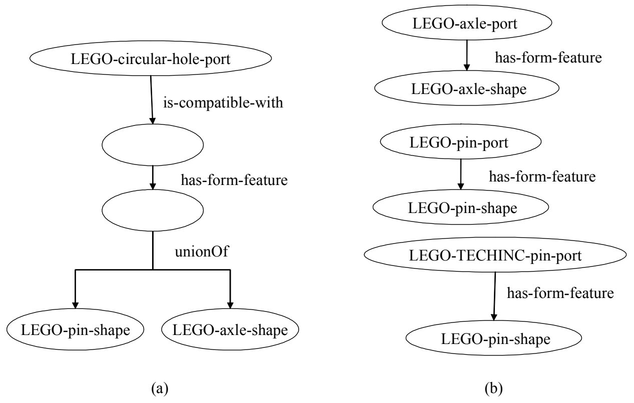

Figure 9: (a) A Circular-Hole-Port Compatibility Rule and (b) Three Possibly Satisfied LEGO Ports

expresses that to connect to a LEGO-circular-hole-port, a LEGO port must have either a pin-shape or an axle-shape form attribute. Note that this compatibility checking is different from Ptolemy II (Liu 2001), which enforces compatibility at the behavior model level; ours enforces it at the product model level.

7.2 Selecting Interaction Models

Ports allow an analyst to define a particular system as a graph of components or sub-systems connected through their ports. When generating a corresponding simulation model for such a component configuration, one needs to consider not only the simulation models for the individual components but also the models that capture the dynamics at the interaction points—the *interaction models*. For instance, the behavior of a system consisting of a car driving over a road is determined not only by the behavior of the car and the road individually, but also by the interaction model between the tires and the road (i.e. contact friction).

Often the component interaction models are trivial and correspond to Kirchhoff's voltage and current laws. For instance, most electrical connections can be modeled sufficiently accurately by setting the voltages equal (Kirchhoff's voltage law) and making the currents add up to zero (Kirchhoff's current law). Similarly, a rigid mechanical connection can be modeled by setting the velocities of the components equal and making the forces/torques add up to zero. In most object-oriented modeling languages, these trivial interaction models correspond to the default port-connections and can therefore be omitted (Mattsson and Elmqvist 1998). However, in general an algebraic or differential algebraic model or even a partial differential equation model is needed to describe the physical phenomena taking place at the point of interaction.

Unlike component models, interaction models have the property that their parameters cannot be encapsulated. The models of components depend only on parameters of the component itself (geometry, material properties, etc.) but do not depend on any parameters of other components or systems—all the parameter relationships are internal to the component object. Component *interaction models*, on the other hand, are not tied directly to a physical instantiation from which its parameters can be derived. Instead, the simulation parameters depend on the physical properties of the *two* interacting components. These components may be different for each instantiation of the interaction model. For example, a tire component has certain dynamic properties that depend on the physical parameters of the tire and the tire only (size, type of rubber, pressure, etc.) However, when this tire interacts with the road, the interaction model (friction) depends on the physical parameters of *both* the tire and the road surface.

To automate the process of instantiating interaction models, we somehow need to determine first which set of parametric models is appropriate for modeling a certain interaction. To establish a this association with an interaction model, we introduce a connection class.

A connection class defines which ports are connected, the type of the connections, and the possible simulation models that can be used for modeling such a connection. For instance, Figure 10 illustrates the definition of a revolute connection between a circular-hole-port and an axle-port. The connection specifies the interaction model that can be used to describe the revolute behavior. Note that this connection class is different from the circular-hole compatibility rule. The compatibility rule specifies whether the ports are compatible, while the connection class relates the port connection to the applicable simulation models for the interaction. The Modelica model class in Figure 10 associates the connection class with the underlying behavior model. The query and retrieval of interaction models from a model repository are still part of our ongoing research.

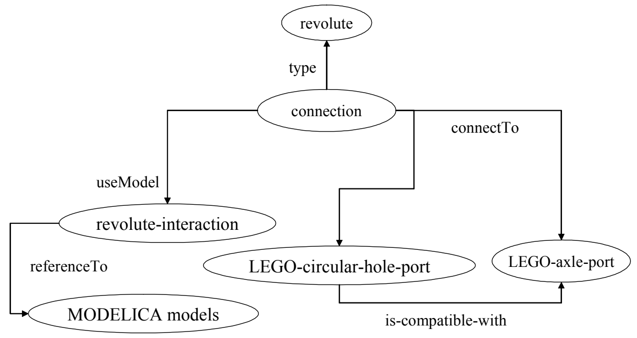

Figure 10: Definition of Revolute Connection Class and Possible Simulation Models

8 SUMMARY

This paper investigated the concept of ports as useful abstractions for system configuration and simulation. To take full advantage of ports, we introduced an ontology that can specify in an unambiguous, computer-interpretable fashion the attributes of and relationships between ports. The attributes are divided into three major categories: form, function, and behavior attributes. By combining these attributes in integrated port representations, one can represent and store diverse knowledge about ports in a knowledge base. This knowledge can then be used to reason about port connections and the corresponding simulation models. We illustrated this with examples for port compatibility checking and interaction model selection. This concludes our preliminary investigation of explicitly representing port concept for system configuration and simulation. The goal of our current research is to extend this work towards the automated composition of simulation models from port-based system configurations.

ACKNOWLEDGMENTS

This research was funded in part by the Pennsylvania Infrastructure Technology Alliance and by Bombardier Transportation Systems. Additional support was provided by the Institute for Complex Engineered Systems at Carnegie Mellon University, and the G.W. Woodruff School of Mechanical Engineering at Georgia Tech.

REFERENCES

Benson, M. and J. P. Terpenny 2001. A Survey of Methods and Approaches to Knowledge Management in the Product Development Environment. *21st ASME International Computers and Information in Engineering Conference (CIE)*, Pittsburgh, ASME.

Borgida, A., R. J. Brachman and D. L. McGuinness 1989. CLASSIC: A structural Data Model for Objects. *ACM SIGMOD International Conference on Management of Data*, Portland, Oregon.

Borst, P., H. Akkermans and J. Top 1997. Engineering Ontologies. *International Journal of Human-Computer Studies* 46: 365-406.

Bray, T., J. Paoli, C. M. Sperberg-McQueen, et al. 2000. *Extensible Markup Language (XML) 1.0 second edition.* World Wide Web Consortium. <http://www.w3.org/TR/REC-xml>

Counsell, J., I. Porter, D. Dawson, et al. 1999. Schemebuilder: Computer Aided Knowledge Based Design of Mechatronic Systems. *Assembly Automation* 19(2): 129-144.

Dean, M., D. Connolly, F. van Harmelen, et al. 2002. *Web Ontology Language (OWL) Reference Version 1.0.* W3C. <http://www.w3.org/TR/2002/owl-ref/>

Donini, F. M., M. Lenzerini, D. Nardi, et al. 1996. Reasoning in Description Logics, *Principles of Knowledge Representation.* G. Brewka. Stanford, CA, CSLI Publications: 191-236.

Duckering, B. C. 2000. Behavioral Modeling Technology: Leveraging Engineering Knowledge in CAD Models. *The Fourth IFIP Working Group 52 Workshop on*

Knowledge Intensive CAD (KIC-4),U. Cugini and M. Wozny, Parma, Italy.

Eastman, C. M. and N. Fereshetian 1994. Information Models for Use in Product Design: A Comparison. *Computer Aided Design* 26(7): 551-572.

Elmqvist, H. and S. E. Mattsson 1997. An introduction to the physical modeling language Modelica. *European Simulation Symposium*, Passau, Germany, Society for Computer Simulation.

Farquhar, A., R. Fikes and J. Rice 1997. The Ontolingua Server: A tool for collaborative ontology Construction. *International Journal of Human-Computer Studies* 46: 707-728.

Fensel, D. 2000. *Ontologies: silver bullet for knowledge management and electronic commerce* Berlin: Springer-Verlag.

Fensel, D., I. Horrocks, F. van Harmelen, et al. 2001. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems* 16(2): 38-45.

Fensel, D., F. van Harmelen and I. Horrocks (2003). OIL and DAML+OIL: Ontology Languages for the Semantic Web. *Towards the Semantic Web: Ontology-Driven Knowledge Management*. J. Davies, D. Fensel and F. van Harmelen. Hoboken, NJ, John Wiley & Sons**:** 5-28.

Haarslev, V. and R. Moller 2001. Description of the RACER system and its applications. *2001 International Description Logic Workshop (DL2001)*,D. L. M. Carole A. Goble, Ralf Möler Peter F. Patel-Schneider, Stanford, CA.

Hendler, J. and D. L. McGuinness 2000. The DARPA Agent Markup Language. I*EEE Intelligent Systems* 15(6): 67-73.

Hirtz, J., R. B. Stone, S. Szykman, et al. 2001. A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts. R*esearch in Engineering Design* 13(2): 65-82.

Horrocks, I. 1998. Using an expressive description logic: FaCT or fiction? *Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR'98),*A. G. Cohn, L. Schubert and S. C. Shapiro, San Francisco, CA, Morgan Kaufmann Publishers.

ISO 1994. 10303 *Industrial Automation Systems and Integration - Product Data Representation and Exchange.* International Organization for Standardization. <www.iso.ch/cate/cat.html>

Iwasaki, Y., A. Farquhar, R. Fikes, et al. 1997. *A Web-Based Compositional Modeling System for Sharing of Physical Knowledge*. KSL-98-17, Stanford, California, Stanford University.

Kopena, J., M. D. Peysakhov and W. C. Regli 2002. Extensible Semantics for Representing Electromechanical Assemblies. Philadelphia, PA, Drexel University, Department of Mathematics and Computer Science.

Liu, J. 2001. *Responsible Frameworks for Heterogeneous Modeling and Design of Embedded Systems.* Ph.D. Thesis, Electrical Engineering and Computer Science, Berkeley, California, University of California at Berkeley.

Mattsson, S. E. and H. Elmqvist 1998. An overview of the modeling language Modelica. E*urosim '98 Simulation congress,* Helsinki, Finland.

Mentor Graphics 2000. *IC-Design Suite.* Wilsonville, OR, Mentor Graphics.

Sinha, R., C. J. J. Paredis and P. K. Khosla 2001. Interaction modeling in configuration design. 2*001 ASME Design Engineering Technical conferences,* Pittsburgh, PA, ASME.

Stone, R. B. and K. L. Wood 2000. Development of a Functional basis for design. J*ournal of Mechanical Design* 122(4): 359-370.

Susca, L., F. Mandorli, C. Rizzi, et al. 2000. Racing Car Design Using Knowledge Aided Engineering. A*rtificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM)* 14: 235-249.

Szykman, S., J. W. Racz and R. D. Sriram 1999. The Representation of Function in Computer-Based Design. 1*999 ASME Design Engineering Technical Conferences - Design Theory and Methodology,* Las Vegas, Nevada, ASME.

Szykman, S., R. D. Sriram, C. Bochenek, et al. 2000. Design Repositories: Engineering Design's New Knowledge Base. I*EEE Intelligent Systems* 15(3): 48-55.

Wood, W. H. and A. M. Agogino 1996. A Case-based conceptual design information server for concurrent engineering. C*omputer-Aided Design* 28(5): 361-369.

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

AUTHOR BIOGRAPHIES

VEI-CHUNG LIANG is a Ph.D. Candidate in the Department of Civil and Environmental Engineering at Carnegie Mellon University. In 1992, He graduated from National Central University, Taiwan. He received his M.S. in Structural Engineering from Stanford University in 1996 and went to Carnegie Mellon University where, in 1998, he received his M.S. in Computer Aided Engineering. His research interests include computer aided engineering and design, knowledge management in engineering, simulation-based design, and system engineering. `<vliang@cs.cmu.edu>`.

CHRISTIAAN J. J. PAREDIS is currently an Assistant Professor in the School of Mechanical Engineering at Georgia Tech. He 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. Until 2002 he was research faculty at the Institute for Complex Engineered Systems at Carnegie Mellon University. Dr. Paredis has a broad multidisciplinary background. His research combines aspects of information technology, simulation, and modularity to support the design of mechatronic systems. The goal of his current research is to develop an integrated IT framework for product development that encompasses both representations and tools across the different design activities, from requirements definition and synthesis, to analysis, interpretation, and visualization. He is also still active in the robotics areas in which he did his Ph.D. research. <chris.paredis@me.gatech.edu>.