A Semantic Product Modeling Framework and Its Application to Behavior Evaluation

Jae H. Lee, Steven J. Fenves, Conrad Bock, Hyo-Won Suh, Sudarsan Rachuri, Xenia Fiorentini, and Ram D. Sriram, *Member, IEEE*

***Abstract*—Supporting different stakeholder viewpoints across the product lifecycle requires semantic richness to represent product-related information and enable multiview engineering simulations. This paper proposes a multilevel product modeling framework enabling stakeholders to define product models and relate them to physical or simulated instances. The framework is defined within the Model-Driven Architecture using the multilevel (data, model, metamodel) approach. The data level represents real-world products, the model level describes models (product models) of real-world products, and the metamodel level describes models of the product models. The metamodel defined in this paper is specialized from a web ontology language enabling product designers to express the semantics of product models in an engineering-friendly way. The interactions between these three levels are described to show how each level in the framework is used in a product engineering context. A product design scenario and user interface for the product metamodel is provided for further understanding of the framework.**

***Note to Practitioners*—The views of stakeholders in a product lifecycle may be different according to their concerns. However, they develop product models and data for a same product. The product models and data should be managed in a single framework to validate consistency of product information. The framework proposed in this paper enables stakeholders to define their product models and relate them to physical or simulated instances. A generic metamodel is also proposed to guide engineers in building their product models using engineer-friendly terms. A product design scenario and user interface prototype was implemented for further understanding of the framework and the metamodel. If the prototype is integrated with a Computer-Aided Design (CAD) system, it can be a powerful mechanism for semantically annotating CAD models.**

***Index Terms*—Multilevel modeling framework, OWL, product information modeling, semantic product modeling.**

Manuscript received April 28, 2011; accepted July 08, 2011. Date of publication September 15, 2011; date of current version December 29, 2011. This paper was recommended for publication by Associate Editor Y. Ma and Editor M. Zhou upon evaluation of the reviewers' comments. No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied. Certain commercial equipment, software, instruments, or materials are identified in this report to facilitate better understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply the materials, software, or equipment identified are necessarily the best available for the purpose.

J. H. Lee, S. J. Fenves, C. Bock, S. Rachuri, and R. D. Sriram are with the National Institute of Standards and Technology, Gaithersburg, MD 20899 USA (e-mail: lee.jaehyun@nist.gov; steven.fenves@nist.gob; conrad.bock@nist.gov; sudarsan@nist.gov; sriram@nist.gov).

H.-W. Suh is with the Korea Advanced Institute of Science and Technology, Daejoen 305-701, South Korea (e-mail: hw_suh@kaist.ac.kr).

X. Fiorentini was with the National Institute of Standards and Technology, Gaithersburg, MD 20899 USA. She is now an independent consultant in Italy (e-mail: xenia.fiorentini@gmail.com.).

Digital Object Identifier 10.1109/TASE.2011.2165210

I. Introduction

COLLABORATIVE environments enable participants in a product lifecycle to interact and reach agreement by sharing design knowledge and product information [1], [2]. Product knowledge and information in a product lifecycle requires representation of data and models in order to integrate the systems across different lifecycle stages [3]. Product lifecycle information also requires sophisticated product modeling techniques such as multiple object representations and complex semantic relationships [4]. In addition, collaborative environments need a product ontology to provide open semantics in which knowledge from multiple sources can be easily combined and checked for consistency [5]. The product ontology should include a generic product model that: 1) can be readily specialized for specific products; 2) can provide information to all stakeholders (i.e., designer, engineer, manufacturer, supplier etc.) throughout the product lifecycle; and 3) has explicit, logical semantics of the concepts and relationships involved, without requiring that the stakeholders be versed in ontological thinking.

These needs gave rise to multilevel information modeling frameworks. The frameworks defines the multilevel (data, model, metamodel) approach to build product models and relate them to physical or simulated data and thus enable integration of product lifecycle information [6]. Fig. 1 shows a high-level view of our framework and possible implementation. The semantic-based product metamodel (SPMM) consists of generic product domain concepts and relationships, such as artifact, behavior, and their relationships (explained later in the paper), which can guide engineers in building their product-specific models, such as models of cars, airplanes, and ships. An impeller product model is shown in Fig. 1 as a product-specific model example. Information about physical product data can be defined from models built in the framework. Product-specific models and physical product data are then saved in a formal knowledge representation for reasoner to understand and infer new knowledge.

The generic product domain concepts alleviate the burden on engineers to learn syntax of the ontology language. Product models and data defined by engineers are then converted into a formal representation with axioms so that reasoners can check consistency of the product information. Domain ontology and rules for a product model can infer new knowledge from the information about the product model and its physical instances.

Among product lifecycle information, product behavior information must be modeled in a multilevel information mod-

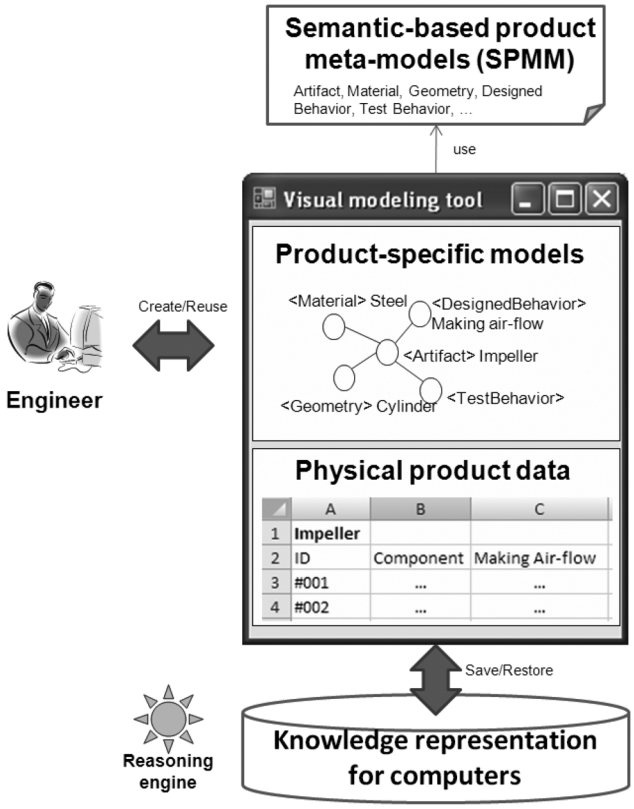


Fig. 1. A high-level view of the product modeling framework.

eling framework because product behavior information involves both data and languages. For example, product behavior information can be either a behavior model or test data of prototypes. Each level can have multiple degrees of abstraction within it. For example, product behavior information in design can be a behavior model of either conceptual design or detail design.

Behavior design models are generally evaluated by either a simulation or a prototype test, so the relationship between behavior models and test data of prototypes also should be addressed in a multilevel information modeling framework. Behavior evaluation is a design activity to find out whether a product design model satisfies its requirements. Behavior evaluation happens often in a product lifecycle. Test data can be collected from a design prototype or simulation, manufactured product, or product in use, and compared to the design model for evaluation. Even when test data fail their design model during evaluation, the evaluation result is a knowledge asset of a company to improve future designs.

This paper defines the structure of an information modeling framework for product behavior evaluation, and proposes how engineers can build their product models and data in the multilevel framework. The framework clarifies the meaning of the product information modeling levels through: 1) the use of generic product model, SPMM, as the product metamodel; 2) the description of product models using concepts and relationships in SPMM; and 3) the representation of information about real-world products as instances of the product models. The concepts and relationships in SPMM extend existing work on ontological product modeling languages [6] and the Core Product Model 2 (CPM2) [7] to support behavior modeling in the product lifecycle.

This paper is organized as follows. Section II reviews previous research on multilevel product modeling frameworks. Section III presents an overview of the proposed multilevel product modeling framework, including the generic product model. Section IV addresses the concepts and relationships in the generic product model, and their semantics. Section V describes the interaction mechanisms between levels of the framework. Section VI provides a proof-of-concept for SPMM with a fan and motor example, and shows an SPMM editing user interface. Section VII gives suggestions for future work and conclusions of this paper.

II. Previous Research

Previous work on multilevel product modeling varies in how they define levels in product ontologies. Lee *et al.* [8] proposed a multilayered ontology architecture for collaborative enterprises, in which representation layers and domain modeling layers are separated, and adopted the Model-Driven Architecture (MDA) [9] for the representation of layers. The modeling layers included: a domain-independent layer; a domain-dependent layer; and a domain-specific layer. The domain modeling layers could be built using a top-level ontology such as the suggested upper merged ontology (SUMO) [10]. Lee and Suh [11] proposed an ontology-based four-layered product knowledge framework to manage comprehensive product lifecycle knowledge. Product ontologies were classified into four types: a product-context model; product-specific model; product-planning model; and product-manufacturing model. The ISO-15926 standard for Life Cycle Data for Process Plants [12] also had a multilayered knowledge framework for plant process information and built multilayered plant ontologies. Yang *et al.* [13] proposed a multilayered ontology architecture for product configuration consisting of four layers: a representation layer; a metamodel layer; a model layer; and an instance layer. The CPM2 [7] focused on a generic product model that encompasses a range of engineering design concepts beyond the artifact's geometry, including function, form, behavior, and material, as well as physical and functional decompositions, and various kinds of relationships among these concepts. The CPM2 report [7] suggests a level named "intermediate model" to express product-specific models specialized from the generic concepts and relationships.

Some previous efforts in multilevel product modeling [8]–[13] have a level just for an ontology language, such as first-order logic [14], description logic [15], or the web ontology language (OWL) [16]. Product information is captured with instantiation relationships between product models and the ontology language level. Specialization relationships are used to build product-specific models from generic product models. The "Previous Approach" column in Fig. 2 shows an example of how these levels are related to each other. These approaches assume product designers understand ontology languages well enough to build product-specific models. However, it is a heavy burden on product designers to understand these formalisms

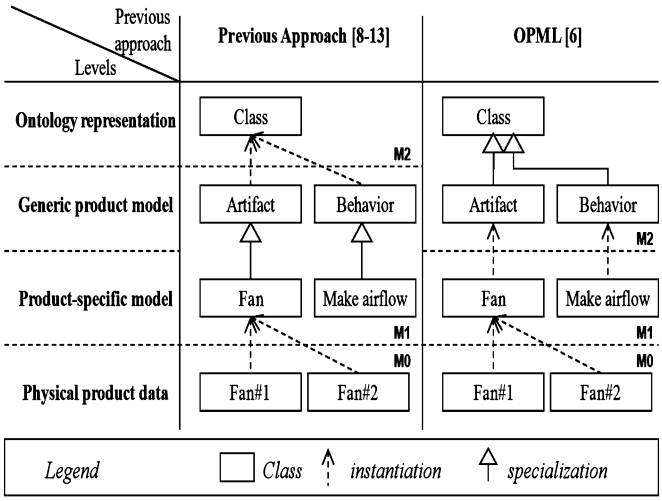


Fig. 2. Multilevel approaches and relationships between levels.

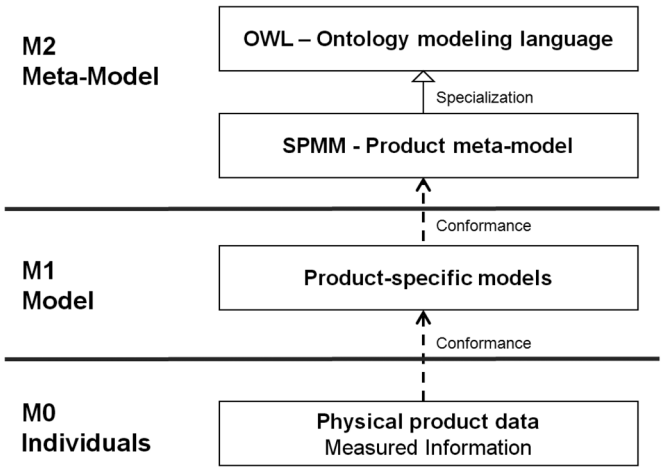


Fig. 3. The multilevel product information modeling framework.

and use them to define product models. Therefore, a generic product modeling language specialized from an ontology language is necessary for product designers to build and use their product models based on generic product modeling concepts that are independent of specific product domains.

The ontological product modeling language (OPML) [6] has the capabilities of ontology languages, such as taxonomies, which are available to engineers in their own terminology, rather than in those of ontology languages such as OWL. The OPML adopted the multilevel approach of the MDA [17], and clarifies the meanings of three levels, labeled M0, M1, and M2. Instantiation relationships express relationships between each two adjacent levels. The M0 level consists of individuals that can have no further instantiations. The M1 level consists of models that classify individuals. The M2 level consists of a metamodel, and consists of shorthand expressions for the semantics of the model. OPML defines a metamodel specialized from OWL in M2. The "OPML" column in Fig. 2 shows an example of how levels in the OPML are related to each other.

The multilevel product modeling framework is useful especially when both product-specific models and physical product data are required during product development activities, such as behavior evaluation. The generic product model of the CPM2 provides concepts and attributes for behavior evaluation such as "behavior," "observed behavior," and "evaluated behavior." However, the CPM2 did not explain how to use these concepts and attributes in a multilevel product modeling framework. The OPML provides generic concepts and relationships for behavior modeling, such as objects involved in behaviors. The OPML also supports the concept of the Environment, the part of the universe that uses, interacts with, or is affected by the artifact being designed, but is beyond the designer's control. However, the OPML needs to be harmonized with additional aspects of the CPM2, in particular, modeling and reasoning for behavior evaluation. This paper provides the harmonization of the OPML and the CPM2 for behavior evaluation, and explains how behavior concepts and relationships are modeled in the multilevel framework and how reasoning can be exploited in the framework for behavior evaluation.

III. The Multilevel Product Information Modeling Framework

The proposed framework contains three levels for ontological product modeling, namely, metamodels (M2), models (M1), and individuals (M0). Fig. 3 shows the three levels and the relationships among them.

The M2 level is for the metamodel, where SPMM is proposed. As shown in Fig. 3, the classes and relationships in SPMM are specialized from OWL classes, and the syntax and semantics of SPMM inherit the syntax and semantics of OWL. The semantics of SPMM need to be specified further because its classes and relationships are specific to product modeling. Since OWL is used as a basis for SPMM at M2, OWL is used to represent the axioms of SPMM classes and relationships explicitly. A detailed description of SPMM classes and relationships at the M2 level will be given in Section IV.

The M1 level is for product models, representing information about particular products. It consists of product concepts and relationships such as artifacts, behaviors, forms, and structures of specific products, including attributes and their (required or designed) values. Product designers define the concepts and relationships as instances of SPMM, so that they can be checked by a reasoner to determine their conformity to the axioms of SPMM (this is shown in Fig. 3 as Conformance between the M1 and M2 levels). If the concepts and relationships at M1 do not create any inconsistency with SPMM, they are valid, and their axioms can be added or inferred. Domain experts can define most of the axioms at M1, but some of the axioms can be inferred from axioms of the product metamodel at M2.

The M0 level is for product data about physical instances observed at a certain time and place, or in computer simulations. The observed product data may or may not satisfy axioms of its product model at M1, and their conformity can be checked by a reasoner (this is shown in Fig. 3 as Conformance between the M0 and M1 levels). A conformance relationship between a product model and a physical product is a relationship engineers

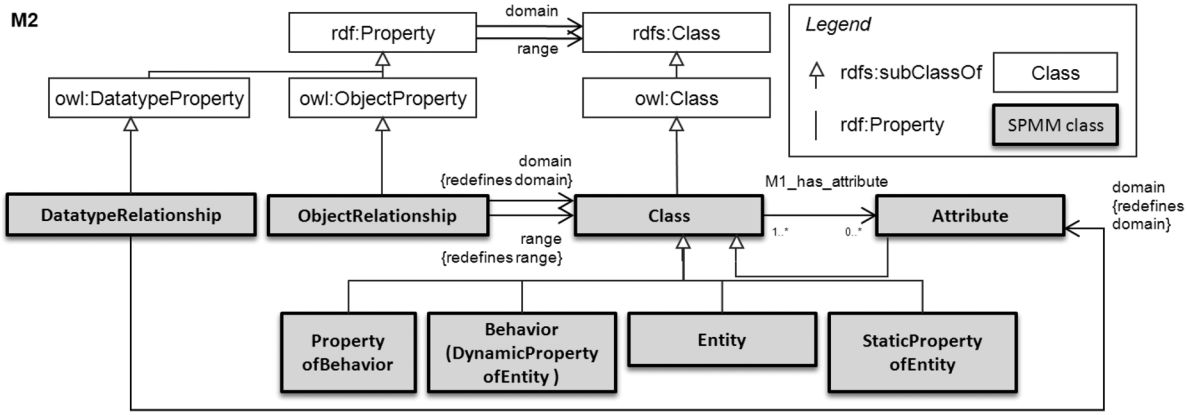


Fig. 4. Relationships between OWL and SPMM classes.

want to know when they validate a product model or test a physical product. The relationship is particularly important when the lifetime of the manufactured product is so long that the physical instance may change over time, as in the case of ships, aircraft, and buildings.

Fig. 3 includes specialization and conformance relationships. The specialization relationship is used in conceptual modeling to represent taxonomies, such as small cars being a specialization of cars in general. If a concept A is a concept specialized from a concept B, then the instances of concept A are collectively a subset of those of B, or more precisely, the instances of A are also instances of B. The more specialized concept A might constrain properties of concept B or introduce new ones, but any constraints or properties of B apply to all the instances of A as well as those of B, by the definition of specialization [18]. The specialization relationship is implemented in this paper as the subclass relationship in OWL ("rdfs:subClassOf"), but this is not to be confused with subclassing in software modeling, which has a related but different meaning. The conformance relationship between an instance and a class is an instantiation relationship in which reasoners logically checked the consistency of the instance along the class's definition.

A physical individual at M0 can conform to several product models at M1 such as a requirement model, a conceptual design model, an engineering model, etc. Since a physical individual has detailed information, it can have more attributes than its product models at M1. Definitions at the M1 level can be used to infer the conformance relationships between the M0 and M1 levels. A detailed description of the interaction among the M0, M1, and M2 level will be given in Section V.

IV. The Semantic-Based Product Metamodel: SPMM

This section describes the classes and relationships of the product metamodel at the M2 level, in short, SPMM classes. SPMM classes are described in OWL, but they are visualized using UML class notation [19] in this section.

SPMM classes have axioms to specify their meanings explicitly. The axioms are also described in OWL, so that they can provide OWL syntax for SPMM to describe specific product models at M1. Fig. 4 shows the relationships between SPMM classes and OWL primitives. The OWL primitives in Fig. 4 are shown with the "owl:" prefix before their class names. The "spmm:Class" is the top class in SPMM, and it is a subclass of the "owl:Class." The "spmm:Class" can have relationships with other "spmm:Class" through "spmm:ObjectRelationship" class. The "spmm:Class" can also have relationships with "spmm:Attribute," which represents attributes of a class such as "weight," "color," and "speed." SPMM defines "spmm:Attribute" as a subclass of "spmm:Class" in order to give unique identifications to each attribute at M1. Attributes in a product model should be identified for defining constraints among attributes and for sharing attributes among product models. The "spmm:DatatypeRelationship" is used to represent relationships between "spmm:Attribute" and "rdf:Literal" so that the "spmm:Attribute" can have relationships with data-types, units, as well as literal values. In the following sections, the bold font is used in the text to avoid repeating usage of the "spmm:" prefix. For example, **Class** means spmm:Class.

Entity, **Behavior**, **StaticPropertyofEntity**, and **PropertyofBehavior** are crucial concepts for building product models. **Entity** is things that can be described with static properties and dynamic properties. **Behavior** is a dynamic property of one or more **Entities**. A dynamic property includes the notion of time in its description. For example, a behavior "rotating a fan" happens over time, involving "motor" and "fan" entities. **StaticPropertyofEntity** is a property of one or more **Entity** that does not include the dynamics of the **Entity**, such as the shape of a 'motor' entity being two cylinders. **PropertyofBehavior** is a property of **Behavior**. It is specially defined to capture relationship information among product behaviors.

Entity, **Behavior**, **StaticPropertyofEntity**, and **PropertyofBehavior** have relationships among themselves, and have further subclasses that inherit the relationships. The following subsections will explain their subclasses further. Fig. 5 shows the class hierarchy and relationships graphically.

A. Entity Class

Entity could be **ExternalEntity** and **SpecifiedEntity**, and **SpecifiedEntity** is divided further into **Artifact** and **Feature**.

ExternalEntity is a kind of entity that interacts with artifacts in a context of use, which is a required behavior. The required behavior identifies some entities participating in it as the external entities. The external entities interact with the entity being

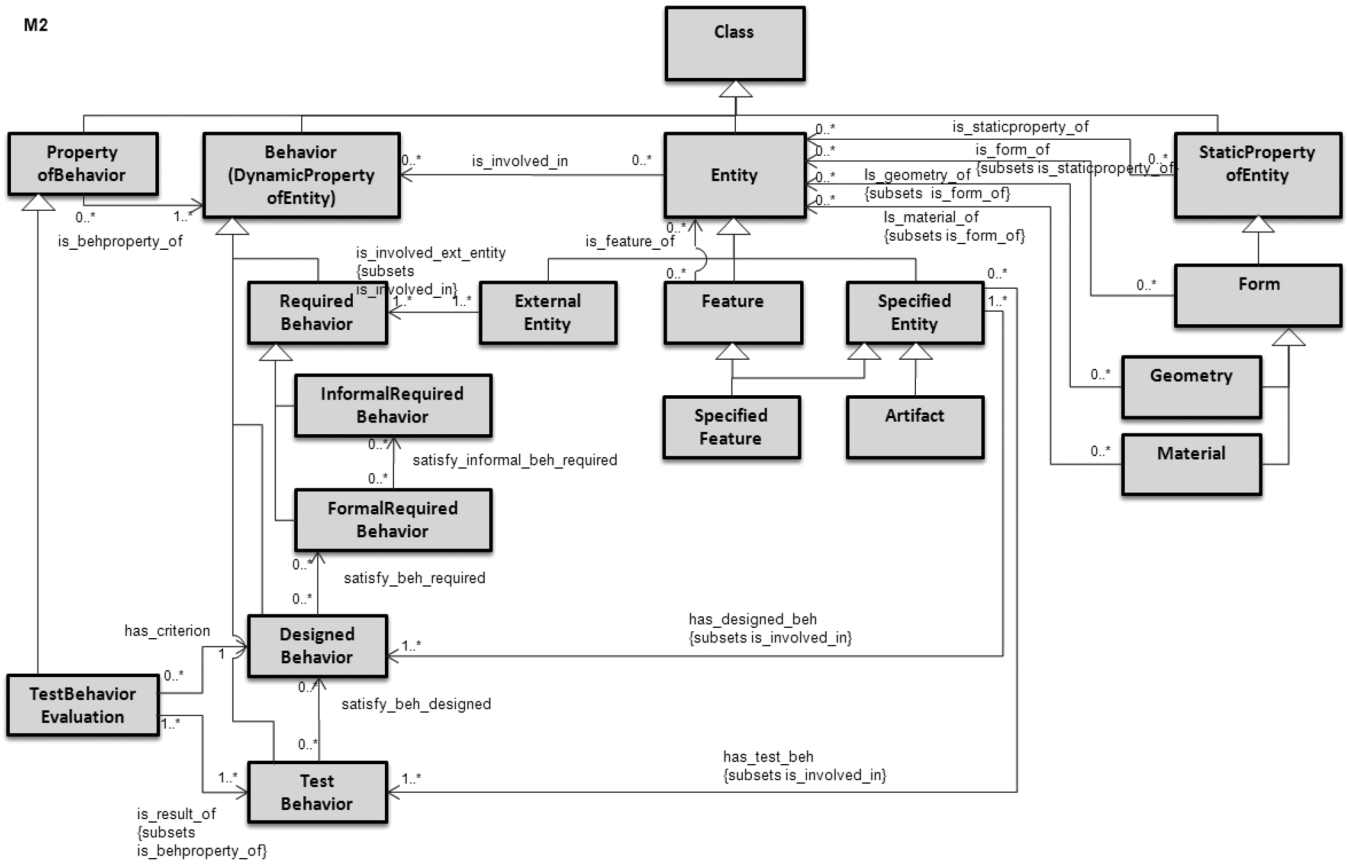


Fig. 5. Entity, behavior, StaticPropertyofEntity, and PropertyofBehavior classes and relationships.

specified (the artifact), which is also involved in the required behavior. For instance, a person's hand and mouth, and water can be external entities for designing a water bottle because they are involved in required behaviors of using a water bottle.

SpecifiedEntity is a kind of **Entity** that is specified with **SpecifiedBehavior** and **Form**. **Artifact** and **SpecifiedFeature** are subclasses of **SpecifiedEntity**. A specified entity must be involved in at least one designed behavior. A specified entity may have a form such as geometry or material information. For example, a water bottle can be a specified entity that has a specified behavior "containing water" and a material (a subclass of form) "plastic."

Artifact is a kind of **SpecifiedEntity** which is designed, as opposed to entities that are naturally occurring. Artifacts can be involved in required behaviors through the "spmm:is_involved_in" relationship between **Entity** and **Behavior**. They can also have relationships with forms such as geometry and material. Artifacts must be involved in at least one designed behavior, otherwise they would be natural objects. Artifacts may have a relationship with test behaviors if their designed behaviors are tested.

Feature is a kind of **Entity**. It can be just a design idea, or a portion of an external entity or artifact. The "spmm:is_feature_of" relationship associates **Feature** to either **ExternalEntity** or **Artifact**. A feature has relationships with forms and behaviors because it is a subclass of **Entity**. **SpecifiedFeature** is a kind of **Feature** that is designed by engineers, so it is also a kind of **SpecifiedEntity**. A specified feature must have at least one designed behavior. For example, a "bottleneck" feature of a "water-bottle" artifact can be a feature the form of which is a "through-hole," and the designed behavior of which is "guiding waterflow."

B. Behavior (DynamicPropertyofEntity) Class

Behaviors are specialized into three subclasses: **RequiredBehavior**, **DesignedBehavior**, and **TestBehavior**. All of these can identify entities participating as specified entities (artifacts) and others as external entities. Behaviors of **ExternalEntity** include interactions with artifacts, so they are named **RequiredBehavior** in this paper. Required behaviors do not give any information about artifacts beyond the interaction with external entities. They are specialized into two subclasses:

- **InformalRequiredBehavior** is an informal behavioral description of the dynamics of the external entities surrounding artifacts when they are used, including interactions with the artifact. For example, "Person is drinking water" might be an informal required behavior. A designer can start to think about the interaction of the artifact and external entities such as "person" and "water," based on the informal description by adding sub-behaviors: "Hand holds the artifact" and "Mouth contacts the artifact." Once a description goes into any detail about the artifact, even to classify it, such as being a water bottle, the behavior crosses into design, and is no longer a pure requirement.
- **FormalRequiredBehavior** explicitly specifies the behavior of external entities, including their interactions

with artifacts. A formal required behavior can be derived from an informal required behavior. While an informal required behavior is a class expressed with a sentence, a formal required behavior is a class expressed with a structured sentence and attributes. A structured sentence consists of a subject, verb, and object [20]. Entities can be used as a subject or object. From the previous example, "person," "hand," "mouth," and "water" are defined as external entities, and a "water bottle" is defined as an artifact. Verb taxonomies are required to specify verbs. For instance, Hirtz [21] and Kitamura [22] defined verb taxonomies to describe engineering behaviors and functions, respectively. From the previous example, "drink," "hold," and "contact" are verbs to describe the required behaviors, and a formal required behavior "FB01G," can be expressed with a subject "person," a verb "drink," and an object "water." Attributes are also needed to specify a formal required behavior. For instance, the "FB01" formal required behavior has an attribute "waterflow-rate" whose value is "more than 50 cc/s."

DesignedBehavior specifies behaviors of an artifact or feature beyond interaction with external entities, as determined by product designers in response to formal required behaviors. For example, "Water-bottle provides water to mouth" is a designed behavior for the above required behavior example, because it specifies that the artifact is a water bottle. The subject of its description should be an artifact or feature, while the subject of a required behavior's description is an external entity. A designed behavior can have specific attributes and values that satisfy the attributes and values defined at corresponding required behaviors. For example, the "waterflow-rate" of the above designed behavior can have a value "more than 60 cc/s."

Designed behaviors are defined to satisfy required behaviors. The relationship between **RequiredBehavior** and **DesignedBehavior**, "satisfy_beh_required" relationship in Fig. 5, is used to assert that the design behavior is enough for an artifact to perform its required behavior. For example, since the "waterflow-rate" of the previous designed behavior is "more than 60 cc/s," it satisfies the "waterflow-rate" of the "FB01" formal required behavior, which was "more than 50 cc/s."

TestBehavior is a behavior of an artifact or feature that can be observed after testing with a test method. External entities and specified entities can be involved in a test behavior, but a test behavior cannot exist without a related specified entity. **TestBehavior** is the observed behavior of the artifact or specified feature. It can be observed through test methods such as running mathematical models or simulation models, or operating physical prototypes or manufactured items. Test behaviors should keep their test method information by referring to classes of an imported test method ontology, for example, by defining a test method attribute of the test behaviors at the M1 level.

The proposed behavior classification is not a new one. The CPM2[7] has classes such as requirement class, intended behavior class, and observed behavior class. A product model proposed by Garcia *et al.* [23] also classifies product behavior into three classes such as desired product behavior, predicted product behavior, and observed product behavior. Although those class names are different, their meanings are similar to the **RequiredBehavior**, **DesignedBehavior**, and **TestBehavior**.

The difference between the proposed behavior classes and the classes in the CPM2 [7] is that the proposed behavior classes are defined at M2 so that behavior classes at M1, defined by engineers, can have logical and explicit definitions. The definitions of behavior classes at M1 can be inputs for a reasoning engine to infer specialization relationship among behavior classes at M1 as well as to check conformance relationship between behavior classes at M1 and their individuals at M0. For example, let us assume that an engineer defines "rotating_a_fan_req" and "rotating_a_fan_des" classes at M1 as a formal required behavior and a designed behavior of a motor, respectively, and each class has definitions including their attributes and certain values. If the engineer defines a "satisfy_beh_required" relationship between the two behavior classes, then the relationship is interpreted as a specialization relationship so that a reasoning engine can check the consistency of the specialization relationship. The usage of the inference for product behavior models will be explained more in Section V.

C. StaticPropertyofEntity class

StaticPropertyofEntity has one subclass, **Form**. Forms are properties of **Entity** that do not include time. Only one subclass is defined in this paper, but the class hierarchy of **StaticPropertyofEntity** can be expanded with more subclasses in future work if product development domain requires more static properties of entities.

Form is properties of entities that explain geometry and material aspects of the entities, giving the subclasses **Geometry** and **Material**. **Geometry** describes the measurements of lines, angles, surfaces, solids, and relationships among them. **Material** describes the substances that entities can be made from. The relationship between **Form** and **Entity**, "is_form_of" relationship, is used to specify forms of entities at the M1 level.

D. PropertyofBehavior Class

PropertyofBehavior has one subclass, **TestBehaviorEvaluation**, currently, but the class hierarchy of **PropertyofBehavior** can be expanded with more subclasses in future work. A test behavior of an artifact or specified feature should be compared to a designed behavior. It can satisfy the designed behavior if its observed behaviors at the M0 level satisfy the designed behavior at the M1 level. The evaluation results are recorded with **TestBehaviorEvaluation**. Conformance relationships between the M0 and M1 levels are used to evaluate test behaviors, which will be explained in Section V.

V. Developing Product Models and Instances Using SPMM

Using SPMM at the level M2 will allow engineers to write their product models at the M1 level in SPMM (i.e., interaction between M2 and M1), which provides product-specific syntax developed in SPMM, and adding semantics to the product models (i.e., interaction between M1 and M0). Once engineers define product models using SPMM, they can instantiate their product models and check conformance of the real-world instances to the product models (i.e., interaction between M1 and M0).

Index	*SPMM - OWL expressions*
(a) SPMM axioms example	**Class**: SpecifiedEntity **SubClassOf**: Entity **Class**: Artifact **SubClassOf**: SpecifiedEntity **and** **SubClassOf**: (has_des_behavior **only** DesignedBehavior) **and** **SubClassOf**: (has_des_behavior **some** DesignedBehavior)
(b) Engineer's initial description example	**<spmm:Artifact** rdf:ID= "Motor">
(c) Template example provided by inference systems	**<spmm:Artifact** rdf:ID= "Motor"> **<spmm:has_des_behavior** rdf:ID= "___"> **<spmm:DesignedBehavior** rdf:ID= "___"/> </spmm:has_des_behavior> </spmm:Artifact>

Fig. 6. An example of interactions between the M2 and M1 levels.

A. Developing Product Models (M2-M1 Interactions)

Engineers and inference systems perform the interaction between the M2 and M1 levels. Axioms of SPMM provide OWL syntax for SPMM, and inference systems can assist engineers by telling what classes and relationships need to be defined. Inference systems can also check syntactic consistency of engineers' descriptions at M1 based on the SPMM syntax. For example, let us assume that there are SPMM axioms like those in Fig. 6(a). The axioms are represented in the Manchester OWL syntax [24]. They specify that every artifact at M1 must have at least one relationship "has_des_behavior" with a designed behavior. If an engineer defines a "Motor" artifact class, as in Fig. 6(b), inference systems can check syntactic consistency of the description and tell what information is missing. Then, systems can provide templates for engineers to describe the missing information. The underlined portions of Fig. 6(c) are where engineers fill the missing information.

B. Adding Semantics to Product Models at M1 Layer

If a product model description at M1 satisfies the axioms of SPMM, engineers can add axioms for the product model description at M1. Axioms at the M1 level are defined for each behavior, form, and entity. A behavior can be described with its attributes and sub-behaviors. For example, let us assume that a designed behavior "Rotating a fan" of a motor is described like Fig. 7. A behavior can be differentiated from other behaviors based on its attributes and sub-behaviors. So, the definition of the behavior can be generated from the attributes and sub-behaviors, as in Fig. 8. The contents in Figs. 7 and 8 are shown for explanation, but ideally would be generated automatically using an SPMM editor user interface.

Product model axioms can also be generated automatically by inference systems that have axiom generation rules. Axiom generation rules can be expressed or implemented in various ways. Lee [25] and Liang [26] showed implementation of rules using eXtensible Stylesheet Language Transformations (XSLT) [27] and Java [28], respectively. They find patterns in a product description and generate axioms or definitions for the description. The rules should be designed considering the axioms of SPMM

```
<spmm:DesignedBehavior rdf:ID= "Rotating_a_fan">
  <spmm:specifies>
    <spmm:Artifact rdf:ID= "Motor"/>
    <spmm:ExternalEntity rdf:ID= "Fan" />
  </spmm:specifies>
  <spmm:M1_has_attribute>
      <spmm:Attribute rdf:ID= "torque">
        <spmm:M1_has_value>  [>50, <100]
</spmm:M1_has_value>
        <spmm:M1_has_datatype rdf:Resource= "xsd:integer"/>
        <spmm:M1_has_unit rdf:Resource = "ut:Nm" />
      </spmm:Attribute>
      <spmm:Attribute rdf:ID= "rpm">
        <spmm:M1_has_value>  [>3000, <5000]
</spmm:M1_has_value>
        <spmm:M1_has_datatype rdf:Resource= "xsd:integer"/>
        <spmm:M1_has_unit rdf:Resource = "ut:rpm" />
      </spmm:Attribute>
  </spmm:M1_has_attribute>
  <spmm:has_sub_behavior>
      <spmm:DesignedBehavior rdf:ID= "Receive_electricity"/>
      <spmm:DesignedBehavior rdf:ID= "Spin_axis"/>
  </spmm:has_sub_behavior>
</spmm:DesignedBehavior>
```

Fig. 7. A behavior description example at the M1 level.

Definition	*Note*
spmm:DesignedBehavior: Rotating_a_fan **EquivalentTo**	
(spmm:M0_has_attribute **only** (torque **or** rpm)) **and** (spmm:M0_has_attribute **exactly** 2) **and** (spmm:M0_has_attribute **some** (torque **and** (spmm:M0_has_value **some** int[>50, <100]))) **and** (spmm:M0_has_attribute **some** (rpm **and** (spmm:M0_has_value **some** int[>3000, <5000])))	Axioms for attributes 'torque' and 'rpm'
and (spmm:has_sub_behavior **only** (Receive_electricity **or** Spin_axis)) **and** (spmm:has_sub_behavior **some** Receive_electricity) **and** (spmm:has_sub_behavior **some** Spin_axis)	Axioms for sub-behaviors

Fig. 8. A behavior's definition example at the M1 level.

at the M2 level. Then, axioms at the M1 level can be generated automatically from the sentences defined by engineers. However, axioms generated by rules may not be sufficient to specify all the semantics of product models in M1. Especially when the semantics concern engineering knowledge that is product specific, some axioms must be generated manually by engineers to express the exact meanings of concepts. For example, if engineers know what attributes and sub-behaviors can discriminate the "Rotating_a_fan" behavior from other behaviors, they can select attributes and sub-behaviors for the definition.

Axioms for form and entity descriptions in M1 can also be generated like behavior axioms can. Since entities are described with their behaviors, axioms of entities can reuse the axioms of the behaviors by referring to the behaviors. For example, a "Motor_A" artifact class can be defined as an artifact that must have a designed behavior "Rotating_a_fan." Then, the definition of the motor will refer to the behavior, and the definition of the behaviors will be included in the definition of the motor.

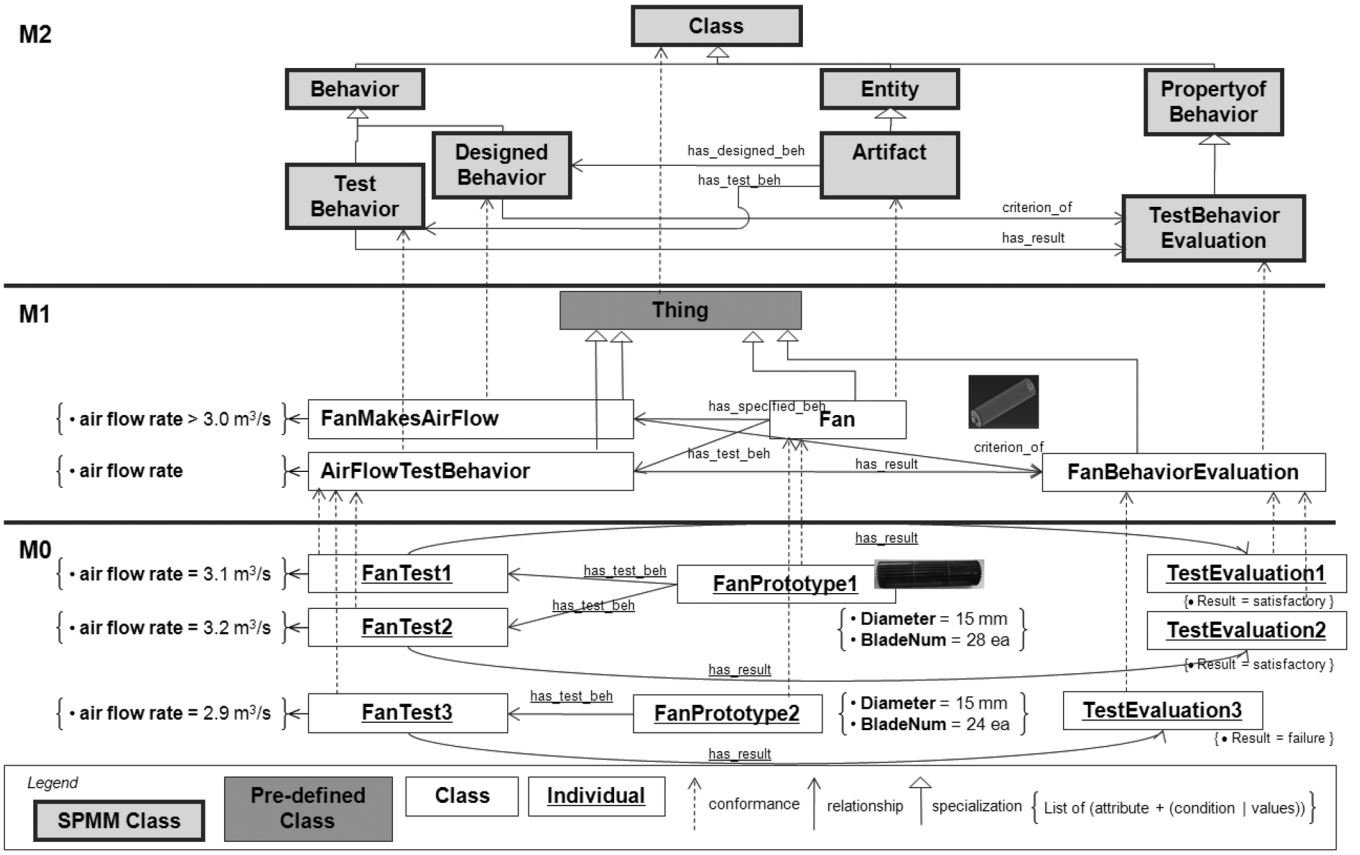


Fig. 9. Behavior evaluation of M0 instances.

Axioms for SPMM relationships also need to be generated and added to the product model descriptions. Some SPMM relationships are interpreted as specialization relationships, such as "satisfy" relationships among behaviors. If a designed behavior satisfies a required behavior, then the attributes and sub-behaviors of the designed behavior should satisfy the attributes and sub-behaviors of the required behavior. A specialization relationship between two classes enables inference systems to verify the "satisfy" relationship between behaviors.

Engineers can also define specialization relationships between behaviors, entities, or forms if they wish to represent the same product at different levels of detail [6]. Then, a "Motor_A" in a detailed design can be a specialization of a "Motor" concept in a conceptual or preliminary design because all physical motors conforming to "Motor_A" should also conform to "Motor," by the definition of specialization. In addition, the specialization relationship can save engineers the effort of defining duplicate axioms at the M1 level. Since specialization implies axiom inheritance, engineers can use specialization relations to reuse existing axioms of concepts. For example, if axioms of a "Motor" artifact exist, a new specialized "Motor_A" concept inherits the axioms, because all physical motors conforming to "Motor_A" should also conform to "Motor" and its axioms. Moreover, the specialization relationship between classes can be inferred by ontological reasoning (i.e., description logic (DL) reasoning). If the axioms of classes at the M1 level are consistent, ontological reasoning can exploit those axioms to find new specialization relationships between classes. For example, if a designed behavior satisfies a required behavior, entities specified by them should also have a specialization relationship. Reasoners can also check if specialization relationships have been added incorrectly. For example, if an engineer says a designed behavior satisfies a required behavior, and adds a specialization relationship to capture this, reasoners can check if the axioms of the designed behavior are consistent with those of the required behavior.

C. Developing Product Instances (M1-M0 Interactions)

Product models are realized in the real-world as physical items. The interaction between the M1 and M0 levels is necessary to build and trace the relationships between product models and physical items. While the M1 level represents different views of a product model, the M0 level represents occurrence or measured information about physical realizations (items) of the model. A physical entity at M0 can be involved in multiple behavior occurrences. While a designed behavior at M1 is invariant once it is specified, behavior occurrence information of physical items at M0 is dependent on its observation time and place. Measurement information may also depend on the accuracy and precision of instruments used, so there can be multiple measurements of the same M0 physical item that gives different values and uncertainties. SPMM does not address multiple measurements of the same item, or measurement uncertainties.

The interaction between the M1 and M0 levels is implemented as a conformance relationship. Conformance

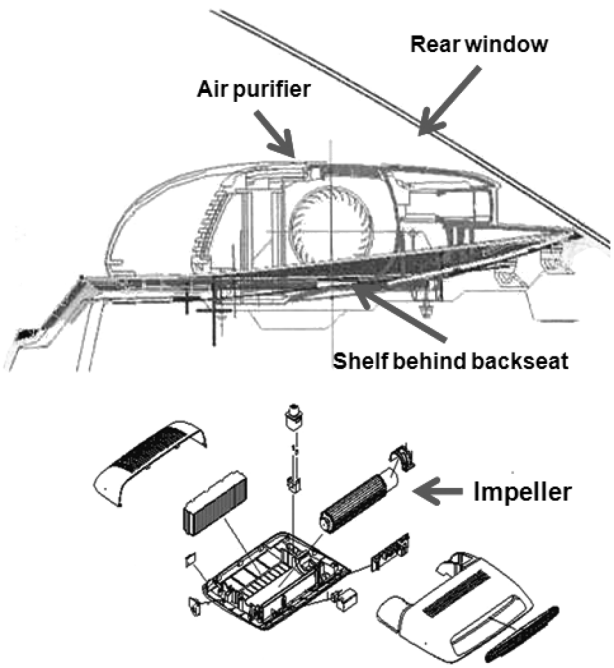


Fig. 10. Air purifier product structure and its installation environment.

relationships between classes at the M1 and instances at the M0 can be established automatically by inference systems if the information pertaining to the instances satisfies the definitions of the classes at the M1 level. In addition, if an instantiation relationship is manually established between a class at the M1 level and an instance at the M0 level, inference systems can check whether the information of the instance satisfies the definition of the class or not.

A conformance relationship can be established between a behavior class and behavior occurrence. A behavior occurrence at the M0 level is used to test (measure) behavior information of a physical item. A behavior occurrence may or may not satisfy a designed behavior. If a behavior occurrence satisfies a definition of a designed behavior, it means that the physical item performs well as designed. Since engineers learn more from failures than successes, tested behaviors and their evaluation results at M0 should be connected to the respective product model at M1. For instance, behavior occurrences (FanTest1 and FanTest2) in Fig. 9 have attributes and their values that satisfy a definition of a designed behavior (FanMakesAirFlow). The occurrences should be inferred to be instances of the designed behavior, and their behavior evaluations have a result attribute whose value is "satisfactory." If a behavior occurrence (FanTest3) has attributes and values that do not satisfy a definition of a designed behavior, then the occurrence should be inferred not to be an instance of the designed behavior, and its behavior evaluation has a result attribute whose value is "failure."

Attributes of a class at M1 and attributes of an individual at M0 are listed in braces and have a relationship to each corresponding class or individual in Fig. 9. Attributes at M1 are instances of the Attribute class in SPMM, and attributes at M0 are instances of the attributes of M1. However, those instantiation relationships are omitted in the figure to simplify the diagram.

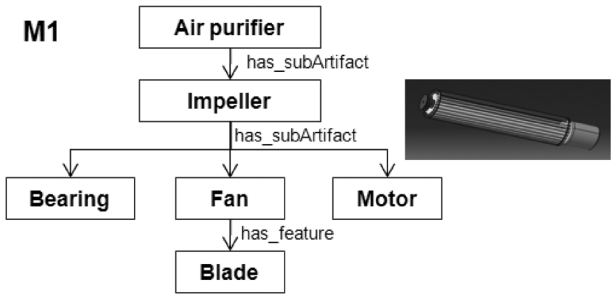


Fig. 11. An impeller structure.

VI. Illustrative Case and Prototype of SPMM

In this section, a product design example is provided to illustrate the concepts in SPMM. In addition, an SPMM editing user interfaces are shown in this section though they are under current development. The scenario used in the example is about developing an impeller assembly, which is a subassembly of an air purifier product, and consists of three components: a fan, motor, and bearing. The air purifier product is an automotive part installed behind a back seat of a car under a rear window, and hence the size of the air purifier is restricted to the space between the rear window and shelf behind the back seat. Fig. 10 shows the air purifier product model and its installation environment. Generally, passengers prefer an air purifier which can clean air quickly and quietly. So, the air purifier should be designed small enough to fit in the given space and move air quickly without much noise. The impeller is a critical subassembly of the air purifier. The air flow-rate of the impeller is determined by the design of its components: fan and motor. A bigger fan can generate more air flow, but the size of the fan is restricted because it should be assembled into the air purifier with other components. Engineers can make several fan design alternatives under the size restriction by changing material, shape, angle, and numbers of blades to satisfy the air flow-rate required for the air purifier and the noise restriction. Fig. 11 shows a product structure of the impeller.

Three actors play roles in the scenario. First, a project manager (PM) coordinates the air purifier design and makes requirements for a fan and motor. Second, a fan designer (FD) makes a fan model to satisfy the requirements given by the PM, including definition of required behaviors for the fan motor. Third, a motor buyer (MB) selects a motor model from a motor catalog, with behaviors satisfying the requirements given by a FD and PM.

The scenario consists of four product design steps: 1) defining requirements of a fan and motor (by PM); 2) making design alternatives of fan (by FD); 3) finding a proper motor model from a market (by MB); and 4) after making a design prototype, testing it and evaluating test results (by FD). These steps are performed with SPMM and are described below.

1) Defining Requirements of a Fan and Motor (by PM): Let us assume that a project manager must develop an air purifier that should be installed in a car. The PM defines required behaviors of the artifact "AirPurifier" informally. In Fig. 12, the PM defines three informal required behaviors ("InformalRequiredBehavior1," "InformalRequiredBehavior2," and "InformalRequiredBehavior3") and one

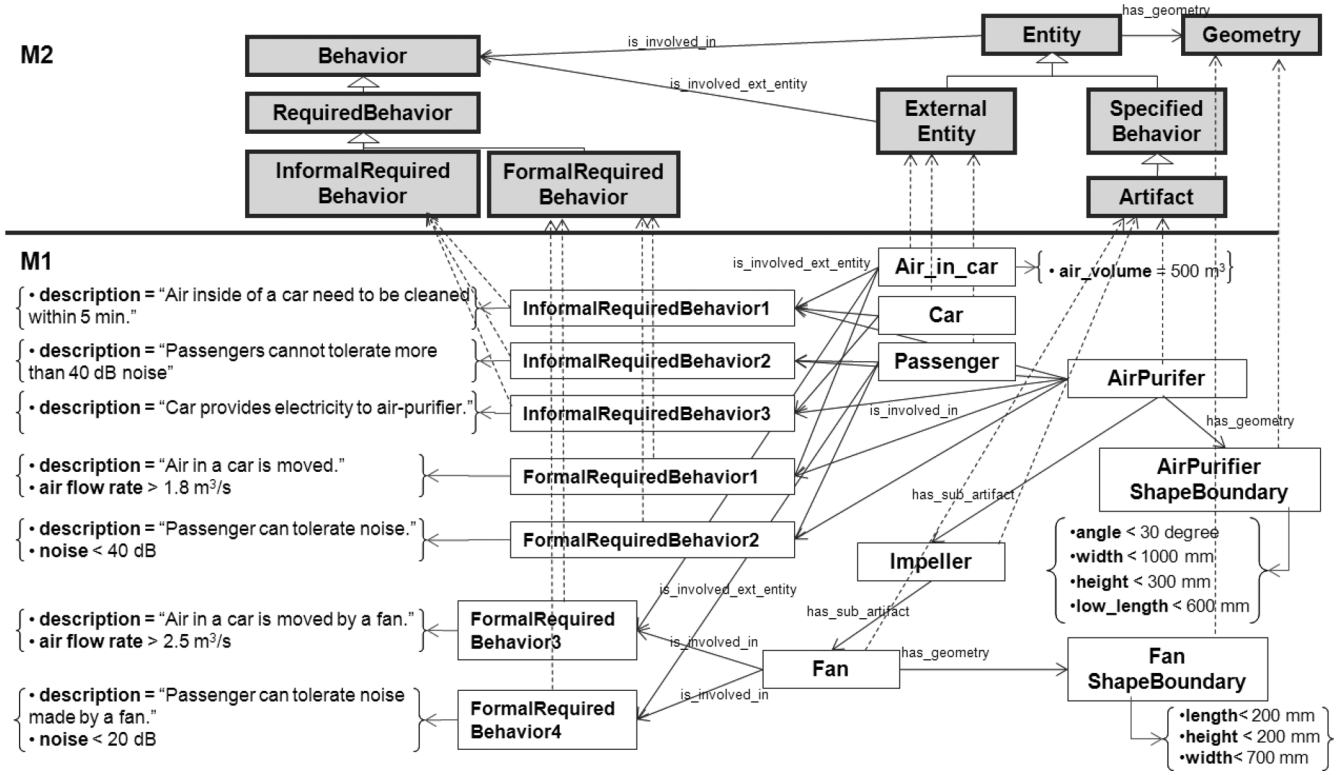


Fig. 12. Defining informal/formal required behavior and geometry for an air purifier.

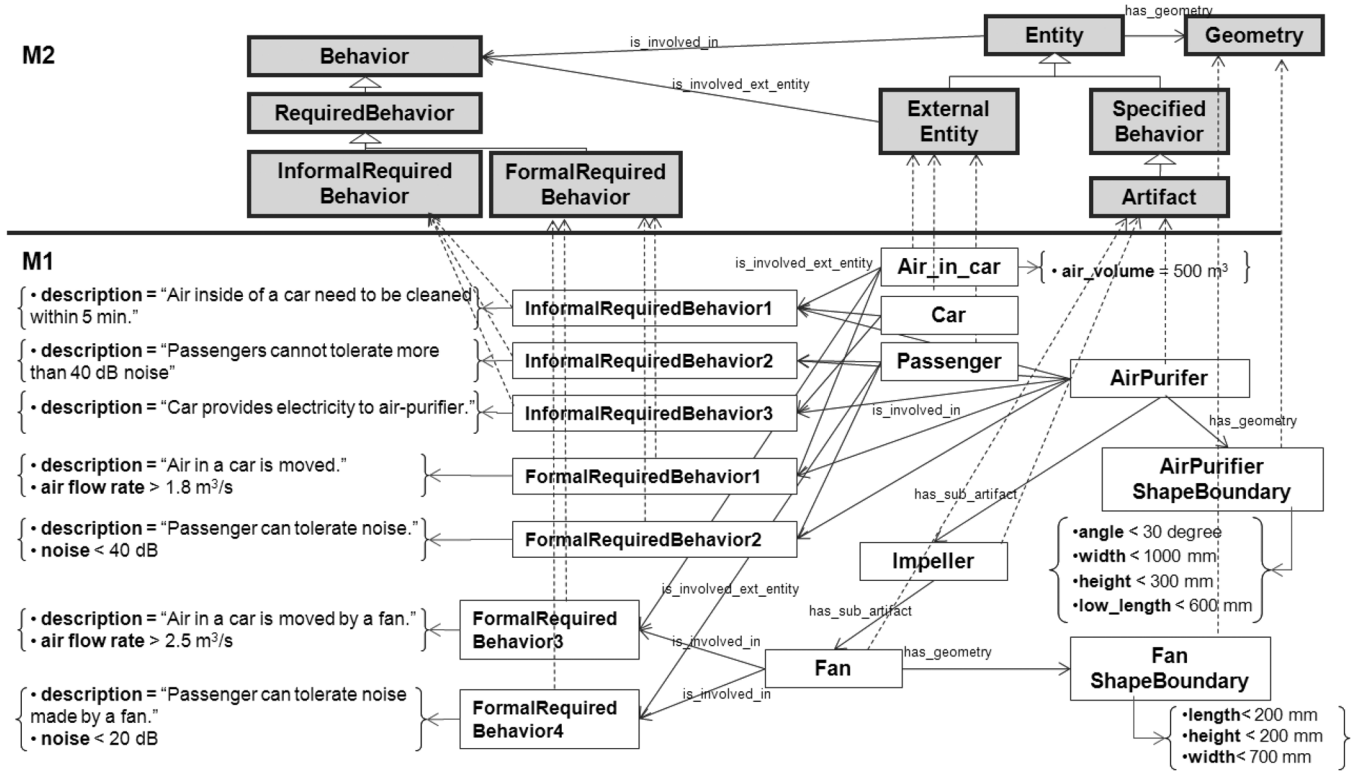


Fig. 13. Defining designed behaviors and forms of a fan.

geometry class ("AirPurifierShapeBoundary") of the artifact. External entities can be also defined from the informal required behaviors.

The PM can define formal required behaviors from the informal required behaviors. The PM can make two formal behaviors like "FormalRequiredBehavior1" and "FormalRequiredBehavior2" in Fig. 12. Each formal behavior has an attribute and value. The PM can define formal required behaviors of subassemblies considering those of the air purifier. The attribute values are assigned by the PM based on domain expertise. For

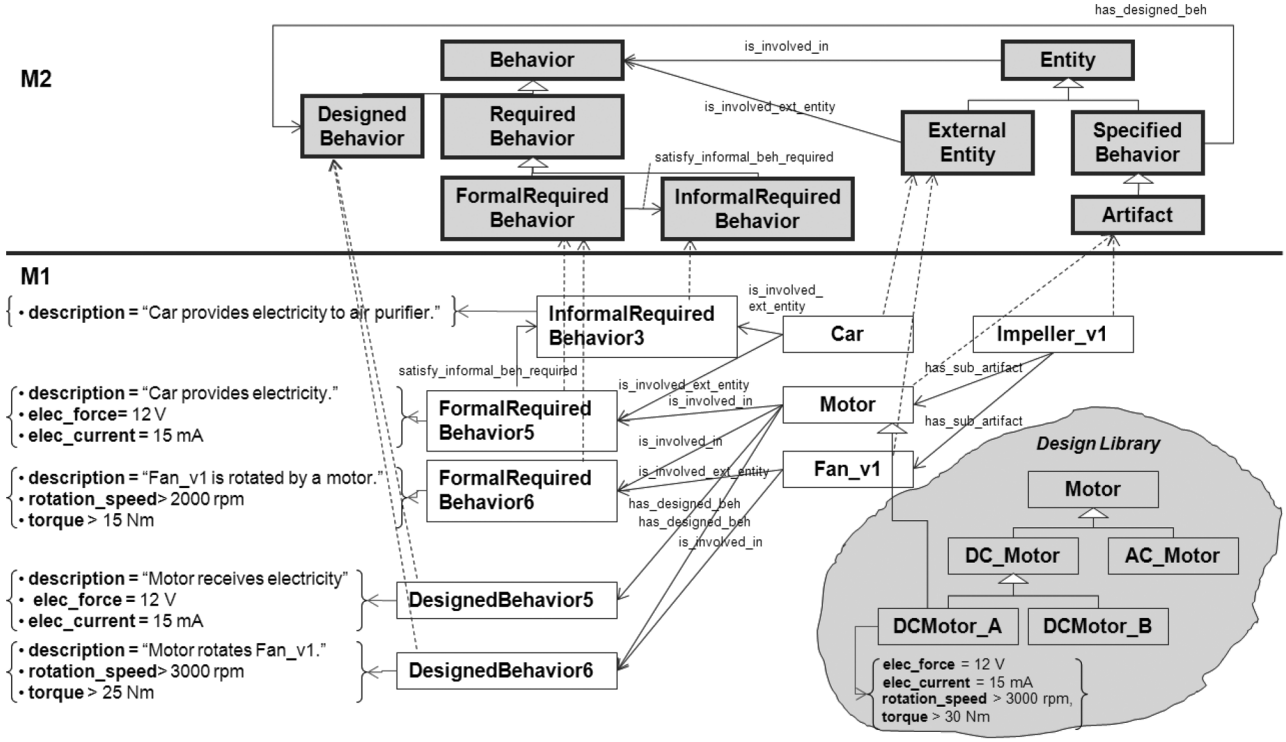


Fig. 14. Reasoning example for interaction between M1 and M0 layer.

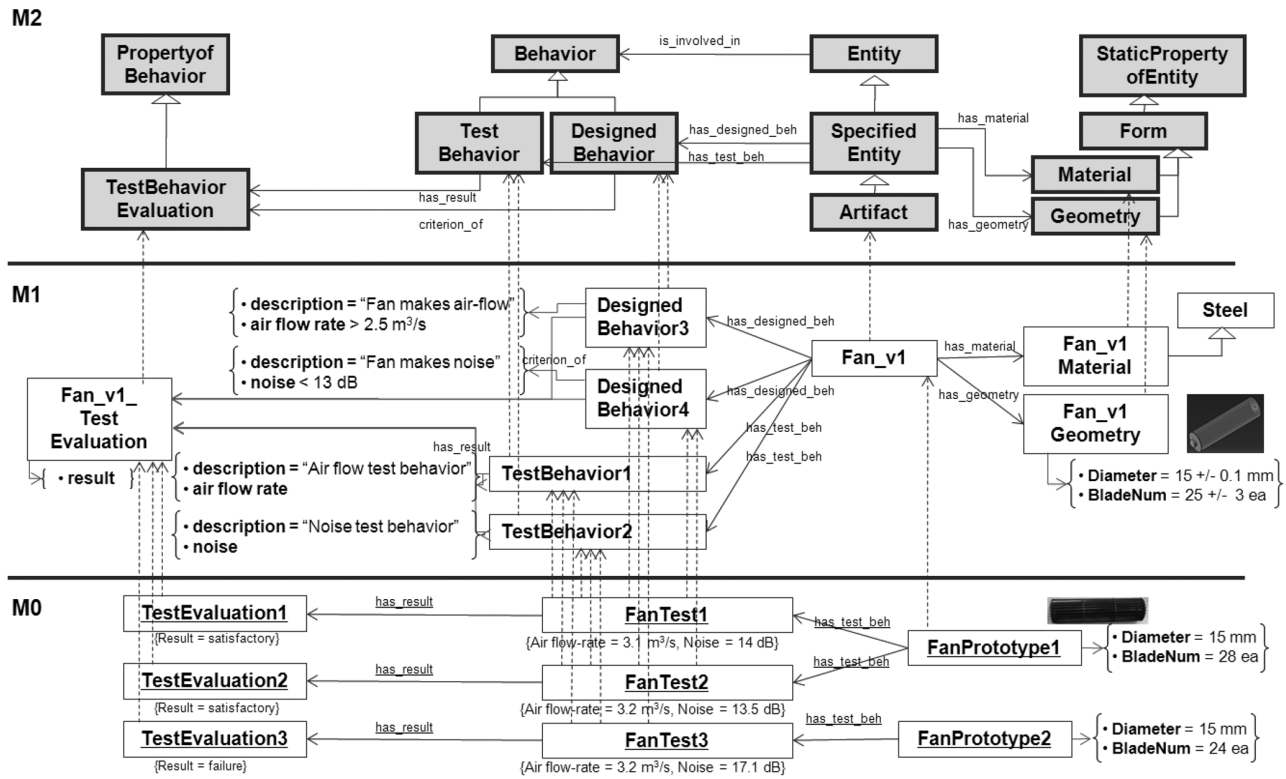


Fig. 15. Reasoning example for interaction between M1 and M0 level.

example, "Fan" needs to make more air flow ("FormalRequiredBehavior3") than the air purifier ("FormalRequiredBehavior1") because there will be loss of air flow when the fan is assembled with other parts such as a fan scroll and air filter. The PM also makes a component layout of the air purifier, and assigns an available space for a fan. The "FanShapeBoundary" geometry in Fig. 12 is a shape boundary of the fan design.

2) Making Design Alternatives of Fan (by FD): A FD can make many fan design alternatives satisfying the required behavior and forms given by a PM. Design alternatives may have

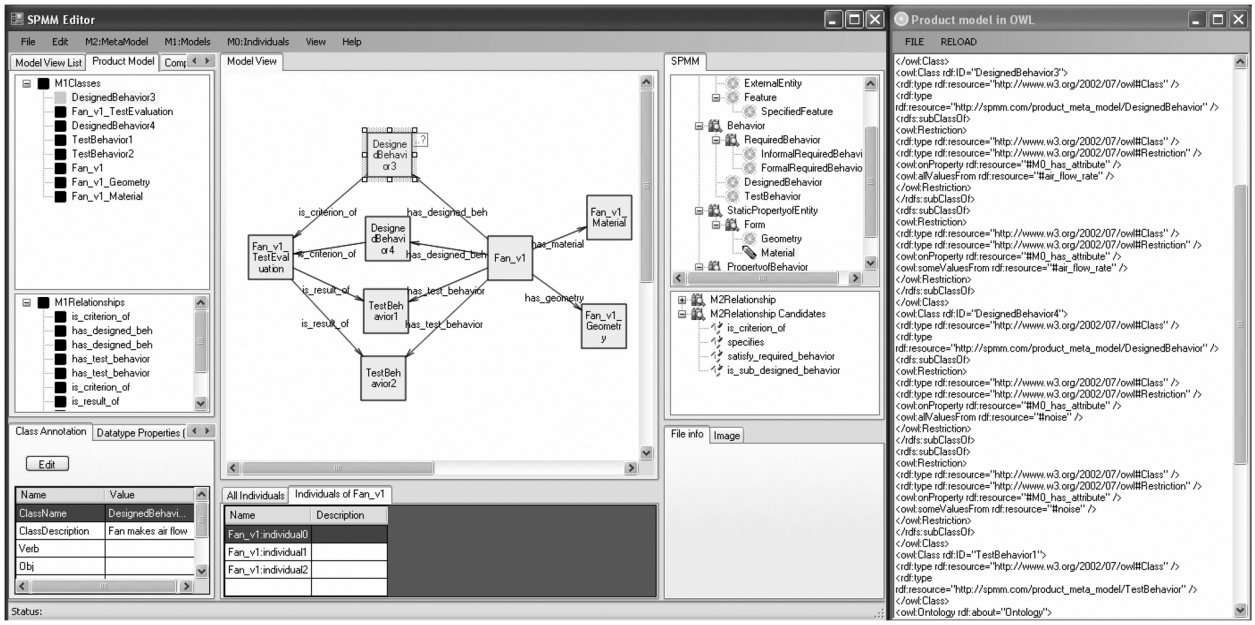


Fig. 16. A screenshot of SPMM user interface.

specific design behaviors or forms that satisfy required behaviors and forms given by a PM. So, each design alternative is defined as a specialization of the "Fan," as shown in Fig. 13. Fig. 13 shows the FD's description about "Fan_v1" artifact that has airfoil blades. The "Fan_v1" artifact has more specific designed behaviors and forms than the "Fan" artifact. So, a specialization relationship can be defined between two artifacts at M1.

3) Selecting a Proper Motor Model (by MB): Let us assume that two formal required behaviors are defined for a motor design.

- FormalRequiredBehavior5
  description: "A car provides electricity to a motor."
  attributes: $\mathrm{elec\_force} = 12\,\mathrm{V}, \mathrm{elec\_current} = 15\,\mathrm{mA}$.
- FormalRequiredBehavior6
  description: "A car provides electricity to a motor."
  attributes: $\mathrm{rotation\_speed} > 2000\ \mathrm{rpm}$, $\mathrm{torque} > 15\ \mathrm{Nm}$.

The "FormalRequiredBehavior5" can be defined by a fan designer after "Fan_v1" is designed. The "FormalRequiredBehavior5" is defined to satisfy the "InformalRequiredBehavior3."

Then, a MB can define specific designed behaviors that satisfy the two formal required behaviors. Fig. 14 shows two designed behaviors defined by a MB ("DesignedBehavior5" and "DesignedBehavior6"). If designed behaviors of a motor are defined, a definition of the motor can be generated as explained in Section V. The definition of a motor can be used to search a motor design library. Motor models in the design library should also have their definitions with their behaviors including specific attributes. Fig. 14 shows an example of inference result of searching a motor design library.

4) Testing a Prototype and Evaluating Test Results (by FD): After the MB selects a motor, the FD makes a prototype of the fan design model and tests its behaviors to validate the fan design model. Fig. 15 shows an example of testing a fan prototype. The "FanPrototype1" and "FanPrototype2" in the figure are physical prototype fans. The former has two tested behavior occurrences ("FanTest1" and "FanTest2"), and the latter has a tested behavior occurrence ("FanTest3"). The behavior occurrences measure air flow-rate and noise of the prototype fans. Each tested behavior occurrence can be evaluated, and its evaluation result is recorded. Fig. 15 shows the behavior evaluation results of the fan prototypes. The "FanPrototype1" item at can be inferred as an instance of "Fan_v1" model because its form-related attributes and behavior occurrences satisfy the form and designed behaviors of "Fan_v1" artifact at M1. However, the "FanPrototype2" item at M0 cannot be an instance of "Fan_v1" model because its behavior occurrence does not satisfy a designed behavior ("DesignedBehavior4") at M1.

Fig. 16 shows the graphical user interface for the SPMM editor and OWL generator. The two tree-views on the right side of the SPMM editor show SPMM classes and relationships at the M2 level. The two tree-views on the left side of the SPMM editor show the list of classes and relationships defined by users at the M1 level. The graphical map in the center of the editor is where users can build their product models graphically using the provided M2 classes and relationships. The grid-view in the bottom-left of the editor shows descriptions and attributes of the M1 class selected by the user. The bottom-middle section in the editor includes multiple tabs, and each tab shows a list of M0 individuals of the selected M1 class, with their attributes and values. The number of tabs can be increased as users create individuals for other M1 classes. The SPMM editor implements an inheritance mechanism at the M1 level so that attributes of a superclass are inherited by its subclasses. The attributes of an M1 class are also used to create attributes of M0 individuals. The interface enables users to manage both product models at M1 and their individuals at M0 in one place.

The OWL generator is shown at the right side of Fig. 16. It takes product model and individual information from the SPMM

editor, and then creates an OWL file. The product model information in the OWL file can be shared or modified in other applications.

VII. Conclusion

In this paper, a multilevel product modeling framework is proposed. The framework has three levels of product information for metamodels (M2), models (M1), and individuals (M0). The semantic-based product metamodel (SPMM) is proposed to include specific classes and relationships to define and model product design and manufacturing. Classes such as "Behavior," "Artifact," etc. are defined in OWL (M2 level), so that the semantics of OWL can be exploited by designers and engineers. This enables designers and engineers to check whether particular entities are behavior, artifact, or other classes in SPMM. It is also to be noted that the axioms that define "Behavior," "ExternalEntity," and "Artifact" can be extended for specific needs, thus supporting extensions to SPMM.

The proposed framework needs additional work and implementation to be usable as a product modeling system. The SPMM editor interface allows description of product models using methods and terminology familiar to engineers, hiding from them the underlying formal logic-based representation of the product models. In addition, user interfaces for integration with applications such as design knowledge bases and computer-aided design (CAD) systems could be implemented. Then, SPMM can be a powerful mechanism for annotating CAD models, so that the CAD models can be semantically enriched with information beyond geometry such as requirement, function, and behavior.

In terms of information modeling scope, SPMM currently focuses on product structure and behavior evaluation within a product lifecycle. Additional information modeling issues need to be addressed such as version control, product configuration, assembly relationship, tolerance, and sustainability.

Acknowledgment

The authors would like to acknowledge the comments and suggestions of the reviewers. They also would like to acknowledge the NIST colleagues J. Lubell and P. Witherell in improving the paper, but any residual mistakes remain ours.

References

[1] R. D. Sriram and S. Szykman, "The NIST design repository project: Project overview and implementation design," National Institute of Standards and Technology (NIST), Gaithersburg, MD, NIST Interagency/Internal Rep. (NISTIR) 6926, 2002.

[2] S. Szykman, S. J. Fenves, W. Keirouz, and S. B. Shooter, "A foundation for interoperability in next-generation product development systems," *Comput.-Aided Design*, vol. 33, pp. 549–559, 2001.

[3] V. Srinivasan, "An integration framework for product lifecycle management," *Comput.-Aided Design*, vol. 43, no. 5, pp. 464–478, 2011.

[4] G. T. Nguyen, D. Rieu, and J. Escamilla, "An object model for engineering design," in *Proc. Eur. Conf. Object-Oriented Programming, ECOOP'92*, 1992, vol. 615, LNCS, pp. 233–251.

[5] T. Gruber, "Toward principles for the design of ontologies used for knowledge sharing," *Int. J. Human-Comput. Studies*, vol. 43, pp. 907–928, 1995.

[6] C. Bock, X. F. Zha, H. W. Suh, and J. H. Lee, "Ontological product modeling for collaborative design," *Adv. Eng. Inform.*, vol. 24, pp. 510–524, 2010.

[7] S. J. Fenves, S. Foufou, C. Bock, S. Rachuri, N. Bouillon, and R. D. Sriram, "CPM 2: A revised core product model for representing design information," National Institute of Standards and Technology (NIST), Gaithersburg, MD, NIST Interagency/Internal Rep. (NISTIR) 7185, 2005.

[8] J. Lee, H. Chae, C. H. Kim, and K. Kim, "Design of product ontology architecture for collaborative enterprises," *Expert Syst. With Appl.*, vol. 36, pp. 2300–2309, 2008.

[9] Object Management Group, "Model-driven architecture," 2007. [Online]. Available: http://www.omg.org/mda

[10] I. Niles and A. Pease, "Towards a standard upper ontology," in *Proc. 2nd Int. Conf. Formal Ontology Inform. Syst., FOIS'01*, Ogunquit, ME, 2001, pp. 2–9.

[11] J. H. Lee and H. W. Suh, "Ontology-based multi-layered knowledge framework for product lifecycle management," *Concurrent Eng. Res. Appl.*, vol. 16, pp. 301–311, 2009.

[12] D. Leal, "ISO 15926 life cycle data for process plant: An overview," *Oil and Gas Sci. Technol.*, vol. 60, pp. 629–638, 2005.

[13] D. Yang, M. Dong, and R. Miao, "Development of a product configuration system with an ontology-based approach," *Comput.-Aided Design*, vol. 40, no. 8, pp. 863–878, 2008.

[14] R. J. Brachman and H. J. Levesque, *Knowledge Representation and Reasoning*. New York: Elsevier, 2004, ch. 2.

[15] R. J. Brachman and H. J. Levesque, *Knowledge Representation and Reasoning*. New York: Elsevier, 2004, ch. 9.

[16] D. L. McGuinness and F. V. Harmelen, 2004, "OWL web ontology language overview W3C," Recommendation, Feb., 10, 2004. [Online]. Available: http://www.w3.org/TR/owl-features/

[17] Object Management Group, "Unified modeling language: Infrastructure." [Online]. Available: http://doc.omg.org/formal/2010-05-03 2010

[18] A. Taivalsaari, "On the Notion of Inheritance," *ACM Comput. Surveys*, vol. 28, pp. 438–479, 1996.

[19] D. Berardi, D. Calvanese, and G. De Giacomo, "Reasoning on UML class diagrams," *Artif. Intell.*, vol. 168, no. 1, pp. 70–118, 2005.

[20] A. Weissman, S. K. Gupta, X. Fiorentini, S. Rachuri, and R. D. Sriram, "Formal representation of product design specifications for validating product designs," National Institute of Standards and Technology (NIST), Gaithersburg, MD, NIST Interagency/Internal Rep. (NISTIR) 7626, 2009.

[21] J. Hirtz, R. B. Stone, D. A. McAdams, S. Szykman, and K. L. Wood, "A functional basis for engineering design: Reconciling and evolving previous efforts," *Research in Engineering Design*, vol. 13, pp. 65–82, 2002.

[22] Y. Kitamura, "A functional concept ontology and its application to automatic identification of functional structures," *Adv. Eng. Inform.*, vol. 16, no. 2, pp. 145–163, 2002.

[23] A. C. B. Garcia, J. Kunz, M. Ekstrom, and A. Kiviniemi, "Building a project ontology with extreme collaboration and virtual design and construction," *Adv. Eng. Inform.*, vol. 18, no. 2, pp. 71–83, 2004.

[24] M. Horridge and P. F. Patel-Schneide, "OWL 2 Web Ontology Language Manchester Syntax," W3C Working Group Note 27, Oct. 2009 [Online]. Available: http://www.w3.org/TR/owl2-manchester-syntax/

[25] J. H. Lee and H. W. Suh, "OWL-based product ontology (POWL) architecture and representation for sharing product knowledge on a web," in *Proc. ASME 2007 Int. Design Eng. Tech. Conf. Comput. Inform. Eng. Conf. (IDETC/CIE)*, Las Vegas, NV, 2007, pp. 853–861.

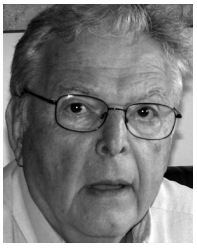
[26] V. C. Liang, C. Bock, and X. F. Zha, "An Ontological Modeling Platform National Institute of Standards and Technology (NIST), Gaithersburg, MD, NIST Interagency/Internal Rep. (NISTIR) 7509, 2008.

[27] J. Clark, 1999, "XSL Transformations (XSLT)," W3C, Recommendation, Nov. 16, 1999, ver. 1.0. [Online]. Available: http://www.w3.org/TR/xslt

[28] J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java Language Specification*, 3rd ed. Reading, MA: Addison-Wesley, 2005.

Jae Hyun Lee received the M.S. and Ph.D. degrees in industrial engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, in 2001 and 2008, respectively.

He has been an Associate Researcher since 2008 in the Systems Integration Division, Engineering Laboratory, National Institute of Standards and Technology (NIST), Gaithersburg, MD. He has worked on developing information models for sustainable product design, manufacturing, and environmental declarations.



Steven J. Fenves received the B.S., M.S., and Ph.D. degrees from the University of Illinoisat Urbana–Champaign, Urbana.

He taught at the University of Illinois from 1958 to 1971 and at Carnegie Mellon University, Pittsburgh, PA, from 1972 to 1998.

He is University Professor Emeritus of Civil and Environmental Engineering at Carnegie Mellon University. He is the author of six books and over 400 papers. His research and teaching dealt with design data modeling, design standards, knowledge-based systems, and structural analysis. He was a Guest Researcher at the National Institute of Standards and Technology (NIST), Gaithersburg, MD, from 1999 to 2009.

Dr. Fenves has coauthored with NIST co-workers and have won the Best Paper Award and recognitions as one of the most influential papers and first in citations.

Conrad Bock received the B.S. and M.S. degrees in physics and computer science from Stanford University, Stanford, CA.

He is a Computer Scientist at the Engineering Laboratory, National Institute of Standards and Technology (NIST), Gaithersburg, MD, specializing in formal product and process modeling. He was the founding editor for the Activity and Action models in the Unified Modeling Language and Systems Modeling Language at the Object Management Group, as well as a primary contributor to interaction modeling in the Business Process Model and Notation.

Hyo-Won Suh received the B.S. degree in mechanical engineering from Yonsei University, Seoul, Korea, in 1981, the M.S. degree in mechanical engineering from the Korea Institute of Science and Technology (KAIST), Seoul, in 1983, and the Ph.D. degree in industrial engineering from West Virginia University, Morgantown, in 1991.

Currently, he is a Professor with the Department of Industrial Engineering, KAIST, Daeajeon, Korea. His research interests are product lifecycle management (PLM), ontology application to engineering and he has published several papers in these areas.

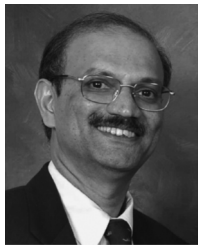
Sudarsan Rachuri received the M.S. and Ph.D. degrees from the Indian Institute of Science, Bangalore.

He is a Computer Scientist in the Systems Integration Division, Engineering Laboratory, National Institute of Standards and Technology (NIST), Gaithersburg, MD. He is the regional editor (North America) for the *International Journal of Product Development, and Associate Editor for the International Journal of Product Lifecycle Management*. His primary objectives at NIST are to develop and transfer knowledge to industry about standards and information models for sustainable manufacturing, green products, assembly representation, system level analysis, and tolerance representation.

Dr. Rachuri is a member of ASME Y14.5.1.

Xenia Fiorentini received the M.Sc. degree in industrial engineering from the Systems Engineering College, Politecnico di Milano, Milan, Italy, and the Diploma degree from the Alta Scuola Politecnica, Milan, a school for the management of complex innovation projects.

She is one of the founders of Engisis, a company that supports the adoption of international standards for product data exchange and business collaboration. She serves as Consultant in the areas of integrated logistic support, systems interoperability, and standards application. Before founding Engisis, she worked for four years as a Guest Researcher at the National Institute of Standards and Technology (NIST), Gaithersburg,b MD.



Ram D. Sriram (M'85–SM'02) received the B.Tech. degree from the Indian Institute of Technology, Madras, and the M.S. and Ph.D. degrees from Carnegie Mellon University, Pittsburgh, PA.

He is currently the Chief of the Software and Systems Division, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD. Before joining the Software and Systems Division, he was the leader of the Design and Process Group, Manufacturing Systems Integration Division, Manufacturing Engineering Laboratory. Prior to joining NIST, he was on the Engineering Faculty of the Massachusetts Institute of Technology (1986–1994). He has coauthored or authored nearly 250 publications, including several books.