Product Modeling Framework and Language for 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's entire lifecycle requires semantic richness for representing product related information. This paper proposes a multi-layered product-modeling framework that enables stakeholders to define their product-specific models and relate product-specific models to physical or simulated instances. The framework is defined within the Model-driven Architecture and adapted to the multi-layer approach of the architecture. The data layer represents real world products, the model layer includes models of those products, and the meta-model layer defines the product modeling language. The semantic-based product modeling language described in this paper is specialized from a web ontology language enabling product designers to express the semantics of their product models explicitly and logically in an engineering-friendly way. The interactions between these three layers are described to illustrate how each layer in the framework is used in a product engineering context.**

I. Introduction

Efficient collaboration is essential when products are designed by temporally and spatially separated engineers. Collaborative environments enable product designers to interact and reach agreement by sharing design knowledge and product information [1], [2]. Ontology can play a role in the environments as a shared product information model because ontology is a formal and explicit specification of a shared conceptualization [3]. The collaborative environment needs the support of a generic product modeling language that: (1) can be readily specialized for the products at hand; (2) can provide information to all stakeholders throughout the product lifecycle; and (3) provides explicit, logical semantics of the concepts and relationships involved, without requiring that the stakeholders be versed in ontological thinking.

Ontological multi-layered product modeling frameworks provide the above capabilities [4]. Figure 1 shows a high level view of the framework. Engineers can describe their product models using a given semantic product modeling language. An editor interface needs to be developed for engineers to describe their product models without understanding the language. The product model descriptions need to be converted into formal descriptions with axioms so that reasoners can check the consistency of the formal descriptions and infer new knowledge based on the descriptions and their instances, i.e., information about physical products. In order to develop the editor interface, converter, and reasoners, the syntax and semantics of the product modeling language need to be defined. In this paper, the structure of the modeling framework and the abstract syntax and semantics of the product modeling language are proposed. Developing the concrete syntax for an editor interface, converter, and reasoners are out of scope for this paper.

The framework includes a generic product modeling language. The language enables product designers to define their product models from an engineering point of view while exploiting the benefits of ontology languages. The product modeling language consists of generic product domain concepts and relationships, such as artifact, behavior, and their relationships (explained later in the paper), which can guide designers in building their product-specific models, such as models of cars, airplanes, ships, etc. The language extends existing work on ontological product modeling languages [4] to support product model verification as in the Core Product Model 2 (CPM2), a product modeling language, which also contains generic product concepts and relationships [5].

The product modeling language is specialized from an

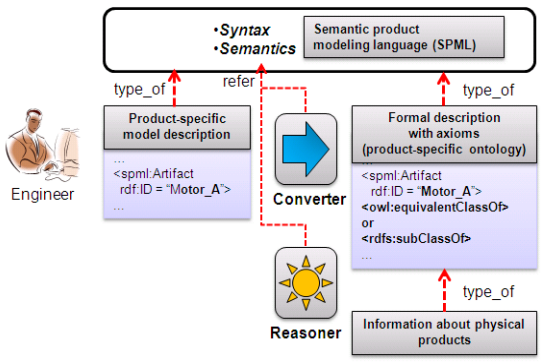


Fig. 1. A high level view of the product modeling framework.

Manuscript received March 31, 2010.

Jae H. Lee*, Conrad Bock, Sudarsan Rachuri, Ram D. Sriram are with the National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg, MD 20899 USA (*corresponding author to provide phone: 301-975-3167; fax: 301-975-4635; e-mail: {lee.jaehyun, conrad.bock, sudarsan, sriram}@nist.gov).

Steven J. Fenves was with the National Institute of Standards and Technology, Gaithersburg, MD 20878 USA. (e-mail: fenvessj@comcast.net).

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

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

ontology language, while product models are represented as instances of the language. Thereby, product designers can share and understand the semantics of the product information and design knowledge. In addition, the framework clarifies the different meanings of the product information modeling layers, so that: (1) the generic product model is used as the product modeling language; (2) product-specific models can be represented in the product modeling language; and (3) information about physical products can be represented as instances of the product-specific models.

This paper is organized as follows. Section 2 reviews previous research on product information models and ontological multi-layered product modeling architectures. Section 3 presents an overview of the proposed multi-layered product modeling framework, including the generic product model. Section 4 addresses the concepts and relationships in the integrated generic product model, and their semantics. Section 5 describes the interaction mechanisms within the multi-layered product modeling framework. Section 6 gives suggestions for future work and also contains the conclusions of this paper.

II. Previous Research

A. Multi-layered product modeling framework

Ontological representations without multiple layers of abstraction could not integrate product lifecycle information at different abstraction levels. This need gave rise to multi-layered approaches for integrated models of product lifecycle information. Research clarified the difference between a generic product ontology and product-specific ontologies, and the multi-layered architecture provided the additional instance layers under the product-specific ontology.

Ontology representations, such as first-order logic, description logic, and the web ontology language (OWL), have been widely used for representing formal descriptions of information models. However, it is a heavy burden on a product designer to understand these formalisms and to be able to use them to represent product models. Therefore, specialized product modeling languages are necessary so that product designers can build and use their product models on top of generic product domain concepts that are independent of specific product domains. Collaborative design environments also need product-specific ontologies, namely, sets of concepts, relationships, and constraints related to a specific product domain such as ‘car,’ ‘axle,’ or ‘bolt,’ defined using a specialized product modeling language. The resulting product-specific ontologies have instances, which have measured values of the design parameters. Finally, collaborative product design environments need integrated semantics of both the generic product modeling language and the product-specific ontologies.

The previous efforts in developing product ontology using multi-layer approach [7-11] did not provide a modeling language for engineers. Although they exploited ontological reasoning capability in their models, they did not explain how engineers can use the modeling language and exploit the reasoning capability.

B. Related NIST efforts

Significant work in product modeling languages has been developed by the Design Process Group at National Institute of Standards and Technology (NIST). In particular, Core Product Model (CPM) [12], CPM2 [5], Open Assembly Model (OAM) [13], and ontological product modeling language (OPML) [4], provide product information models with generic terms such as ‘artifact,’ ‘behavior,’ and ‘form.’ CPM and OPML were developed to represent a generic product modeling language, while OAM is an information model specialized from CPM2 to express geometric constraints, kinematics, and tolerance in assemblies. The current effort augments OPML with additional aspects of CPM2, in particular the evaluation of product behavior.

III. Multi-layered Product Information Modeling Framework

The proposed framework contains three layers for ontological product modeling, namely: the ontological product modeling language (M2), product-specific ontological models (M1), and physical item information (M0). Figure 2 shows the three layers and the relationships among them.

The first layer (M2) is the modeling language. A semantics-based product modeling language (SPML) is proposed at this layer. As shown in Figure 2, the concepts and relationships in SPML are specialized from OWL classes, and the syntax and semantics of SPML inherit the syntax and semantics of OWL. The semantics of SPML need to be specified further because its classes and relationships require more specialized semantics for the domain of products. Since OWL can be a meta-modeling language to express semantics of SPML at M2, OWL is used to represent the axioms explicitly. For example, ‘Artifact’ and ‘Designed Behavior’ classes in SPML have different axioms. Although both classes are sub-classes of owl:Class , only the ‘Artifact’ class has an axiom that it must have at least one relationship with ‘Designed Behavior.’ Specifically, the semantics of ‘Artifact’ constrains an artifact to have at least one ‘Designed Behavior.’

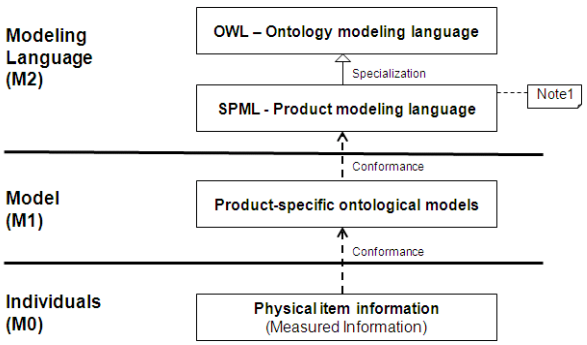
The second layer (M1) is for product-specific ontological models. A product-specific model at M1 holds representative information about a particular product. It consists of product-specific concepts and relationships such as artifacts, behaviors, forms, and structures of a specific product, including attributes and their (required or designed) values. The concepts and relationships are defined by product designers, as instances of SPML, so that they can be checked by a reasoner to determine their conformity to the axioms in SPML (this is shown in the figure as Conformance). For instance, if a ‘Motor’ concept is defined as an instance of the ‘Artifact’ class and it does not have a relationship with any

designed-behavior, a reasoner can find inconsistencies by checking the above axiom of the 'Artifact' class. If the concepts and relationships at M1 do not create any inconsistency with SPML, they are valid, and their axioms can be added or inferred. Most axioms at M1 can be defined by domain experts, but some of the axioms can be inferred from axioms of the product modeling language at M2.

The bottom layer (M0) is for physical product information. The information at M0 represents information about physical products observed at a certain time or space, or simulations of these. The information can be different from the product designers' intent or expectation. Physical product information needs to be related to the original design models, which may include requirements as well as product designers' intent and plan. For example, a physical product in M0 needs to be related to the maintenance policy and geometry information described in the design model in M1.

The relationship between M0 and M1 is particularly important when the lifetime of the manufactured product is so long that information on the physical product may change several times, as in the case of ships, aircraft, and buildings. The conformance relationship in ontology modeling is better at expressing this relationship than the instantiation relationship in object-oriented modeling. The conformance relationship is a relationship between an individual and a class in an ontological modeling perspective. Individuals can exist without their specific classes, so their attributes are not dependent on specific classes. A conformance relation can be established between an individual and a class by an inference engine if the information declared for the individual satisfies the definition of the class. There are other terms such as 'instance of' and 'type of' (used interchangeably in the literature). In this paper, these terms are distinguished with the term 'conforms to' to mean that the instances are checked logically with reasoners to be consistent with the class's definitions. For example, an individual can be defined by engineers as an instance of a class but it may or may not be conformed to the class.

A physical product 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 product has detailed information, it can have more attributes than one of its product models at M1. Definitions at the M1 layer can be used to infer the conformance relationships between the M1 and M0 layers. A detailed description of the interaction between the M1 and M0 layers will be given in Section 5.



Note1: The SPML at M2 uses OWL as a meta-modeling language to express its axioms.

Fig. 2. The multi-layered product information modeling framework.

IV. The Product Modeling Language: SPML

This section describes the classes and relationships of the product modeling language at the M2 layer, in short, the SPML classes. SPML uses OWL for its base language. SPML inherits OWL semantics through the specialization relationship. In addition, SPML classes have axioms to specify their meanings explicitly. The axioms are also described in OWL, so that they can provide OWL syntax for SPML to describe specific product models at M1.

Figure 3 shows the relationships between the SPML classes and the OWL primitives. The 'spml:Class' is the top

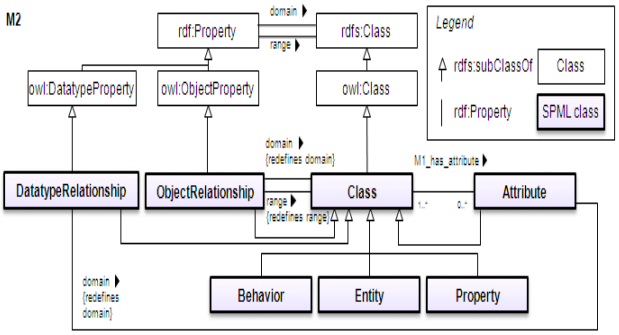


Fig. 3. Relationships between OWL and SPML classes.

class in SPML, and it is a sub-class of 'owl:Class.' 'spml:Entity,' 'spml:Behavior,' and 'spml:Property' classes will be defined in this section. From this section, the "Arial" bold font is used to avoid repeating usage of 'spml:' prefix. For example, **Class** means spml:Class.

Entity, Behavior, and Property classes

Entity, **Behavior**, and **Property** are crucial concepts for building product-specific models. They have relationships among themselves, and have further sub-classes which inherit the relationships. Figure 4 shows the class hierarchy and relationships graphically.

Entity is things that can be described with **Behavior** and **Property**. **Entity** can be involved in some **Behavior**. **Behavior** is a dynamic aspect of one or more **Entities**. A dynamic aspect involves the notion of time in its description. For example, 'motor' and 'fan' are entities involved in a behavior 'rotating a fan.'

Property represents things that describe **Class**, excluding the dynamics of the **Class** represented by **Behavior**. **Property** must describe either **Entity** or **Behavior** so instances of it cannot exist without at least one 'is_property_of' relationship to an instance of either **Entity** or **Behavior**. **Property** is different from **Attribute** because properties can be further described by attributes while **Attribute** cannot have attributes. For example, 'motor cylinder' property (geometry) can describe as a 'motor' with its attributes such as 'diameter' and 'length.' However,

attributes such as ‘diameter’ or ‘‘length’ cannot be further described by other attributes in SPML.

Figure 4 shows subclasses of **Entity**, **Behavior**, and **Property**, and their relationships.

A. Entity class

Entity is divided into **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 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 give required behaviors to the water-bottle design.

SpecifiedEntity is a kind of **Entity** that is specified with **SpecifiedBehavior** and **Form**. **Artifact** and **Feature** are sub-classes of **SpecifiedEntity**. A specified entity must be involved in at least one specified behavior (either required behavior or designed behavior) and have at least one form. 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 must be involved in at least one specified behavior. **Artifact** and **ExternalEntity** are not disjoint because an artifact can be an external entity for another artifact. For instance, a ‘desk’ can be an external entity for a ‘chair’ design, while a ‘chair’ can be an external entity for a ‘desk.’

Feature is a kind of **SpecifiedEntity** which is a portion of an artifact. A feature has relationships with specific forms and behaviors of its artifact to specify relationships between the specific forms and behaviors. For example, a ‘bottle-neck’ feature of a ‘water-bottle’ can be a feature the form of which is a ‘through-hole’, and the designed behavior of which is ‘guiding water-flow.’ The relationship between **Artifact** and **Feature** is defined as a composite relationship because they have a structural relationship and a feature cannot exist without an artifact.

B. Behavior class

Behaviors are classified into two direct sub-classes: **SpecifiedBehavior** and **TestBehavior**.

SpecifiedBehavior is a kind of **Behavior** specifies dynamic aspects of entities, which are invariant once they are specified. It is further specialized into **RequiredBehavior** and **DesignedBehavior**.

Required behaviors describe external entities’ dynamics, which affect artifact design. Required behaviors are classified into two classes.

- **InformalRequiredBehavior** is a behavioral description of the dynamics of the external entities surrounding artifacts. The description explains behaviors of the external entities informally, and provides initial data for the analysis of interactions between external entities and artifacts. For example, ‘Person is drinking water’ is an example of informal description. A designer can start to think about an artifact like ‘water-bottle’ and its interactions with external entities, such as ‘person’ and ‘water’, based on the description. A required behavior can have sub-behaviors. Sub-behaviors are described with more specific entities or verbs. For example, the above example can have sub behaviors: ‘Hand tilts a water-bottle’ and ‘Mouth contacts a water-bottle.’
- **FormalRequiredBehavior** explicitly specifies external entities, artifacts, and their interactions. A formal required behavior can be derived from an informal required behavior. While an informal required behavior is expressed with a sentence, a formal required behavior is expressed with a structured sentence and attributes. A structured sentence consists of a subject, verb, and object [14]. 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 taxonomy is required to specify verbs. For instance, Hirtz [15] and Kitamura [16] defined verb taxonomies to describe engineering behaviors and functions, respectively. From the previous example, ‘drink,’ ‘tilt,’ and ‘contact’ are verbs to describe the required behaviors, and a formal required behavior ‘FB01’ can be expressed with a subject ‘person’, a verb ‘drink’, and an object ‘water.’ Attributes also needed to specify a formal required behavior. For instance, the ‘FB01’ formal required behavior has an attribute ‘water flow-rate’ whose value is ‘more than 50 cc/sec.’

DesignedBehavior is behaviors of an artifact or feature, determined by product designers in response to formal required behaviors. It identifies some entities participating in

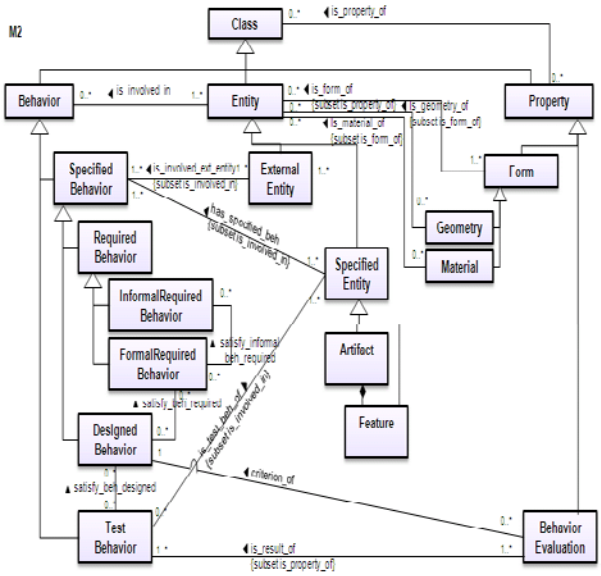


Fig. 4. Entity, Behavior, and Property classes and relationships.

it as specified entities (artifact and feature). The subject of its description should be an artifact or feature, while the subject of required behavior's description is an external entity. For example, 'Water-bottle provides water to mouth' is a designed behavior for the above required behavior example. A designed behavior can have specific attributes and their values that satisfy the attributes and values defined at corresponding required behaviors. For example, the 'water flow-rate' of the above designed behavior can have a value 'more than 60 cc/sec.'

Designed behaviors are defined to satisfy required behaviors. The relationship between **RequiredBehavior** and **DesignedBehavior**, 'satisfy_beh_required' relationship in Figure 4, means a specialization relationship. So, a designed behavior in the M1 layer can be described as a sub-class of required behaviors. In addition, artifacts involved in a designed behavior in the M1 layer should be a sub-class of another artifact that is involved in the required behaviors, or they can be a same class.

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 describing a test behavior, but a test behavior cannot exist without a related specified entity. **TestBehavior** is the observed behavior of the artifact or feature. It can be observed through test methods such as running mathematical models or simulation models, or testing physical prototypes or manufactured items. The test methods should be defined as an attribute of test behaviors at the M1 layer.

A test behavior of an artifact or feature should be compared to a designed behavior. It can satisfy the designed behavior if its observed behaviors are qualified for the designed behavior. Otherwise, it cannot satisfy the designed behavior. The evaluation results are recorded with **Behavior Evaluation** class, which is a sub-class of **Property**. Conformance relationship between the M1 and M0 layer is used to evaluate test behaviors, which will be explained in Section 5.

C. Property class

Property has two sub-classes; **Form** and **BehaviorEvaluation**. Forms are properties of **Entity**, and behavior evaluations are properties of **TestBehavior**. Only those two sub-classes are defined in this paper, but the class hierarchy of **Property** can be expanded with more sub-classes in future works if product development domain requires more properties.

Form is properties of entities that explain geometry and material aspects of the entities. It has two sub-classes: **Geometry** and **Material**. **Geometry** is defined to describe 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 layer. The relationship means a specialization relationship at the M1 layer. For example, a 'Cylinder' geometry at M1 can be used to describe many artifact designs such as a cup, pipe, or drum artifact at M1. All instances of those artifacts at M0 are members of the 'Cylinder' geometry. Therefore, the cup, pipe, and drum artifacts at M1 are sub-classes of the 'Cylinder' geometry. The specialization relationship is derived from the 'is_form_of' relationships between the artifacts and geometry classes at M1. If an engineer defines that 'Cylinder' geometry at M1 has the 'is_form_of' relationship with a 'Cup' artifact at M1, the semantic of the relationship should be defined as a specialization relationship [4].

BehaviorEvaluation is a sub-class of **Property** and is used to record results of behavior tests. It has an attribute 'result' at the M1 layer to specify the test results such as 'satisfactory' or 'failure.'

V. Developing Product models and Instances using SPML

SPML is an extension of OWL, which provides mechanisms for engineers to specify their models in this language. Using SPML at the layer M2 will allow engineers to write their product models at the M1 layer in this language (i.e., interaction between M2 and M1), which provides product-specific semantics developed in SPML (i.e., adding semantics to the product models). Once engineers defined product models using SPML, 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).

A. Developing product models (M2-M1 interactions)

Engineers and inference systems perform the interaction between the M2 and M1 layers. A SPML editor interface needs to be developed for engineers to describe their product-specific ontological models (M1) in SPML (M2). Axioms of SPML provide OWL syntax for SPML, and inference systems can assist engineers by telling what classes and relationships are needed to be defined. Inference systems can also check syntactic consistency of engineers'

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

Fig. 5. An example of interactions between the M2 and M1 layers.

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

Fig. 6. A behavior description example at the M1 layer.

```
spml:DesignedBehavior: Rotating_a_fan
  EquivalentTo:
      (spml:M0_has_attribute only (torque or rpm))  and
      (spml:M0_has_attribute exactly 2)  and
      (spml:M0_has_attribute some ( torque  and
            (spml:M0_has_value some int[>50, <100]))) and
      (spml:M0_has_attribute some ( rpm  and
            (spml:M0_has_value some int[>3000, <5000])))
      and
      (spml:has_sub_behavior only
                  (Receive_electricity or Spin_axis)) and
      (spml:has_sub_behavior some Receive_electricity)  and
      (spml:has_sub_behavior some Spin_axis)
```

Fig. 7. Axioms example at the M1 layer.

descriptions at M1 based on the SPML syntax. For example, let us assume that there are SPML axioms like Figure 5-(a). The axioms are represented in the Manchester OWL syntax [17]. They specify every artifact at M1 must have at least one relationship 'has_form' with a form. If an engineer defines a 'Motor' artifact class like Figure 5-(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 place underlined in Figure 5-(c) is where engineers need to fill the missing information.

B. Adding semantics to product models at M1 layer

If a product model description at M1 satisfies the axioms of SPML, axioms for the product model description at M1 can be added. Axioms at the M1 layer 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 Figure 6. 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 like Figure 7. The contents in Figure 6 and 7 are now hard coded for explanation, but a SPML editor user interface and inference system are in development process.

Inference systems require axiom generation rules in order to add axioms to the behavior descriptions. Axiom generation rules can be expressed or implemented in various ways. Lee [18] and Liang [19] showed implementation of rules using eXtensible Stylesheet Language Transformations (XSLT) [20] and Java [21], 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 SPML at the M2 layer. Then, axioms at the M1 layer 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 have to be generated manually by engineers in order 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. Since entities are described with their behaviors, axioms of entities can be defined with behaviors and reuse the axioms of the behaviors by referring 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 the behavior, and the definition of the behaviors will be included in the definition of the motor.

Axioms for SPML relationships also need to be generated and added to the product model descriptions. Some of SPML relationships are interpreted as specialization relationship such as 'satisfy' relationships among behaviors and 'is_form_of' relationships between entity and form.

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.

An 'is_form_of' relationship between an entity and form class is also interpreted as a specialization relationship, and the entity class becomes a sub-class of the form class. For example, if a 'Motor_Cylinder' class is a form of a 'Motor' artifact, a specialization relationship between two classes is added, and the 'Motor' class becomes a sub-class of the 'Motor_Cylinder' class.

Engineers can also define specialization relationships between behaviors, entities, or forms if they wish to represent the same product at different levels of detail [4]. Then, a specific 'Motor_A' in a detail design can be a specialization of a 'Motor' concept in a conceptual or preliminary design because all information about physical motors conforming to the former 'Motor_A' also conforms to the latter 'Motor'. In

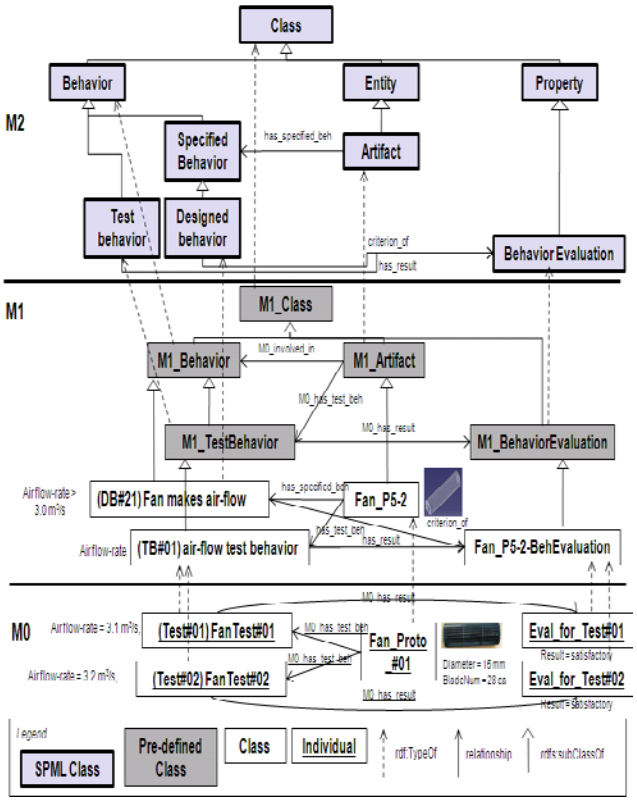


Fig. 8. Predefined classes at M1 for behavior evaluation of M0 instances

addition, the specialization relationship can save engineers the effort of defining duplicate axioms at the M1 layer. Since specialization implies axiom inheritance, engineers can use specialization relations in order 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 information about physical motors conforming to 'Motor_A' also conforms to the '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 layer 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.

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 layers is necessary to build and trace the relationships between product models and physical items. While the M1 layer represents different views of a product-specific model, the M0 layer represents occurrence or measured information about physical realizations (items) of the model. A physical item at M0 can have 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. Behavior occurrence 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.

The interaction between the M1 and M0 layers is implemented as a conformance relationship. Conformance 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 layer. In addition, if the conformance relationship is manually established between a class at the M1 layer and an instance at the M0 layer, the information of the instance should satisfy the definition of the class. For example, let us assume that there is a 'Motor_A' artifact class specialized from a 'MotorCylinder' geometry class whose attribute and value are 'diameter' and '20 ± 0.1 mm,' respectively. Every instance of the 'Motor_A' class must have an attribute 'diameter', and its value must be between 19.9 and 20.1 mm. These conformances can be checked by ontological inference engines.

A conformance relationship can be also established between a behavior class and behavior occurrence. A behavior occurrence at the M0 layer 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.

SPML has some pre-defines classes at M1 in order to classify M0 instances. **M1_Artifact**, **M1_Behavior**, **M1_TestedBehavior**, and **M1_BehaviorEvaluation** classes are pre-defined at the M1 layer for behavior conformance. Figure 8 shows the pre-defined classes and their relationships. Every specified behavior class at M1 should be defined as a specialization of **M1_Behavior**, and every test behavior class at M1 should be defined as a specialization of **M1_TestBehavior**. These specializations allow inference engines to check conformance relationships between a behavior occurrence at M0 and a designed behavior class at M1. For instance, behavior occurrences (Test#01 and Test#02) in Figure 8 have attributes and their values that satisfy a definition of a designed behavior (DB#21). Therefore, the occurrences can be inferred as instances of the designed behavior, and their behavior evaluations have a result attribute whose value is 'satisfactory.'

While the attribute values of a designed behavior at M1 are given by engineers, the attribute values of a test behavior at M1 cannot be specified until engineers get enough tested behavior occurrence information at M0. Therefore, the test behavior (TB#01) in Figure 8 has an attribute 'Air_flow_rate' without its value. However, if an engineer declares that the test behavior has enough occurrences at M0, the attribute may have a value like≥'3.1 m^3/s' which can embrace same

attribute's values of behavior occurrences at M0.

VI. Conclusion

The In the paper, we extended OWL to SPML to include specific language constructs to define and model product design and manufacturing. The constructs like 'Behavior,' 'Artifact,' etc. are defined in SPML (M2 layer), so that the semantics of these constructs can be exploited by designers and engineers. This will allow designer and engineers to check whether particular entities are behavior, artifact, or other classes in SPML. In OAM, 'Assembly' is a class and not defined in M2 layer. As a next step, the notion of 'Assembly' will be defined in SPML (M2 layer). As mentioned earlier, this will be a powerful feature to check whether an entity is an assembly or not. It is also to be noted that the axioms that define 'Behavior,' 'ExternalEntity,' 'Artifact' can be extended for specific needs, thus allowing extension to SPML.

A SPML editor interface is in development process. The interface will allow engineers to describe their product models semantically, which can be easily understood not only by engineers but also by computers. In addition, the interface will be connected to Computer-Aided Design (CAD) systems as a future work for semantically annotating the existing CAD models. SPML can be a powerful mechanism for annotating current CAD models using SPML constructs, so that the CAD models can be semantically enriched not only within geometric aspects but also beyond geometric aspects such as requirement, function, behavior, and sustainability aspect. The proposed framework needs additional work and implementation in order to be usable as a product modeling system. In terms of information modeling scope, SPML currently focuses on product structure and behavior evaluation in product lifecycle. Additional information modeling issues need to be addressed such as version control, product configuration, assembly relationship, tolerance, and sustainability. In addition, interfaces for integration with applications such as design knowledge bases and CAD systems need to be implemented.

Disclaimer

No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied. Certain commercial equipments, software, instruments, or materials are identified in this report to facilitate better understanding. Such identification does not imply recommendations 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.

Acknowledgment

The authors would like to acknowledge the comments and suggestions of Joshua Lubell, NIST staff, in improving the report 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, USA, NIST interagency/internal report (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," Computer-Aided Design, vol. 33, pp. 545-599, 2001.

[3] T. Gruber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," International Journal Human-Computer Studies, vol. 43, pp. 907-928, 1995.

[4] C. Bock, X. F. Zha, H. W. Suh, and J. H. Lee, "Ontological Product Modeling for Collaborative Design," NIST, Gaithersburg, MD, USA, NISTIR 7643, 2009.

[5] S. J. Fenves, S. Foufou, C. Bock, R. Sudarsan, N. Bouillon, and R. D. Sriram, "CPM 2: A revised core product model for representing design information," NIST, Gaithersburg, MD, USA, NISTIR 7185, 2005.

[6] Srinivasan, V., "An integration framework for product lifecycle management," Computer-Aided Design, 2009, doi:10.1016/j.cad.2008.12.001.

[7] C. K. Edmond, X. Chan, and K. M. Yu, "A framework of ontology-enabled product knowledge management," International Journal of Product Development, vol. 4, pp. 241-254, 2007.

[8] J. Lee, H. Chae, C. H. Kim, and K. Kim, "Design of product ontology architecture for collaborative enterprises," Expert Systems with Applications, vol. 36, pp. 2300-2309, 2008.

[9] J. H. Lee and H. W. Suh, "Ontology-based multi-layered knowledge framework for product lifecycle management," Concurrent Engineering Researches and Applications, vol. 16, pp. 301-311, 2009.

[10] D. Leal, "ISO 15926 Life Cycle Data for Process Plant: an Overview," Oil and Gas Science and Technology, vol. 60, pp. 629-638, 2005.

[11] D. Yang, M. Dong, and R. Miao, "Development of a product configuration system with an ontology-based approach," Computer-Aided Design, vol. 40, pp. 863-878, 2008.

[12] S. J. Fenves, "A core product model for representing design information," NIST, Gaithersburg, MD, USA, NISTIR 6736, 2002.

[13] S. Rachuri, Y. Han, S. Foufou, S. Feng, U. Roy, Fujun W., R. D. Sriram, and K. Lyons, "A Model for Capturing Product Assembly Information," Journal of Computing and Information Science in Engineering, vol. 6, pp. 11-21, 2006.

[14] A. Weissman, S.K. Gupta, X. Fiorentini, S. Rachuri, and R.D. Sriram, "Formal representation of product design specifications for validating product designs," NIST, Gaithersburg, MD, USA, NISTIR 7626, 2009.

[15] 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.

[16] Y. Kitamura, "A functional concept ontology and its application to automatic identification of functional structures," Advanced Engineering Informatics, vol. 16, pp. 145-163, 2002.

[17] M. Horridge, N. Drummond, J. Goodwin, A. rector, R. Stevens, and H. H. Wang, "The Manchester OWL Syntax," in Proc. of the OWL experiences and Directions Workshop (OWLED'06), 2006.

[18] J. H. Lee and H. W. Suh, "OWL-based Product Ontology (POWL) Architecture and Representation for Sharing Product Knowledge on a Web," in Proceedings of the ASME 2007 International Design Engineering Technical Conferences Computers and Information in Engineering Conference (IDETC/CIE) Las Vegas, NV, USA, 2007.

[19] V.C. Liang, C. Bock, and X.F. Zha, "An Ontological Modeling Platform," NIST, Gaithersburg, MD, USA, NISTIR 7509, 2008.

[20] J. Clark (1999) XSL Transformations (XSLT) Version 1.0 - W3C Recommendation 16 November 1999. [Online]. Available: http://www.w3.org/TR/xslt

[21] J. Gosling, B. Joy, G. Steele, and G. Bracha, The Java language specification, third edition: Addison-Wesley, 2005.