A Port Ontology for Conceptual Design of Systems

Vei-Chung Liang
Email: vliang@cs.cmu.edu
Institute for Complex Engineered Systems,
Carnegie Mellon University,
Pittsburgh, PA 15213

Christiaan J. J. Paredis
Email: chris.paredis@me.gatech.edu
Systems Realization Laboratory,
G.W. Woodruff School of Mechanical Engineering,
Georgia Institute of Technology,
Atlanta, GA 30332

During conceptual design of systems, the emphasis is on generating the system architecture: the configuration of sub-systems and the interactions between them. Ports, as locations of intended interaction, play an important role at this stage of design. They are convenient abstractions for representing the intended exchange of signals, energy or material; they can be applied at different levels of detail, across different energy domains, and to all aspects of design: form, function, and behavior. But to play this versatile role, ports need to be represented in an unambiguous yet flexible fashion, accommodating the differences in vocabulary and approach across different disciplines and perspectives. In this article, we introduce the semantic structure for such an unambiguous representation: a port ontology. The ontology formalizes the conceptualization of ports such that engineers and computer aided design applications can reason about component connections and interactions in system configuration. It defines ports in terms of form, function and behavior attributes and further includes axioms that can be used to support a variety of engineering design tasks, such as port refinement, port compatibility checking, and the instantiation of interaction models. A LEGO example is used to illustrate the ontology and its applications in conceptual design. [DOI: 10.1115/1.1778191]

Introduction

Ports are defined as locations of intended interaction between a component and its environment. Together they constitute the interface of a component, and define its boundary in a system configuration. As illustrated in Fig. 1, the conceptual design of a windshield wiper system can be represented as a configuration of components or sub-systems that interact with each other through well-defined interfaces. The port connections represent interactions consisting of the exchange of energy, matter or signals (information). For instance, the configuration interface of the motor in Fig. 1 has ports for the stator, the shaft of the rotor, and the electrical connectors. The interactions between components (subsystems) are indicated in the graph by the connections between ports.

Representing design alternatives as configurations of port-based objects is useful at the conceptual design stage when the geometry and spatial layout is still ill-defined. At this stage of design, the emphasis is on generating the system architecture, and this architecture can be captured conveniently as a (hierarchical) configuration of port-based interfaces. As an example, consider a satellite system which is typically composed of a propulsion system, an electrical power system, a thermal management system, a control system, a communication system, and the mechanical structure connecting all the other systems [1]. Designers rarely deviate from this top-level decomposition (partly due to the alignment between subsystems and human organizational structures); however, for each subsystem, there exists a large variety of solutions and/or further decompositions. A graph representation is an important tool for a systematic (possibly automated) exploration of this multitude of design alternatives.

Representing systems as port-based objects is particularly useful in the context of collaborative design [2]. Design problems are often decoupled by discipline into multiple problems solved separately. However, the decoupling is rarely complete so that information regarding other subsystems needs to be exchanged, updated, and integrated frequently throughout the design process. An explicit, formal interface between decoupled subsystems, as provided by ports, allows information management systems to propagate design changes to the other connected subsystems efficiently.

Although there has been previous work on the development of comprehensive design representations that recognize the decomposition of artifacts from a functional and systems perspective [3–5], there has been little attention paid to the representation of ports. The goal of this article is to provide an unambiguous representation framework in which design decisions about component interactions can be captured at the conceptual design stage, and in which knowledge about system composition can be expressed to support computer reasoning. This leads to the following two questions: Which types of knowledge and information about component interactions should be captured to support system design? And how should this knowledge and information be formalized?

As has been previously recognized, design knowledge can be represented in terms of the three complementary aspects: function, behavior and form (or structure) [6]. These three aspects also form the foundation of the port ontology introduced in this article. They are formalized through a set of attributes and relations between them. By combining these attributes, a virtually infinite set of ports can be defined. In addition, the port ontology includes axioms to support compatibility checking and the instantiation of interaction models. A LEGO example is used to illustrate these applications of the port ontology.

Past Work on Ports in Engineering Design

The idea of including the port concept in conceptual design modeling is not new. Early work by Roth [7] identified different types of mechanical connections (with zero or more degrees-of-freedom). He recognized that mechanical connections always result from parts in contact through sets of surfaces: *Wirkflächenpaarungen*—the equivalent of ports. Horváth et al. [8] further developed these ideas and classified port concepts used in conceptual design into external, internal, concrete and abstract ports. The external ports denote the energy, material or signal interfaces between a component and its environment. The internal ports, on the other hand, refer to the interfaces of the *sub*-components—they are internal to a component. Abstract ports represent the interfaces of abstract design entities, while concrete ports represent the interfaces of design entities with specified geometries. In related research [9], various situational relations be-

Contributed by the Engineering Informatics (EIX) Committee for publication in the JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING. Manuscript received May 25, 2003; revised June 1, 2004. Associate Editor: S. Szykman.

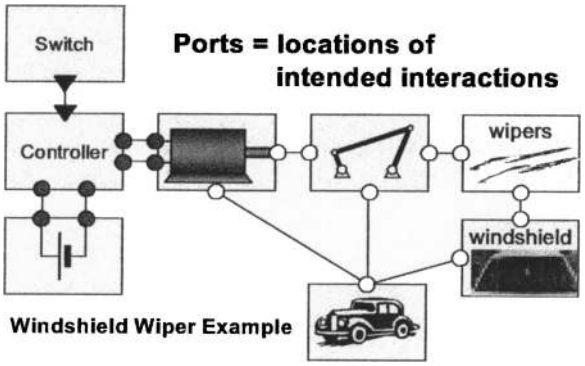

Fig. 1 A system represented as a configuration of configuration interfaces.

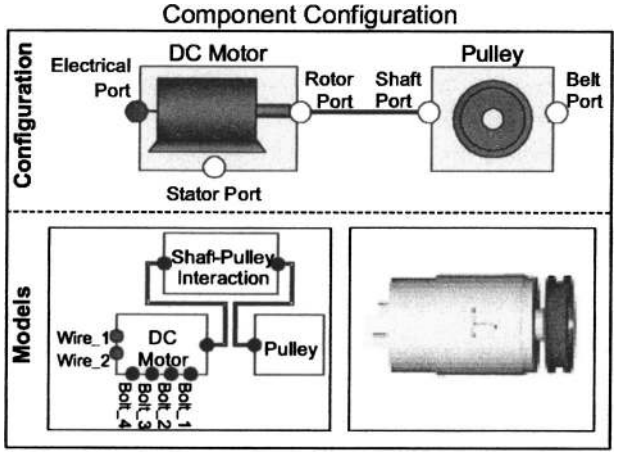

Fig. 2 An integrated product representation specifies the composition of a DC motor and a pulley.

tween design entities are also formalized into seven ''situation views,'' such as morphology, relative position and structure. In this article, these situation views are folded into the port ontology as constraints on attributes of connected ports. Ports are also used in Schemebuilder [10], where they formalize the connections of elements. They are classified statically as either power, material, or information ports, each with a specified flow direction. No other characteristics, such as geometry or intended use, are considered. Singh and Bettig [11], on the other hand, focus primarily on the geometric aspect of ports in the context of component assembly. They provide conditions for compatibility and connectability and suggest several static labeling schemes for ports.

In this article, multiple perspectives (form, function, and behavior) are combined in a comprehensive view of ports. This work builds on earlier contributions by Sinha [12] who defined ports as locations of intended interaction between a component and its environment. Sinha separated interaction concepts from component connections at the behavioral model level. The type of interaction between two components is determined based on the component's port attributes. Instead of using statically defined port types, Sinha used attribute grammars to provide flexibility in defining and retrieving components with compatible port information (e.g. intended use, material, and CAD features). From a set of primitive attributes, ports in any system and all application domains can be represented by refining the primitives into domain-specific concepts. In this article, we provide a *formal definition* of ports and their attribute structures as a foundation for supporting such domain-specific refinements.

Why Ports?

Ports are more than just a convenient abstraction for representing component interactions in system configuration. In this article, ports are formalized to represent the functional, behavioral, and structural, relationships between components. Although the concepts of form, function and behavior are commonly used in engineering design, in the context of ports, these concepts have a much more restricted scope: *interactions between components*. With respect to the representation of function, ports correspond to *flows*—a change in material, energy or signal over time [13]. The notion of exchange of material, energy or signals is also used in *behavioral* modeling. For instance, the bonds in bond-graph modeling represent energy flow [14]. When connecting two components, the energy, material, or signals flowing out of one component must flow into the other component, resulting in conservation models. Connecting two components also imposes constraints on the *form* of the ports. For instance, the choice of a ''120V AC plug'' for one component requires a very specific geometry of the mating component, the outlet.

Formalizing functional, and behavioral and structural aspects of ports is needed to support composition or configuration in systems design. Teams of designers working on the design of the individual components of a system need to communicate clearly and unambiguously which decisions they have made about the component interaction. At the early stages of design, such decisions may be limited to functional specifications, but as the design process progresses, more and more geometric specifications will be used. To maintain as much flexibility as possible in future design decisions, it may be desirable for the designers to specify only those port characteristics that impact their design decisions about the internals of the component. Yet, even such partial port specifications need to satisfy certain compatibility constraints. A formal representation of ports would allow design teams to communicate their decisions with each other and verify whether their decisions remain compatible.

A second important reason for formalizing ports during conceptual design is that in formal port representations, *associations* are stablished between form, function, and behavior. These associations can then be used to support a variety of *composition tasks*. For instance, in Fig. 2, each configuration interface is linked through port associations to a container of behavioral models. These integrated product representations may also include functional and geometric models, and the associations between them. The associations between the ports of the configuration interface and the ports of the corresponding behavioral models are often, but not always, one-to one mappings. For instance, the DC motor has three configuration ports, two for transferring mechanical energy and one for electrical energy. The configuration port for the shaft of the DC motor corresponds in the behavioral view to a single port conveying rotational mechanical energy, but the configuration port for the electrical connection is modeled as two electrical behavioral ports, one for each wire. Such associations are established by explicitly specifying the relations among form, function, and behavior of the components in the design representation.

As a result of the associations between configuration interfaces and the corresponding simulation models, the definition of a system architecture as a composition of component objects provides the necessary information to compose the corresponding behavioral models into a system-level model [12]. As is shown in Fig. 2, the design configuration consisting of the pulley mounted onto the motor shaft is represented by a motor object and a pulley object connected through their rotor and shaft ports, respectively. When composing the components at the configuration level, one has to consider not only the composition of the underlying simulation models of the components but also the interaction models that describe the interactions between components [12]. For example, in Fig. 2, when the rotor port is connected to the shaft port, the connection of their simulation models is not a straightforward connection. The corresponding system simulation model is ob-

tained by connecting the ports of the simulation models of the motor and pulley through the rotor-pulley interaction model.

Having established the importance of *ports* in the context of systems design, consider now the importance of defining a port *ontology*. In order for product models to be useful for knowledge representation, the information encoded in the models needs to be unambiguously understood by all processing agents (human as well as computer agents), 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. The representation of ports is particularly susceptible to such ambiguity because ports reach across disciplines, each with their own concepts and vocabulary. To eliminate such ambiguity, a common port ontology is needed to provide application developers a common vocabulary to describe their ports. However, the implementation of such ontology is not straightforward. One has to consider carefully which concepts to include in the ontology and how to represent them.

Related Work on Design Ontologies

There are two common approaches to support unambiguous computable representations: labels and metadata. These two approaches provide different levels of using symbols in the representation. The label approach considers only the semiotic or syntactic consistency among concepts. The metadata approach provides further meanings to the symbols that are used in the representation. 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 they are easy to create by the designers. However, the labeling approach does not include semantics in its representation and therefore requires extra effort to sort and retrieve synonyms and maintain relations among the same terms used for different design concepts.

The metadata approach, on the other hand, assigns primitive and compound attributes to the terms used to define concepts. This makes it more general than the labeling approach since it compares not only the labels of the concepts but also the semantics represented by the attributes. For example, a hinge can be defined as a connection that does not resist an 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.

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. In the past decade, syntactic metadata definition languages such as the Extensible Markup Language (XML) [15] or EXPRESS [16] have been developed and used to represent design concepts and design products. While XML provides an important solution for making Internet information computer interpretable, it has by itself limited expressiveness for describing the relationships between concepts, and is therefore not suited for representing ontologies. 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. Even a simple change of a tag label requires changing the code of all the parsers.

These syntax ambiguities of the metadata can be eliminated by introducing semantic metadata as defined in an ontology. The ontology is the content theory that specifies the concepts and their relations used in a specific knowledge domain (domain of discourse) [17]. Recently, more expressive languages than EXPRESS and XML have been defined in support of ontologies. Constructs in the ontology language are associated with semantic models so that they can be used to infer the meanings of the relationships and support unambiguous data exchange and definitions.

Ontologies consist of both concept taxonomies and axioms, allowing computer programs to share, exchange, extend, reuse and translate information and knowledge within the domain of discourse. The representations can be based on either frame-based languages or description logic languages [18]. In frame-based languages such as Ontolingua [19], the knowledge domain is described using frames and slots. The frames and slots are similar to the classes and attributes in object-oriented modeling except that the slots can be defined outside the definition of the frames, providing additional options to refine and specify the attributes. Similar to the frames and slots defined in the frame-based languages, the description logic languages use concepts and roles as the two basic elements to define a knowledge domain. While frame-based languages are usually supported by first-order logic reasoners, description logic languages only use a subset of the first-order logic constructs. This subset remains highly expressive but is at the same time decidable [20], a desirable property that is lost in first-order logic. Representative description logic systems include CLASSIC [21], FaCT [22], and RACER [23].

Recently, two ontology languages, the DARPA Agent Markup Language (DAML) [24] and DAML+OIL (Ontology Inference Layer) [25,26] have been merged and extended into the Web Ontology Language (OWL) as a W3C working draft [27]. 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 [28,29]. Only a few research efforts are underway in the engineering design area.

Two notable research efforts in design ontologies are taking place at the Delft University of Technology and at Osaka University. In Delft, Horváth and co-authors [9,30] have defined a general ontology for design concepts and proposed a nucleus-based conceptualization to describe the interactions between design concepts. A nucleus describes the interactions between two objects or devices as a set of connected surface regions. The nucleus concept applies primarily to the mechanical domain. From what we have described in the previous section, we envision that the ports and their connections can be used in multiple domains. As compared to the nucleus concept, the ports can be thought of as the interacting surface regions while the connections correspond to the nuclei. One important difference in approach is that, in the nucleus based approach, the behavior models are defined in the design concepts rather than in the nuclei.

In Osaka, Kitamura and Mizoguchi [5,31] have proposed an extended device ontology to describe artifacts, based on intended use, as compositions of devices which process input and produce output. This device ontology is part of a research effort for developing a functional concept ontology in support of function-based design. In relation to ports and interactions, the authors describe device connections in terms of conduits, input/output ports, operands, mediums or objects. When used in mechanical systems, a conduit is a physical component that conveys or transfers flows from one device to the other in an ideal fashion (e.g. a shaft, wire, or pipe). A conduit is virtual when it is used to describe the contact conditions (point, line, or surface) between mechanical elements. No detailed geometrical and functional attributes are used to describe the ports—their purpose is solely as a placeholder for input and output flows (operands, mediums, or objects). The port ontology introduced in this article can be considered as another conceptualization of component interactions. However, it treats connections as virtual by default and defines ports in terms of form, function, and behavior attributes. This provides more flexibility and extensibility in support of many design tasks in conceptual design, such as component composition and component refinement, as is illustrated later in this article.

Design and Definition of the Port Ontology

As pointed out earlier, ports are characterized by form, function and behavior aspects. In this section, these aspects and the rela-

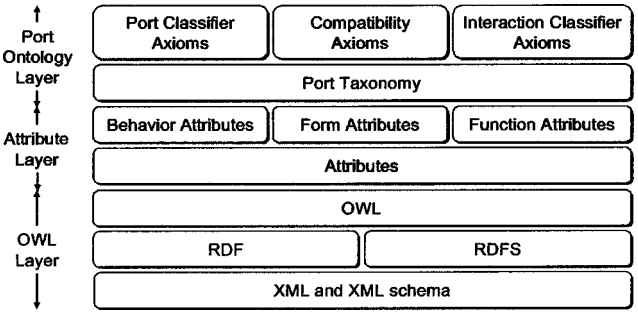

Fig. 3 Main Structure of the port ontology.

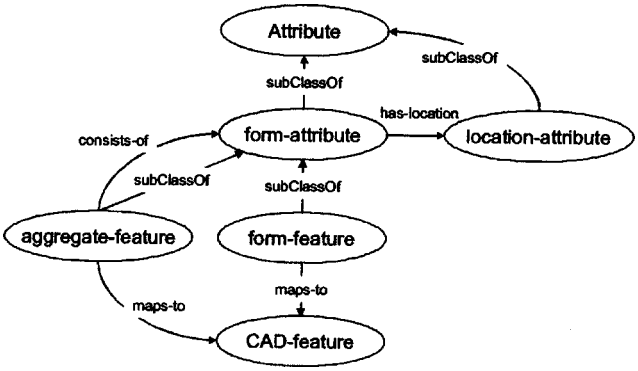

Fig. 4 Relations among form-attributes.

tions between them are investigated more closely and their representation is formalized in classes, properties and axioms.

The Structure of the Port Ontology. The main focus of the port ontology is on port classes. The port classes define, at different levels of abstraction, the types of ports that can be used in systems design. These ports are also part of the ontology for artifacts, in which not only the interface ports but also the internal characteristics of components and sub-systems are defined. Although there is bound to be a strong connection and some overlap between a port ontology and an artifact ontology, it is still meaningful to capture the specific characteristics of ports as a separate ontology.

In addition to port classes, the port ontology also contains classes for *attributes* of ports. Instead of having ports directly defined by the semantic constructs, an intermediate attribute layer is defined in between the OWL constructs and port classes. The resulting structure of the port ontology consists then of three layers: the ontology layer, the concept layer, and the OWL layer (Fig. 3). The port ontology layer and attribute layer will be described in detail later in this section. The OWL layer is an implementation layer, which may be replaced by other implementations such as native description logic languages or restricted frame-based representations. OWL is based on the Resource Definition Framework and Schema (RDF/RDFS), which in turn is serialized in XML syntax. RDF and RDFS provide a simple semantic model for interpreting customized tags. It allows one to define inheritance relationships for both classes (resources in RDF) and properties (any generic association between concepts). OWL further extends the expressiveness of RDF/RDFS by including constraints.

The use of attributes to define ports allows one to formulate knowledge about the ports in more general axioms. For instance, rather than describing compatibility between any two ports directly, one could generalize the knowledge by defining it in terms of the form attributes of the ports. The same axiom then applies to all the ports that share these particular form attributes.

The introduction of the attribute layer also allows the ontology to be extended more easily. It is not possible to have a complete set of class definitions for conceptual design—engineers may need to add new attributes to define new design concepts. If the design concepts were defined using fixed class schemas, the engineers would have to use an ad hoc attribute list to attach new design information, as is for instance common practice in IFC (Industry Foundation Classes) [32]. The problem with such a generic attribute list is that it lacks clear semantics—it does not allow one to establish (semantic) relationships between attributes. By forcing a designer to define new attributes as classes and insert them into the attribute layer before using them, it is easier to maintain the consistency of the ontology (e.g., one can establish equivalence relationships between identical attributes with different names) and extend the previously defined axioms (e.g., an existing axiom that refers to all the children of a given attribute can be automatically extended by adding a new child attribute).

The set of all port attributes has been divided into three main categories—each represented by a top-level abstract class: *form-attribute, function-attribute* and *behavior-attribute*.

Form Attribute Classes. Form attributes describe all characteristics of ports that relate to geometry. Although during conceptual design, the complete geometry of a port is not yet determined, it is still important to consider some geometric aspects. For example, a port transferring rotational mechanical energy can be characterized already during conceptual design by the rotation axis of the port. This attribute can then be used to specify that the rotation axis must be aligned with the rotation axis of the connecting port. In addition to supporting the definition of partial geometry during conceptual design, the ontology should provide sufficient flexibility to refine this partial geometry incrementally into a fully defined detailed design.

To achieve these goals, the ontology includes the following sub-classes of *attribute*, as shown in Fig. 4: *form-attribute, location-attribute, form-feature*, and *aggregate-feature*. The base class, from which all geometric attributes derive, is *form-attribute*. Each form attribute has a *location-attribute* that indicates the position and orientation of the form attribute relative to its parent in an aggregation hierarchy. The top of this hierarchy is the component of which the port is part. Each port can then have multiple form-attributes for which the location is defined relative to the reference frame of the component. If one of these *form-attributes* is an *aggregate-feature*, then the locations of all the children in the aggregation are referenced relative to the location of the parent, i.e., the *aggregate-feature*, rather than the component. The location may not be fully defined until embodiment design—partially defined or even undefined *location-attributes* are allowed.

The intended interaction region of a port can be defined in terms of primitive geometric *form-features*, such as points, curves, and surfaces. In addition, more complex form features can be defined as aggregations of *form-features*. Both simple and aggregate form features can be mapped to existing CAD features [33,34]. These CAD features often define commonly used geometric concepts such as cylinders or holes—possibly including chamfers, rounds, etc.

Form features and aggregate features differ from CAD features in that they may represent complex geometries without specifying all the geometric details. Such features are especially useful during conceptual design because they allow the designer to establish the associations between the form of a port and its function and behavior without having to define all the detailed geometry. One class of such incompletely defined form features are standardized features that can be unambiguously referred to by a single label, e.g., a 120V AC plug. (This is the type of form features used in the LEGO example in the next section.) Even though the detailed geometry is not specified explicitly, it is implied in the use of the standardized label. As a result, one can define compatibility axi-

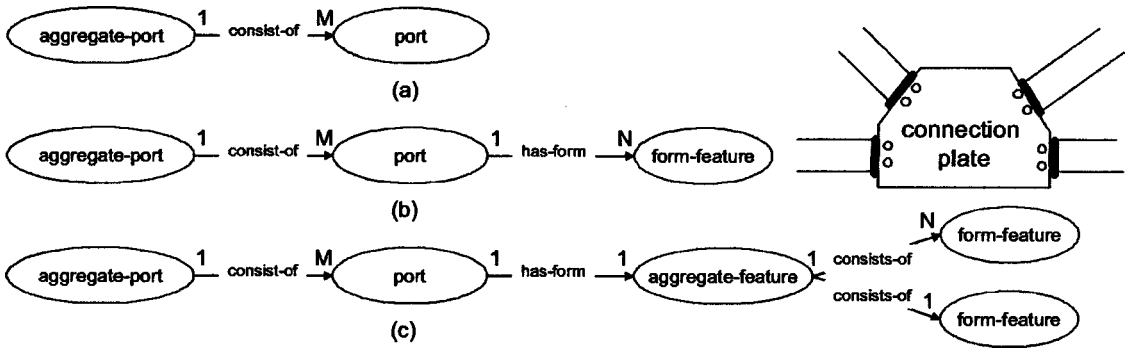

Fig. 5 Three possible cases of relations between form attributes and ports. (The numbers by the arcs represent cardinality or multiplicity.)

oms that refer directly to the labels rather than to the underlying geometry. For instance, a standard 120V AC plug is compatible only with a standard 120V AC outlet.

A second class of incompletely defined form features consists of parameterized families of partially defined port geometry. For example, a heat sink port may have form features indicating its size as a bounding box and its total surface area, but without specifying the number and exact geometry of the fins. This allows one to associate the surface area specification with a corresponding behavioral parameter, or use the bounding box for geometric interference checking. Later, during detail design, the partially defined form features can be mapped to fully specified CAD descriptions.

Based on these basic concepts, there exist multiple methods for describing a single port-attribute relation. Depending on the design stage, one method may be preferred over another. Consider a scenario in which an engineer is designing a connection plate that connects four beams, as is illustrated in Fig. 5. The scenario starts at (a) where the connection plate has an aggregate port with four port elements (M=4). When the engineer decides to use bolt connections for each individual port, he expands the port definition with multiple form features, one for each of the bolts (b); however, the number of bolts, N, may not be decided until the detail design stage. To increase the connection's resistance to moments, the engineers then decide to combine the bolt connection with welds, which expands the port declaration into (c) where each port is associated with an aggregate-feature (weld and bolt form features). Only during detail design are these form features further defined into complete, CAD-based geometry specifications.

Function Attribute Classes. Function attributes describe the intended use of a port. Functions have been researched extensively, and we therefore leverage the concepts defined by others. In [35], product functions are specified as "verb-object" pairs, in which the function corresponds to the active "verb," and the flow corresponds to the "object." (Note: The grammatical "objects" in this paragraph are not to be confused with the object-oriented "objects" in the rest of the paper.) The flow is thus the recipient of the function's operation. For example, an electric motor *converts* (verb) an *electrical energy flow* (object) into a *mechanical energy flow* (object). Since ports are specifically intended to facilitate the interaction between a component and its environment, the functions related to ports are limited to interaction functions such as "transfer" (of energy, material, or signals), "connect" (join or link), or "support" (secure and position). A (partial) graph of function attributes for ports is shown in Fig. 6. Multiple of these function attributes can be combined to describe a single port.

We consider the definition of any "energy flow" to encompass the static case in which no energy is actually flowing (e.g., the "join" or "link" interactions identified above). Each energy flow can be represented by power conjugate complements: effort and flow variables [35]. For static interactions, the effort variable is

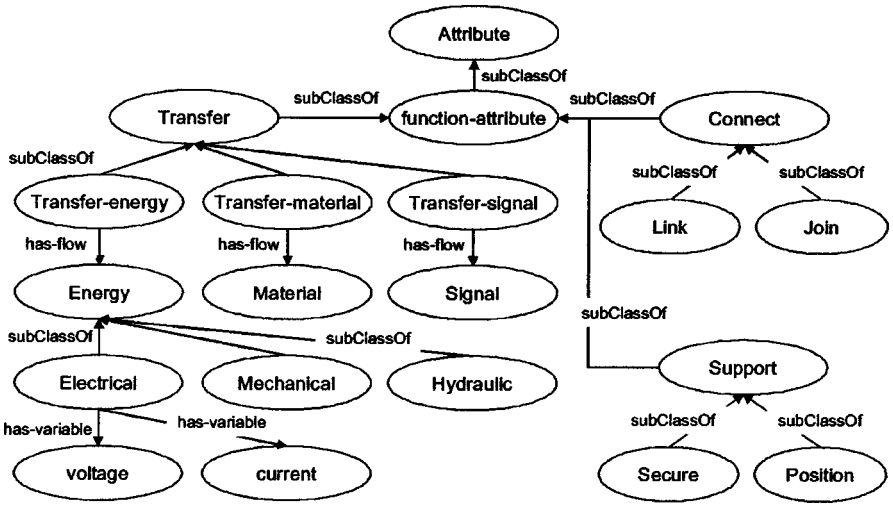

Fig. 6 Function attributes used in the port ontology.

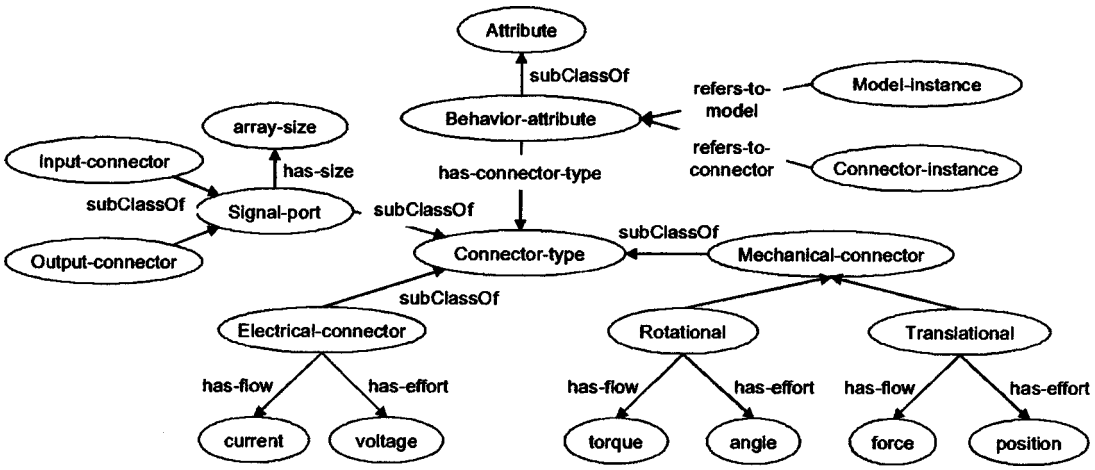

Fig. 7 Partial behavior attributes used in the port ontology.

zero (resulting in zero energy flow), but the flow variable is nonzero so that the interaction—the transfer of effort—is still adequately characterized.

As in [35], ''transfer'' functions are associated with flows of energy, material, or signal. Using this definition, the ''transfer'' function attribute can be refined into ''transfer-energy,'' ''transfer-material,'' or ''transfer-signal.'' The energy flows can be further refined as electrical, mechanical, and hydraulic, etc. for which there exist corresponding energy conjugate variables, such as ''voltage'' and ''current.''

The ''connect'' function has two child functions: ''link'' and ''join.'' The ''join'' function connects two ports in a predetermined manner while the ''link'' function connects two ports through an intermediary flow [35]. In the context of port connections, the join function connects two ports directly, while the link function connects two ports through an intermediary artifact or model. For example, the shaft-port of the motor is joined directly to the pulley-port, while two beams are linked using welds or fasteners. The distinction between these two child functions can thus be used to determine if any additional models or artifacts are needed for the connection.

The purpose of defining the semantics of port functions in a detailed fashion is to enforce compatibility and guide the selection of appropriate (and compatible) form features. For a connection between two ports to be compatible it is necessary that the ports have the same function and that, for ''transfer'' ports, also the flow types are the same. As will be shown later in the LEGO example, a detailed functional description of a port can also guide the selection of form features stored in a design repository.

Behavior Attribute Classes. Ports can also refer to behavior. Since the intended behavior of ports is to transfer energy, material or signals (see section on function attribute classes), their behavior can be described by a set of two or more variables characterized as either *flow* or *effort*, similar to the power conjugate complements identified in [35]. However, unlike the function and form characteristics of ports, the behavior is not an intrinsic characteristic of the port. The amount of energy or material transferred through a port is not determined solely by the behavior of the port, but by the behavior of the entire system of which the port is part. To determine the values of a port's flow and effort quantities, it is necessary to link together the behavioral models of all the components of the entire system and its environment, and to evaluate (i.e., simulate) this combined model.

This is where the behavior attributes in the ontology play a role. For each component, one needs to identify which variables in its behavioral model correspond to the exchange of energy, material or signals through a given port. These variables are the interface of the component's behavioral model. Recent object-oriented modeling languages, such as VHDL-AMS [36] or Modelica [37], define modular simulation models with interfaces consisting of connectors (or ports). By associating these connectors with the corresponding ports through behavior attributes, one can derive from the port connections which model connections to establish. This will be illustrated in the LEGO example.

As in bond-graphs [14], the model connectors represent not a single variable but a *set* of flow and effort variables. When introducing a connection, the modeling software will automatically introduce equations that correspond to Kirchhoff's voltage and current laws, indicating that the effort variables are equal and flow variables add up to zero. In the mechanical domain, there are two conflicting conventions in use for effort and flow. In bond-graph modeling [14], flow corresponds to velocity and effort to force; in object-oriented simulation modeling (Modelica, VHDL-AMS), the opposite convention is used (flow→force; effort→velocity). Both conventions are acceptable as long as they are applied consistently. We use the flow→force convention because it maintains the property that, in a connection, the effort variables are equal and the flow variables add up to zero [38].

Figure 7 illustrates a taxonomy of behavior attributes based on the Modelica standard library [39]. Each behavior attribute refers to a specific behavioral model for the component of which the port is part. Within this model instance, the behavior attribute refers to a specific connector. The behavior attribute also carries the type of that connector so that the type compatibility can be verified when establishing connections. Even when a modeling language other than Modelica is used a similar type taxonomy and connector to port mapping can be established. Often the types of connectors are identical but have been given different names. Within OWL, such classes can be related to each other through the property *owl:equivalentClass*.

Port-Attribute Properties. In addition to concepts, expressed as classes, ontologies also contain relationships, expressed as properties. Several relations between attributes have already been introduced in the previous section (Figs. 4–7). In general, relationships between ports and attributes can be expressed in ⟨S P O⟩ triple form, where S stands for the subject domain, P for the predicate (relation), and O for the object range. Thus, ⟨port has-attribute attribute⟩ denotes a relation between a port and an attribute.

However, it would be confusing to use the same OWL property, ''has-attribute,'' to define the relationships between all possible ports and attributes. The semantics of the relationships can be captured more precisely using the *owl:subPropertyOf* construct. Specific categories of relationships, such as *has-function, has-*

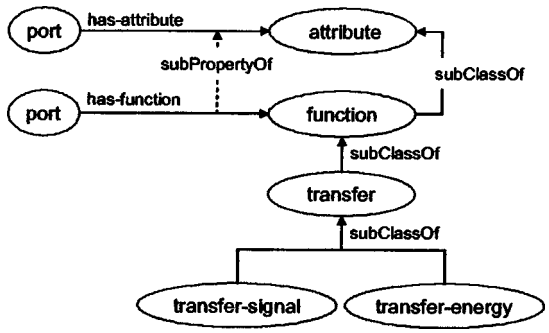

Fig. 8 An illustration of relationships between ports and attributes.

form-feature, and *has-behavior-connection*, can be accurately labeled as sub-properties of general *has-attribute* relationship. As illustrated in Fig. 8, the *has-function* property can be used to refer to any function attribute or its sub-classes, such as *transfer* or *transfer-signal*.

Capturing Knowledge in the Port Ontology: A LEGO Example

This section illustrates how the basic port ontology introduced in the previous sections can be used and extended for a specific application domain: LEGO systems. The focus is specifically on capturing design knowledge about ports in port axioms. Although the design of LEGO systems may at first seem like a toy problem, many of the issues which designers are facing during conceptual design also exist for LEGO systems. This example therefore provides a good illustration for how information and knowledge about ports for a particular application domain can be represented in the port ontology. It demonstrates first how the port ontology can be used and extended to represent the *semantic knowledge* for LEGO ports, for instance, by introducing specific LEGO form features. Secondly, it demonstrates how a variety of knowledge axioms can be defined, for instance, to verify the compatibility of port connections, and to support the instantiation of behavioral models for component interactions.

Extending the Port Ontology for LEGO Ports. The ontology described in the previous sections provides a basic set of concepts, attributes and properties for defining ports in general. This set can be expanded to include abstractions of specific port types in a particular application domain. For instance, certain combinations of geometric features that are standardized or occur frequently in a given application domain can be abstracted as a specific aggregate form feature. Within the LEGO domain, all ports are standardized with fixed dimensions and compatible shapes; it therefore makes sense to define a specific form feature for each type of geometry. Some of the common ports that have been identified are illustrated in Fig. 9.

Each of the ports in Fig. 9 can be described not only by a form feature, but also by an entire graph of attributes and properties. An example of an attribute graph for the axle port is shown in Fig. 10. The axle port contains a form-feature attribute, two function attributes, and a behavior attribute. Once defined, the new port, ''LEGO-Axle-Port,'' can be used to define additional LEGO-specific axioms, for instance, for port compatibility checking. This definition can also be saved in a LEGO design repository for reuse in the future. In the next section, it is illustrated how a designer can take advantage of such a design repository combined with the knowledge in the LEGO port ontology.

Supporting Design Refinement Through Subsumption. Consider the design of a windshield wiper system as introduced at the beginning of this article (Fig. 1). At this early stage of design, the ports are merely placeholders—an indication that there is a need for the transfer of energy, material or signals between components. During the design process, as the designer makes additional decisions about the components and their interactions, these initial placeholders will be gradually transformed into specific port definitions. In terms of the port ontology, the incremental decisions of the designer will result in the addition of attributes to the port definitions, the sub-classing or refinement of attributes, or the addition of constraints on the attribute values. For instance, in Fig. 11, the shaft port of the motor is incrementally refined from a generic port for energy transfer to a port in the shape of a LEGO axle for the transfer of rotational mechanical energy.

The port ontology plays an important role in this incremental refinement process: it provides the designer with the knowledge about valid refinement operations. For instance, the ontology can provide the designer with a list of valid refinements for the function ''transfer energy'' based on the knowledge that the abstract type energy can be refined into one of many specific energy types (Fig. 11): ''energy'' → ''mechanical energy'' → ''rotational mechanical energy.'' In combination with a design repository, the port ontology can also be used to suggest specific solution *instances*. For example, the transfer of rotational mechanical energy can be implemented as a LEGO axle port.

The suggestions for refinement and for specific instances can be obtained through a process called *subsumption*. In general terms, subsumption is a deduction mechanism for deciding whether one

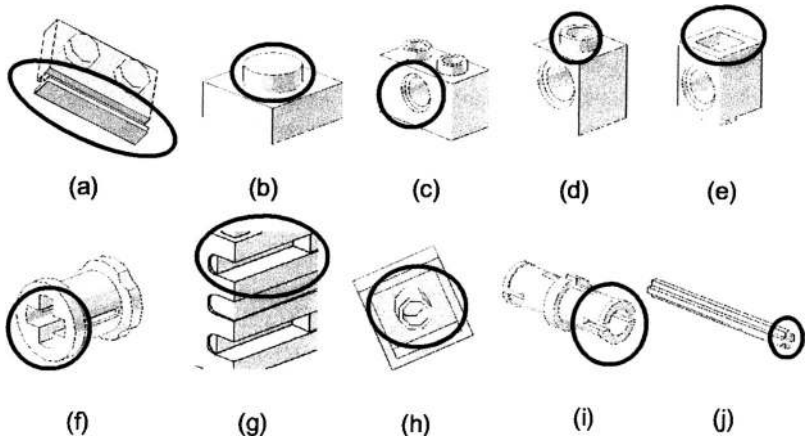

Fig. 9 The circled areas indicate LEGO-ports: (*a*) rail-port, (*b*) stud-port, (*c*) circular-hole-port, (*d*) TECHNIC-stud-port, (*e*) TECHNIC-tube-port, (*f*) axle-hole-port, (*g*) channel-port, (*h*) tube-port, (*i*) friction-pin-port, and (*j*) axle-port.

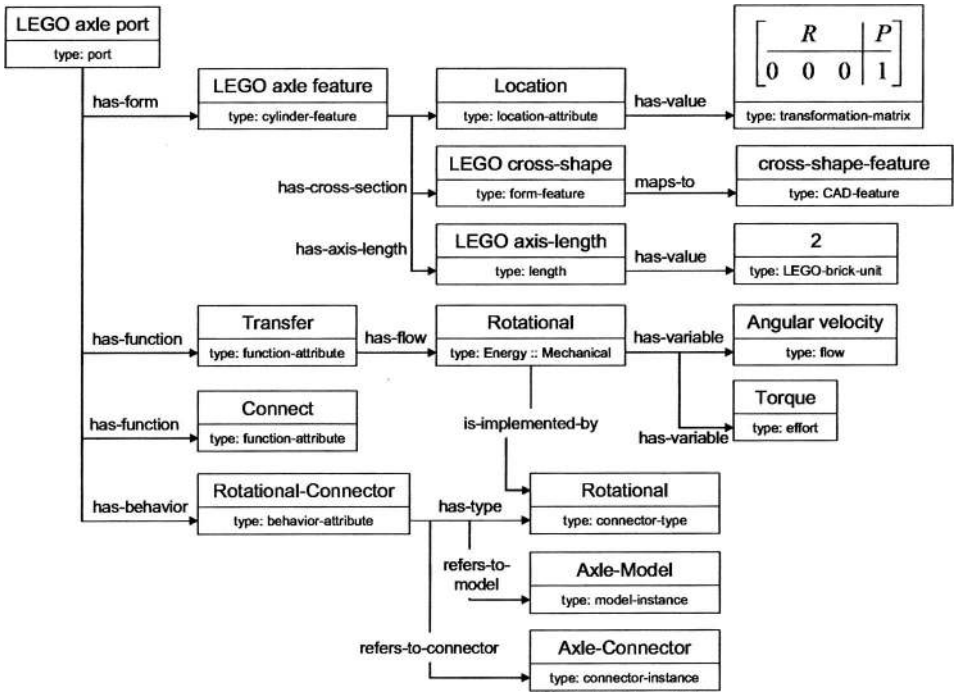

Fig. 10 An attribute graph for a LEGO axle port.

description, D1, is more general than another, D2; that is, whether D2 logically implies D1 [40]. It is the predominant deduction mechanism in Description Logics (DL) [41]. In this case, subsumption can be used to determine parent-child relationships between ports based on their attributes. For instance, the use of a LEGO-axle-port, P1, implies the implementation of a port, P2, for the transfer of rotational mechanical energy. That means that the port P2 subsumes the port P1. In terms of attributes, this can be concluded from the fact that the attributes of P2 are a subset of the attributes of P1. When a designer has defined a port as having the function "transfer rotational mechanical energy," the LEGO-axle-port will be offered as a suggestion for refinement. Since there are possibly many such suggestions, the subsumption mechanism can also be used to arrange the suggestions in a hierarchy from most abstract to most specific. The completeness of this hierarchy will depend on the number of ports that have been previously defined in the design repository.

One could argue that the same conclusion about the LEGO-axle-port could also have been derived from a simple port taxonomy. However, taxonomies are based on a fixed order in which the attributes are considered. For incremental refinement in design, the use of taxonomies would force a designer to make decisions according to this fixed attribute order. This is an unnecessary restriction. For instance, in Fig. 12, LEGO ports are organized by considering either form-attributes or function-attributes first. Depending on the situation, either taxonomy could be most appropriate for defining design refinements. Using the attribute-based port ontology, a Description Logic reasoner (such as RACER [23] or FaCT [22]) can generate the taxonomy dynamically based on *subsumption*.

Compatibility Checking. Beyond the knowledge about possible refinements of ports, the ontology can also represent knowledge about compatibility of ports. Most LEGO ports can be connected to each other by snapping together compatible 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: A cross-shaped axle fits not only into a cross-shaped hole, but also into a circular hole with the same circumscribed radius.

The knowledge about compatibility can be included in the ontology as compatibility axioms. One way to represent this knowledge would be to list exhaustively every valid combination of ports; however, such a list would be unmanageably long and very cumbersome to maintain. A different approach is illustrated in Fig. 13. Here the compatibility axioms relate the *attributes* of the connecting ports. For instance, any port that has a form attribute of type *LEGO-circular-hole-shape* is compatible with any port that has a form attribute of type *LEGO-axle-shape* or *LEGO-pin-shape*. A compatibility axiom at this level of abstraction is most useful in a domain with a limited set of commonly used port geometries. In general, one could define compatibility rules based on complex combinations of detailed form attributes. Compatibil-

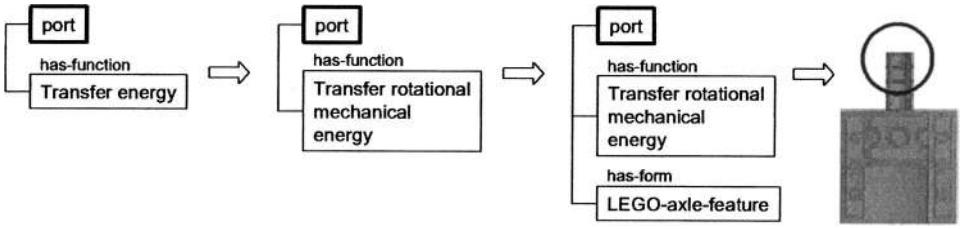

Fig. 11 Port refinement through the addition of attributes.

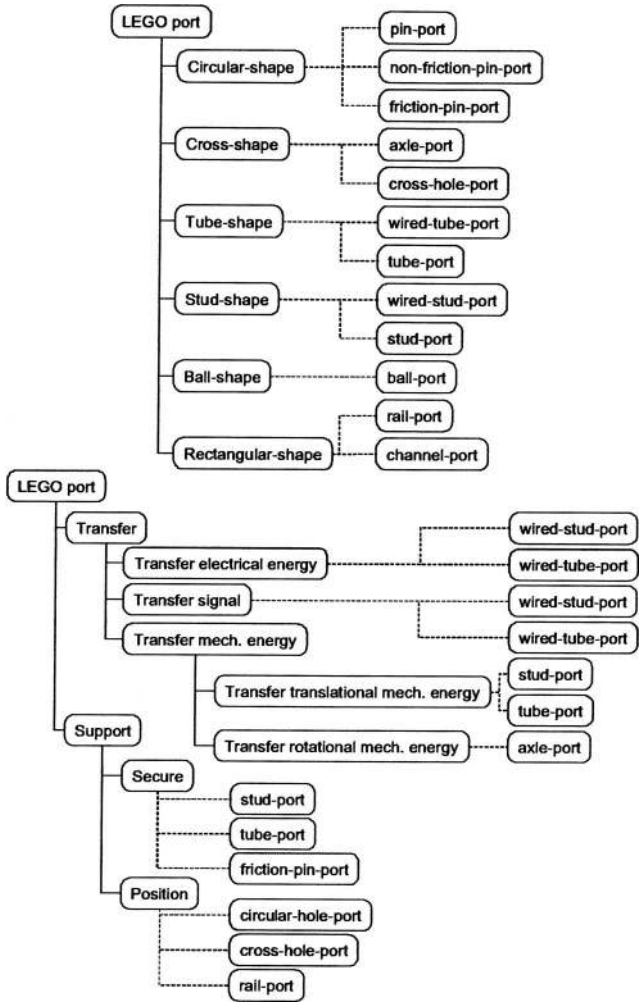

Fig. 12 Two (partial) taxonomies of LEGO ports.

ity could not only depend on the type of the form features but also on the values of the parameters defining the feature instances.

It is also possible to define axioms based on partially defined geometric attributes. For instance, two rotational mechanical ports can only be connected when their rotation axes are aligned. Even when no geometry has been defined yet, compatibility axioms may apply. For instance, during conceptual design, compatibility constraints often apply to functional attributes: An energy transfer port can only be connected to an energy transfer port of the same type.

Interaction Model Associations. A third type of knowledge that can be included in the port ontology is closely related to port compatibility rules: interaction model associations. When generating a simulation model for a system defined as a configuration of components, one needs to consider not only the simulation

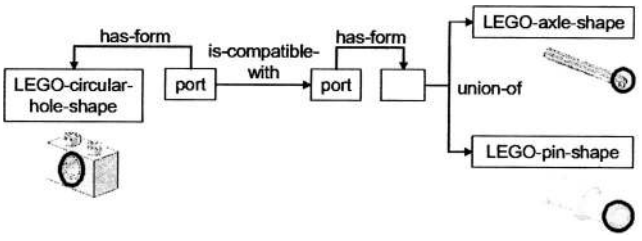

Fig. 13 Compatibility axiom for hole-ports.

models of 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 LEGO axle rotating in a circular-hole-port of a LEGO brick is determined not only by the behavior of the axle and the brick, but also by the interaction between the axle and the brick, 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 with sufficient accuracy 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 [37]. However, in general an algebraic, differential algebraic or even a partial differential equation model is needed to describe the physical phenomena taking place at the area of interaction.

The knowledge needed to decide which interaction models are applicable for a given port connection can be included in the port ontology. To establish the association between a port connection and an interaction model, a connection class is used. Each connection class contains templates for the ports that may be connected and a list of applicable interaction models. One can think of the port templates as the *context* in which the model is applicable; that is, the port templates define the necessary and sufficient attributes:

- *Sufficient*: the interaction models apply for any combination of ports that subsume the templates (i.e., are children of the templates)
- *Necessary*: if any of the attributes in the templates are removed, the models no longer apply.

For instance, Fig. 14 shows a connection class for gear interactions. The class contains only one model (a connection class with multiple models is possible only if all the models have the exact same context, which is not very common). This model (an ideal gear reduction) is applicable for all gear pairs as is captured in the context, the two port templates. The necessary and sufficient attributes for each of the ports specify that each port must have the form of a gear with a given number of gear teeth, and each port must be linked to a rotational connector. In addition to the two port templates, the connection class also defines the *associations* between the model and the two connected ports.

An example of how such connection classes can be used is provided in Fig. 15. For each of the components, there is a corresponding behavioral model. The behavioral models are connected based on the port connections in the configuration graph. Most of the port connections do not require an explicit interaction model because the default connection (corresponding to Kirchhoff's laws) captures the interaction behavior adequately. The interaction between the two gear ports results in the instantiation of a gear interaction model. Based on the connection class in Fig. 14, this would be a model called "Ideal Gear Ratio." However, it is likely that other interaction models also satisfy the particular port context. For instance, there could exist a very high-level model that defines the interaction behavior in terms of power (e.g., Power_in=Power_out) and does not require knowledge of the number of teeth of the gears. Since its port templates would be subsumed by the more detailed port templates in Fig. 14, it would automatically be applicable to any two ports for which the "Ideal Gear Ratio" model applies.

The knowledge captured in the connection classes does not allow us to decide which of the (possibly many) interaction models is most appropriate for a particular system analysis. Answering

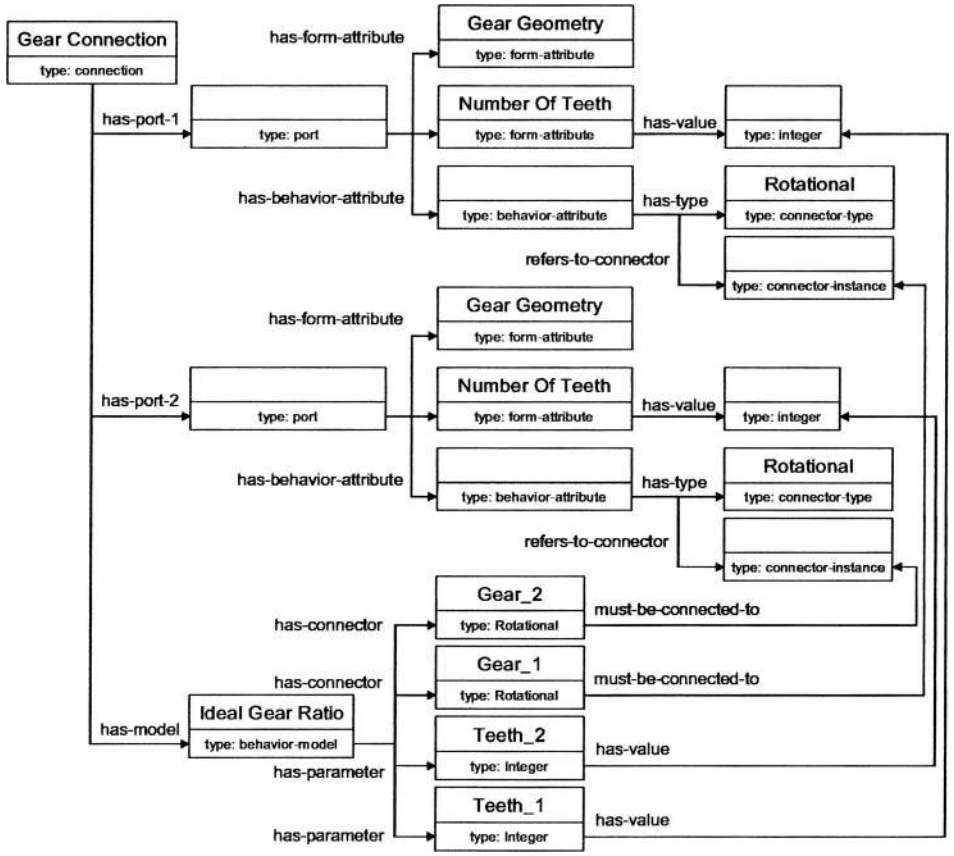

Fig. 14 A connection class for gear interactions. An empty box indicates a wild-card.

that question would require the consideration of many additional factors, such as the purpose of the analysis, the required accuracy, and the available computing resources.

Summary and Discussion

In this article, we investigated ports—abstractions often used in systems design to indicate *locations of intended interaction*. Although ports had been used previously in the context of engineering design, they were never considered from all three perspectives commonly used in design: form, function and behavior. Combining these three aspects into an integrated, computer-interpretable representation of ports provides several advantages:

- It allows designers to capture and formalize design decisions about all aspects of a component's interface. A port ontology allows these decisions to be represented in an unambiguous, computer-interpretable fashion. Although there is bound to be overlap between port and artifact ontologies—ports are subsets of artifacts—it is still useful to consider a separate ontology for ports because ports play such an important role in systems design. Particularly in the context of collaborative design—across space and time—it is essential to be able to communicate unambiguously the decisions that have been taken by different design teams about the interfaces between components or sub-systems.
- The semantic knowledge captured in a port ontology can also support the incremental refinement of design decisions as they relate to component interactions. Based on subsumption, one can retrieve from a design repository those port definitions that contain all the attributes of the existing ports. Any port definition that subsumes the current port is a refinement; that is, it may be derived from the current port by taking additional design decisions. Such refinements, retrieved from a repository and organized in a taxonomy, provide the designers with possible design alternatives from which to choose. Other refinements that are not limited to component interactions, such as the refinement of the function of component as a whole, require a broader ontology that encompasses the entire component and are not supported by the current port ontology.
- A second type of design knowledge that can be included in the port ontology is port compatibility knowledge. When two ports are connected they should satisfy certain compatibility criteria. When this compatibility knowledge is included in the ontology, a computer-based support tool could verify compatibility between ports when different design teams make changes to the interface between the sub-systems that they are designing.
- Finally, a third type of knowledge in the port ontology sup-

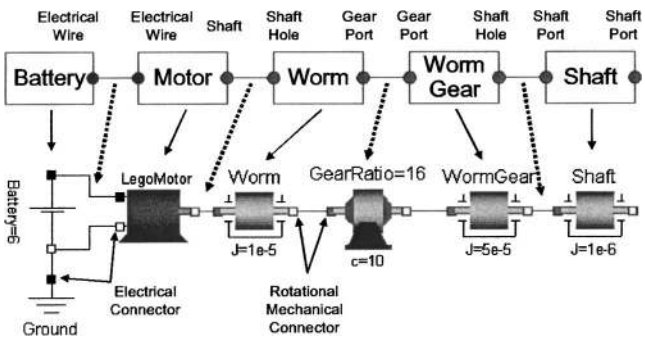

Fig. 15 A simple LEGO system configuration with a corresponding behavioral model (as modeled in Modelica).

ports interaction model instantiation. The dynamic interaction between two ports needs to be modeled in an interaction model. Associating models to particular combinations of port types can be accomplished through a ''connection'' class in which associations are established between the behavioral variables in port templates and the behavioral variables in the corresponding interaction model. Associations between port parameters and interaction model parameters are also established.

Design refinement, compatibility checking and interaction model instantiation have all been demonstrated using an ontology for LEGO systems. Although this article provided a basic framework for representing port information and knowledge in a port ontology, this ontology is far from complete and needs to be further expanded in several directions:

- Since ports can be used to characterize interactions in almost any application domain and from any perspective, the amount of knowledge necessary to characterize all such ports is enormous—well beyond the scope of a single article, a thesis, or even a career. The purpose of this article is to establish the basic structure for formalizing such knowledge with the understanding that this basic structure needs to be filled in and expanded to include the specific knowledge about a given application domain—as was illustrated for LEGO systems.
- Ports provide a natural interface demarcating sub-systems within a system, and they facilitate the negotiation of interface constraints among different design teams. However, for some aspects of the design problem, ports need to be considered within the context of the artifacts of which they are part. For instance, the functions defined in ports are related to the overall functionality of the system. In the example of a windshield wiper, the motor transforms electrical energy into mechanical energy and must, therefore, also have ports for transferring the electrical energy in and the mechanical energy out; every energy flow serving as an input or output to the main function of the artifact must be associated with a corresponding port.
- An ontology, as introduced in this article, requires a specialized graphical user interface to be used by designers. Although possible, it is not convenient to create the instances of ports, attributes and connections using a text editor. Ontology editors, such as Protégé [42], may provide an intermediate solution during development, but ultimately a specialized software tool will need to be developed.
- The reasons for developing this ontology are to improve the capture of design decisions, to facilitate the communication of these decisions among disparate teams of designers, and to enable computer support and possibly automation of these design tasks. The validation of whether this ontology satisfies these goals will require further development of the computer tools that make use of the ontology followed by controlled experiments to verify improvements in the productivity of designers and in the quality of the designed systems. Within the scope of this article, the validation was limited to a proof-of-concept illustration in the context of LEGO systems.

Acknowledgment

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.

The authors would also like to acknowledge the anonymous reviewers. Their insightful and constructive comments were very instrumental in the revision of this article.

References

[1] Wertz, J. R., and Larson, W. J., 1999, *Space Mission Analysis and Design (Third Edition)*, Kluwer Academic Publishers, London, UK.
[2] Ostergaard, K. J., and Summers, J. D., 2003, ''A Taxonomy for Collaborative Design,'' *ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, ASME, Chicago, Illinois, DAC-48781.
[3] Szykman, S., Racx, J. W., and Siram, R. D., 1999, ''The Representation of Function in Computer-Based Design,'' *ASME Design Engineering Technical Conference*, ASME, Las Vegas, Nevada, DTM-8742.
[4] Chakrabarti, A., and Bligh, T. P., 1996, ''An Approach to Functional Synthesis of the Mechanical Design Concepts: Theory, Applications, and Emerging Research Issues,'' Artificial Intelligence for Engineering Design, Analysis and Manufacturing, **10**, pp. 313–331.
[5] Kitamura, Y., and Mizoguchi, R., 2003, ''Ontology-Based Description of Functional Design Knowlege and its Use in a Functional Way Server,'' Expert Sys. Applic., **24**(2), pp. 153–166.
[6] Umeda, Y., Takeda, H., Tomiyama, T., and Yoshikawa, H., 1990, ''Function, Behavior and Structure,'' Applications of Artificial Intelligence in Engineering, Springer-Verlag, **1**, pp. 177–193.
[7] Roth, K., 1984, ''Analyze und Systematische Einteilung Fester Verbindungen,'' Konstruktion, **36**(7), pp. 241–252.
[8] Horváth, I., Dorozsmai, K., Thernesz, V., 1994, ''A Feature-Object-Based Practical Methodology for Integration of Conceptual and Morphological Design,'' *Lancaster International Workshop on Engineering Design CACD '94*, Lancaster University EDC, Lancaster University, pp. 131–149.
[9] Horváth, I., Vergeest, J. S. M., and Kuczogi, G., 1998, ''Development and Application of Design Concept Ontologies for Contextual Conceptualization,'' *ASME Design Engineering Technical Conferences*, ASME, Atlanta, GA, DETC98/CIE-5701.
[10] Counsell, J., Porter, I., Dawson, D., and Duffy, M., 1999, ''Schemebuilder: Computer Aided Knowledge Based Design of Mechatronic Systems,'' Assembly Automation, **19**(2), pp. 129–144.
[11] Singh, P., and Bettig, B., 2003, ''Port-Compatibility and Connectability Based Assembly Design,'' *ASME 2003 Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, ASME, Chicago, IL, DETC2003/DAC-48783.
[12] Sinha, R., 2001, Compositional Design and Simulation of Engineered Systems, Ph.D. Thesis, Institute for Complex Engineered Systems, Carnegie Mellon University, Pittsburgh, PA.
[13] Stone, R. B., and Wood, K. L., 2000, ''Development of a Functional Basis for Design,'' J. Mech. Des., **122**(4), pp. 359–370.
[14] Karnopp, D., and Rosenberg, R. C., 1968, *Analysis and Simulation of Multiport Systems, the Bond Graph Approach to Physical System Dynamics*, MIT Press, Cambridge, Massachusetts.
[15] Bray, T., Paoli, J., Sperberg-McQueen, C. M., and Maler, E., 2000, Extensible Markup Language (Xml) 1.0 (Second Edition), World Wide Web Consortium, http://www.w3.org/TR/REC-XML.
[16] ISO, 1994, 10303-11 Industrial Automation Systems and Integration-Product Data Representation and Exchange-Part 11: Description Methods: The Express Language Reference Manual, International Organization for Standardization, www.iso.ch/cate/cat.html.
[17] Uschold, M., and Gruninger, M., 1996, ''Ontologies: Principles, Methods, and Applications,'' The Knowledge Review, **11**(2), pp. 93–136.
[18] Fensel, D., 2000, *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*, Springer-Verlag, Berlin.
[19] Farquhar, A., Fikes, R., and Rice, J., 1997, ''The Ontolingua Server: A Tool for Collaborative Ontology Construction,'' International Journal of Human-Computer Studies, **46**, pp. 707–728.
[20] Nebel, B., 1996, ''Artificial Intelligence: A Computational Perspective,'' *Principles of Knowledge Representation*, CSLI Publications, Stanford, CA, pp. 237–266.
[21] Borgida, A., Brachman, R. J., and McGuinness, D. L., 1989, ''CLASSIC: A Structural Data Model for Objects,'' *ACM SIGMOD International Conference on Management of Data*, Portland, Oregon, pp. 59–67.
[22] 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)*, Morgan Kaufmann Publishers, San Francisco, CA, pp. 636–647.
[23] Haarslev, V., and Moller, R., 2001, ''Description of the RACER System and Its Applications,'' *2001 International Description Logic Workshop (DL2001)*, Stanford, CA.
[24] Hendler, J., and McGuinness, D. L., 2000, ''The DARPA Agent Markup Language,'' IEEE Intell. Syst., **15**(6), pp. 67–73.
[25] Fensel, D., Horrocks, I., van Harmelen, F., McGuinness, D. L., and Patel-Schneider, P. F., 2001, ''Oil: An Ontology Infrastructure for the Semantic Web,'' IEEE Intell. Syst., **16**(2), pp. 38–45.
[26] Fensel, D., van Harmelen, F., and Horrocks, I., 2003, ''OIL and DAML + OIL: Ontology Languages for the Semantic Web,'' *Towards the Semantic Web: Ontology-Driven Knowledge Management*, John Wiley & Sons, Hoboken, NJ, pp. 288.
[27] Dean, M., Connolly, D., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D. L., Patel-Schneider, P. F., and Stein, L. A., 2002, OWL Web Ontology Language (OWL) Reference Version 1.0, W3C, http://www.w3.org/TR/2002/owl-ref/.
[28] Schlenoff, C., Denno, P., Ivester, R., Szykman, S., and Libes, D., 1999, ''An

Analysis of Existing Ontological Systems for Applications in Manufacturing,'' *ASME Design Engineering Technical Conference/Computer in Engineering*, ASME, Las Vegas, NV, EIM-9024.
[29] Ray, S. R., 2002, ''Interoperability Standards in the Semantic Web,'' J. Comput. Inf. Sci. Eng., **2**(1), pp. 65–69.
[30] Horváth, I., and van Der Vegte, W. F., 2003, ''Nucleus-Based Product Conceptualization-Part 1: Principles and Formalization,'' *International Conference on Engineering Design*, Stockholm.
[31] Kitamura, Y., and Mizoguchi, R., 2003, ''An Ontological Schema for Sharing Conceptual Engineering Knowledge,'' *International Workshop on Semantic Web Foundations and Application Technologies*, Nara, Japan, pp. 25–28.
[32] Adachi, Y., Forester, J., Hyvarinen, J., Karstila, K., Liebich, T., and Wix, J., 2003, Industry Foundation Classes IFC2x Edition 2, http://www.iai-international.org/iai_international/Technical_Documents/R2×2_final/index.html.
[33] Shah, J. J., 1991, ''Conceptual Development of Form Features and Feature Modelers,'' Res. Eng. Des., **2**, pp. 93–108.
[34] Shah, J. J., and Mathew, A., 1991, ''Experimental Investigation of the Step Form-Feature Information Model,'' Comput.-Aided Des., **23**(4), pp. 282–296.
[35] Hirtz, J., Stone, R. B., Szykman, S., McAdams, D. A., and Wood, K. L., 2001, ''A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts,'' Res. Eng. Des., **13**(2), pp. 65–82.
[36] IEEE, 1999, *1076.1 Working Group: Analog and Mixed-Signal Extensions for Vhdl*, IEEE.
[37] Mattsson, S. E., Elmqvist, H., and Otter, M., 1998, ''Physical System Modeling with Modelica,'' Control Eng. Pract., **6**, pp. 501–510.
[38] Cellier, F. E., 1991, *Continuous System Modeling*, Springer-Verlag, New York.
[39] Modelica Association Members, 2003, Modelica Standard Library (Version 1.5), Modelica Association, http://www.modelica.org/libraries.shtml.
[40] Schmit-Schauß, M., and Smolka, G., 1991, ''Attributive Concept Descriptions with Complements,'' Agric. Water Manage., **48**(1), pp. 1–26.
[41] Nardi, D., and Brachman, R. J., 2003, ''An Introduction to Description Logic,'' *The Description Logic Handbook: Theory, Implementation and Applications*, Cambridge University Press, Cambridge UK.
[42] Stanford Medical Informatics, 2003, The Protégé Project, Stanford Medical Informatics, http://protege.stanford.edu/.