# A CORE PRODUCT MODEL FOR PLM WITH AN ILLUSTRATIVE XML IMPLEMENTATION

Sebti Foufou[1], Steven J. Fenves, Conrad Bock, Sudarsan Rachuri and Ram D. Sriram

National Institute of Standards and Technology
Mail Stop 8263, 100 Bureau Drive, Gaithersburg, MD 20899, U.S.A.
{sfoufou, sfenves, conrad.bock, sudarsan, sriram}@cme.nist.gov

## Abstract

We propose a Core Product Model (CPM) for modeling product information suitable for supporting the information needs of Product Lifecycle Management (PLM) systems over the lifecycle of the product from the earliest ideation to manufacturing, operation and disposal. We make this claim because in CPM a product is modeled as a triad of its function (what it is intended to do), its form (what is its shape and material) and its behavior (how it implements its function). Early pre-design activities can thus deal with the function without having a geometric model of the form, and activities subsequent to design can extend the behavior by the evaluation of multiple causal models (e.g., what is the product's market share).

The components of the conceptual model of CPM are presented in terms of the product's function, form and behavior, as well as the decompositions, associations and various kinds of relationships among these concepts. The model is represented in UML so as to provide interoperability with other models. An implementation model in XML and an example are presented to illustrate the principal elements of the proposed CPM model.

**Keywords**: Product Lifecycle Management (PLM) systems, information models, product models, conceptual models, function, form, behavior, XML, XML Schema.

## 1. INTRODUCTION

Current computer-aided design and engineering (CAD/CAE) tools serve only a very narrow range in the product's lifecycle: from the stage where the specification for the product has been defined to the stage where the design for the product can be turned over to manufacturing. Product Development Management (PDM) systems intended to support these tools cover the same range, with possibly some assistance for requirements development, on one hand, and information transfer to manufacturing, on the other. Furthermore, the information being managed is typically opaque to the PDM systems as they tend to depend on the underlying CAD tool to manage the information about the product's form, which is largely geometric in contents.

[1] Corresponding author. Sebti Foufou is with LE2i Laboratory, University of Burgundy, B.P. 47870, Dijon France. He is currently guest researcher at the MSID Division, NIST, Gaithersburg, MD 20899. sfoufou@u-bourgogne.fr

Interest in industry is increasingly being focused on Product Lifecycle Management (PLM) systems, intended to support the full spectrum of product development activities from the initial ideation to eventual disposal. Scaling up PDM systems to support this full spectrum of product development activities will not work, for the reasons given above: they are only geared to support geometric information for a limited range of design processes and even for this limited information exchange they depend on subsidiary systems [1; 2].

The wide scale adoption of PLM requires a product information model that allows all information used or generated in all the various product development activities to be transmitted to other activities by way of direct electronic interchange [3]. Furthermore, product development across companies, and even within a single company, will almost invariably take place within heterogeneous software environments, requiring a conceptual product information model that can be implemented in a variety of environments and that allows seamless interoperation between environments and applications [4].

A number of recent studies have shown the role that XML can play in a semantics-based product data representation for data portability (transfer of information and knowledge) when used as the information transfer medium. Rezayat presents the bases of a knowledge based product development system and explains why XML must be the support on which the system relies [5]. Mervyn et al. present an approach for developing distributed product and process design applications, where compressed model information embedded in a product data XML schema is used to ensure data portability [6]. The Product Data Markup Language (PDML) is a set of XML vocabularies for deploying product data on the Internet [7].

## 2. OVERVIEW OF CPM2

The core product model (CPM) was initially conceived as a basis of future systems that respond to the demands of the next generation of computer-aided design systems and provide for improved interoperability among software tools in the future [8]. It subsequently became clear that the CPM has the functionality to provide support for the full range of PLM activities [9].

CPM is a generic, abstract model with generic semantics, with meaningful semantics about a particular domain to be embedded within an implementation model and the policy of use of that model.

The key concept that makes CPM a candidate for supporting the full range of PLM activities is that a product is described by a triad:

- *Function* models what the artifact is supposed to do; the term function is often used synonymously with the term *intended behavior*.
- *Form* models the proposed design solution for the design problem specified by the function; in CPM, the artifact's physical characteristics are modeled in terms of its geometry (the "traditional" domain of CAD models) and material properties.
- *Behavior* models how the artifact implements its function in terms of the engineering principles incorporated into a behavioral or causal model; application of the behavioral model to the artifact describes or simulates the artifact's *observed behavior* based on its form.

Figure 1 shows a UML diagram of the Core Product Model (CPM). The descriptions below are condensed from the report on the second version of CPM, called CPM2 [9]. There are four categories of classes in CPM: classes that provide supporting information for the objects (abstract classes): classes of physical or conceptual objects; classes that describe associations (relationships) among the objects; and classes that are commonly used by other classes.

In the rest of this paper, the following naming conventions are used: names of CPM classes are written in boldface and capitalized (e.g., **CoreProductModel**, **EntityAssociation**, **Artifact**). Names of attributes are in boldface (e.g., **`information`**) and lower case. Names of XML elements are also in boldface but in a different font and they start with lower case (e.g., **artifact**, **hasFeature)**. Literals (e.g., attribute values and content of XML elements) and names of instances are in italics (e.g., *cylindricalForm*).

There are five supporting classes for storing common information:

**CoreProductModel** represents the highest level of generalization. The common attributes **`type`**[2], **`name`** and **`information`** for all CPM classes are defined for this class

**CommonCoreObject** is the base class for all the object classes. **CommonCoreRelationship** and its specializations may be applied to instances of classes derived from this class.

**CommonCoreRelationship** is the base class from which all association classes are specialized.

**CoreEntity** is the base class from which the classes **Artifact** and **Feature** are specialized. **EntityAssociation** relationships may be applied to entities in this class.

**CoreProperty** is an abstract class from which the classes **Function**, **Flow**, **Form**, **Geometry** and **Material** are specialized. **Constraint** relationships may be applied to instances of this class.

The following constitute the object classes:

**Artifact** is the key object class in the model; it represents a distinct entity in a product, whether that entity is a component, part, subassembly or assembly. All the latter entities can be represented and interrelated through the **`subArtifacts/subArtifactOf`** containment hierarchy.

**Feature** is a portion of the artifact's form that has some specific function assigned to it. Thus, an artifact may have design features, analysis features, manufacturing features, etc. **Feature** has its own containment hierarchy, so that compound features can be created out

[2] The value of the **`type`** attribute corresponds to one of the terms within a distinct hierarchical taxonomy of terms associated with the class and provides the taxonomy for the given class.

of other features. A Feature has attributes of **Function** and **Form**, but does not have a separate **Behavior**.

**Port** is a specific kind of feature, sometimes referred to as an interface feature.

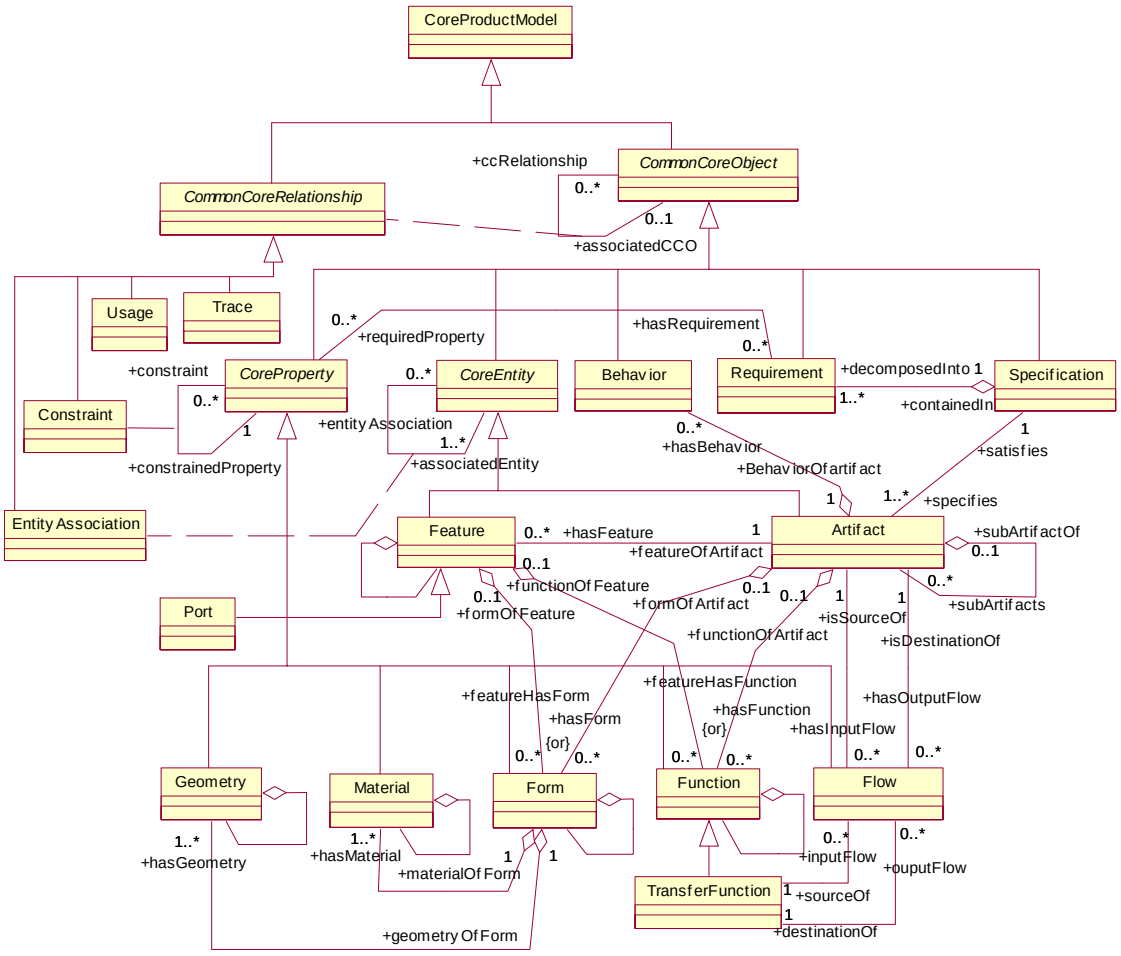


**Figure 1 : Class Diagram of the Core Product Model**

**Specification** is the collection of information relevant to an **Artifact** deriving from customer needs and/or engineering requirements; it is a container for the specific **Requirement**s that the artifact must satisfy.

**Requirement** is a specific element of the **Specification** of an artifact that governs some aspect of its function, form, geometry or material. Requirements cannot apply to behavior, which is strictly determined by a behavioral model.

**Function** represents what the artifact or feature is supposed to do. The artifact satisfies customer needs and/or engineering requirements largely through its function.

**TransferFunction** is a specialized form of **Function** involving the transfer of an input flow into an output flow.

**Flow** is the medium that serves as the output of one or more transfer function(s) and the input of one or more other transfer function(s).

**Behavior** describes how the artifact implements its function; it is governed by physical, chemical or other engineering principles that are incorporated into a behavioral or causal model.

**Form** of the artifact or feature is the design solution for the problem specified by the function. In the CPM, the artifact's or feature's physical characteristics are represented by two distinct classes:

**Geometry** is the spatial description of an artifact or feature.

**Material** is the material composition of an artifact or feature.

The following constitutes the association (relationship) classes derived from the **CommonCoreRelationship** class:

**Constraint** is a specific shared property of a set of entities that must hold in all cases. At the level of the CPM, only the entity instances that constitute the constrained set are identified.

**EntityAssociation** is a simple set membership relationship among artifacts and features.

**Usage** is a mapping from **CommonCoreObject** to **CommonCoreObject,** useful when constraints apply to multiple "target" entities but not to the generic "source" entity.

**Trace** is structurally identical to **Usage**, useful when the "target" entity in the current product description depends on a "source" entity in another product description.

The following three utility classes, not shown on Figure 1, provide additional detail:

**Information,** an attribute of **CoreProductModel** and all its specializations, is a container consisting of three attributes: (i) a textual `description`; (ii) a textual `documentation`; and (iii) `properties`, a set of attribute-value pairs representing all domain- or object-specific attributes.

**ProcessInformation,** an attribute of **Artifact,** contains product development process parameters that may be used in a Product Lifecycle Management (PLM) environment.

**Rationale,** an attribute of **CoreProperty,** documents decision in the product development process.

The classes described are linked by three kinds of associations. First, all object classes have their own separate, independent decomposition hierarchies[3] by attributes such as `subArtifacts/subArtifactOf` for the **Artifact** class.

Second, there are associations between:

- a **Specification** and the **Artifact** that results from it
- a **Flow** and its source and destination **Artifacts** and its input and output **Function**s
- an **Artifact** and its **Feature**s.

Third, and most importantly, four aggregations are fundamental to the CPM:

- **Function**, **Form** and **Behavior** aggregate into **Artifact**
- **Function** and **Form** aggregate into **Feature**
- **Geometry** and **Material** aggregate into **Form**
- **Requirement** aggregates into **Specification**.

The conceptual model of CPM may be used in actual applications, albeit at a high overhead cost: specific instances of entities must be located by means of their `type` and their attributes stored in and retrieved from the `properties` slot of the associated **Information** instance. These same two constructs, type and properties, may be used by a model compiler to create subclasses of **Artifact** from the specifications in the `type` slot, and define attributes on the subclasses from the `properies` list [8].

## 3. AN XML IMPLEMENTATION OF THE CPM

In order to illustrate the implementation and use of the CPM, we have generated the Core Product XML Schema (CPXS) equivalent to the CPM and a set of Java classes. We have also developed a Java graphical user interface to input product data and generate valid XML documents according to the CPXS schema.

The full XML implementation of CPXS is presented in [8]. Here we present only one illustrative example of the consequences of the fact that the XML schema language is not an object-oriented language. The XML schema resulting from the conversion of an UML class diagram needs to be constrained to ensure the consistency of the XML instance document.

### 3.1. The XML Artifact complexType

Figure 2 shows the XML complexType representing an artifact. Inherited attributes are not shown in this figure; the **listOfxxx** type represents a set of strings referring to the **name** subelement of elements of type **xxx** (e.g., within an **artifact**, the **hasFeature** element contains a set of subelements, each of which refers to the **name** of a feature of that artifact).

As an example of the added constraints needed for XML document consistency, consider the element **satisfies** within the **artifact** element. This tag references the name or code of

[3] For clarity, only the `subArtifacts/subArtifactOf` containment hierarchy of **Artifact** is labeled in Figure 1.

the **specification** element that the artifact satisfies; the referenced **specification** element must be an element of the XML document; otherwise the document is not consistent.

```
<xsd:complexType name="Artifact">
<xsd:complexContent>
 <xsd:extension base="CoreEntity">
 <xsd:sequence>
 <xsd:element name="hasBehavior" type="listOfBehaviors" minOccurs="0"/>
 <xsd:element name="hasFunction" type="listOfFunctions" minOccurs="0"/>
 <xsd:element name="hasForm" type="listOfForms" minOccurs="0"/>
 <xsd:element name="satisfies" type="xsd:string" minOccurs="1"/>
 <xsd:element name="hasFeature" type="listOfFeatures" minOccurs="0"/>
 <xsd:element name="subArtifacts" type="listOfArtifacts" minOccurs="0"/>
 <xsd:element name="subArtifactOf" type="xsd:string" minOccurs="0"/>
 <xsd:element name="hasInputFlow" type="listOfFlows" minOccurs="0"/>
 <xsd:element name="hasOutputFlow" type="listOfFlows" minOccurs="0"/>
 </xsd:sequence>
 </xsd:extension>
</xsd:complexContent>
</xsd:complexType>
```

**Figure 2: The XML Artifact complexType**

To ensure this consistency, we define the **name** element to be the key of the **specification** element (Figure 3) and indicate that the element **satisfies** within the **artifact** element is a reference to this key. The figure also shows that the element **containedIn** in the **requirement** element must reference a valid and unique specification **name** (i.e., a key).

```
<xsd:key name="pKSpec">
      <xsd:selector xpath="cpm:specification"/>
      <xsd:field xpath="cpm:name"/>
</xsd:key>

<!-- Elements that shall reference a specification name -->
<xsd:keyref name="specRef" refer="pKSpec">
      <xsd:selector xpath="cpm:artifact"/>
      <xsd:field xpath="cpm:satisfies"/>
</xsd:keyref>
<xsd:keyref name="spec1fRef" refer="pKSpec">
      <xsd:selector xpath="cpm:requirement"/>
      <xsd:field xpath="cpm:containedIn"/>
</xsd:keyref>
```

**Figure 3: Example of a Consistency Constraint in the XML Artifact Schema**

### 3.2. Example: XML representation of a planetary gear

The planetary gear system (PGS) example considered in this section was presented in detail in [10], where it was used to illustrate the representation of both the **Artifact** containment hierarchy and the assembly associations comprising the Open Assembly Model, an extension of CPM. Our interest in the example here is to show how CPM captures design

information about a product; thus, only data important from the design point of view are modeled.

Figure 4 shows the components of the PGS: the main artifact is the planetary gear; it is composed of 13 subartifacts: 8 screws, the output housing, the input housing, the ring gear, the sun gear, and the planet gear carrier. Information pertaining to the function, form, behavior and specification related to these subartifacts is not presented here.

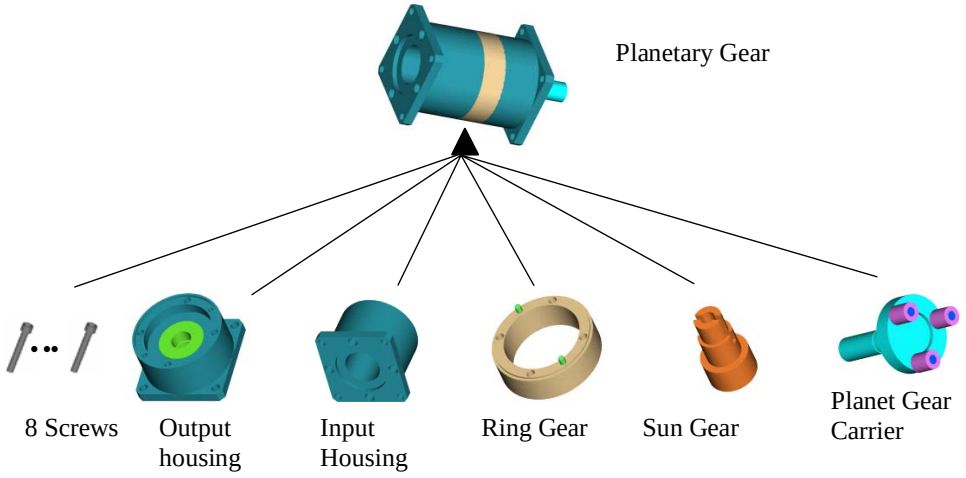


**Figure 4: The Planetary Gear System**

Figure 5 shows an XML **artifact** element describing the PGS. The figure shows that this element includes the following set of subelements:

- **information**: a description and brief documentation of the PGS artifact[4]
- **behaviors**: a list of names of elements that describe the behavior of the PGS artifact.
- **functions**, **forms**, **features**: three lists, the elements of which give the names of the function *changeSpeedOfRotation*, the form *cylindricalForm*, and the features *fasteningHoles* and *outputShaftHole* of the PGS, respectively.
- **satisfies**: name of the XML element that gives the specification that the PGS shall satisfy. In the full example presented in [8], this specification is decomposed into a set of requirements that include form, input speed, output speed, input torque and output torque requirements.
- **subArtifacts**: a list of the names of the subartifacts of the PGS artifact: *planetGearCarrier*, *sunGear*, *ringGear*, *inputHousing*, *outputHousing* and *eight screws*.

[4] Since XML is not an object-oriented language, the **Information** entity and its attributes `description`, `documentation` and `properties` have been inserted in the parent entity.

```
<artifact>
    <name>PlanetaryGearSystem</name>
    <information>
        <description>
             The PlanetaryGearSystem for changing speed rotation
        </description>
        <documentation>
             This is an assembly of 13 different
             subartifacts and subassemblies
        </documentation>
    </information>
    <behaviors> <theBehavior name="pgsBehavior"/> </behaviors>
    <functions> <theFunction name="changeSpeedOfRotation"/>
    </functions>
    <forms> <theForm name="cylindricalForm"/> </forms>
    <satisfies>pgsSpecification</satisfies>
    <features>
        <theFeature name="fasteningHoles"/>
        <theFeature name="outputShaftHole"/>
    </features>
    <subArtifacts>
        <theArtifact name = "planetGearCarrier"/>
        <theArtifact name = "sunGear"/>
        <theArtifact name = "ringGear"/>
        <theArtifact name = "inputHousing"/>
        <theArtifact name = "outputHousing"/>
        <theArtifact name = "screw1"/>
            …
        <theArtifact name = "screw8"/>
    </subArtifacts>
</artifact>
```

**Figure 5: Artifact Element of the Planetary Gear System**

Figure 6 and Figure 7 show the definitions of the behavior and the function of the above artifact, respectively, by means of the *pgsBehavior* and *pgsFunction* elements. One can see how the **properties** slot of the **information** element is used to capture important information such as speed ratio and output torque.

```
<behavior>
  <name>pgsBehavior</name>
  <information>
    <description>the behavior of the planetary gear
        system after assembly analysis and validation
    </description>
    <properties>
      <property name="speedRatio">3.0:1</property>
      <property name="torqueOut">6.78 N.m</property>
    </properties>
  </information>
  <artifact>PlanetaryGearSystem</artifact>
</behavior>
```

**Figure 6: Behavior Element of the Planetary Gear System**

The content of the **artifact** element, within the **behavior**, and the content of **functionOfArtifact** element, within the **function**, are the names of the artifact to which these two elements pertain. As the content is the same for both elements, we understand that they pertain to the same artifact, which is the main artifact of the planetary gear system.

```
<function>
   <name>changeSpeedOfRotation</name>
   <information>
      <description>the main function of thePlanetaryGearSystem. It
      provides adequate and variable speed for all possible
      operations
      </description>
      <properties>
         <property name="input">rotational energy</property>
         <property name="output">rotational energy</property>
         <property name="speedIn">1800rpm</property>
         <property name="speedOut">TBD</property>
         <property name="torqueIn">2.26 N.m</property>
         <property name="torqueOut">TBD</property>
      </properties>
   </information>
   <functionOfArtifact>PlanetaryGearSystem</functionOfArtifact>
</function>
```

**Figure 7: Function Element of the Planetary Gear System**

The XML standard has been used for data representation to overcome a number of interoperability problems in various domain of application (e.g. MathML, CML). By separating the content (data) from the presentation, XML documents are excellent supports to capture product data, for later diffusion (in various formats) and/or any further automatic treatments. The CPM implementation given in this section is far from being complete or final. It is provided to show how CPM and XML can be combined and used as a basis for future product data management systems.

## 4.CONCLUSION

The Core Product Model presented in this paper can be seen as a first step toward the definition of a product model suitable for supporting the information needs of Product Lifecycle Management (PLM) systems over the lifecycle of the product from the earliest ideation to manufacturing, operation and disposal. The CPM captures a wide range of common product information (specification, requirements, function, form, behavior, material, constraints, etc.). Domain-specific models have been defined as extensions of this conceptual model [10]. Implementation models can be generated using the contents of the conceptual model to semi-automatically create domain-specific subclasses with their attributes.

The XML implementation and the planetary gear system example show how the CMP can be used with semantic web standards technologies to model product data in an interoperable manner.

This work is part of the Interoperability Program in the Manufacturing Systems Integration Division of the Manufacturing Engineering Laboratory at NIST. As part of this program, future work will involve the collection and correlation of product information throughout the Virtual Manufacturing Enterprise (VME) under development, leading towards extensions of the CPM to serve as the basic organizing principle for the information in the VME.

## REFERENCES


1. Burkett, M., O'Marah, K., and Carrillo, L., "CAD Versus ERP Versus PDM: How Best To Anchor a PLM Strategy?," AMR Research, Sept. 2003.

2. Philpotts and M, "An introduction to the concepts, benefits and terminology of product data management," *Industrial Management & Data Systems*, Vol. 96, No. 4, 1996, pp. 11-17.

3. Sudarsan, R., Fenves, S., Sriram, R. D., and Wang, F., "A Product Information Modeling Framework for Product Lifecycle Management," *Computer-Aided Design*, 2005.

4. O'Marah, K., and Myer, B., "*The Product Lifecycle Management Applications Report, 2001-2006*," AMR Research, 2002.

5. Rezayat, M.. Knowledge-based product development using XML and KCs. Computer-Aided Design 32, 299-309. 2000.

6. Mervyn, F., Senthil Kumar, A., Bok, S. H., and Nee, A. Y. C., "Developing distributed applications for integrated product and process design," *Computer-Aided Design*, Vol. 36, No. 8, 2004, pp. 679-689.

7. Burkett, W. C., "Product data markup language: a new paradigm for product data exchange and integration," *Computer-Aided Design*, Vol. 33, No. 7, 2001, pp. 489-500.

8. Fenves, S. J., "*A Core Product Model For Representing Design Information*," National Institute of Standards and Technology, NISTIR6736 , Gaithersburg, MD 20899, USA, 2001.

9. Fenves, S., Foufou, S., Bock, C., Bouillon, N., and Sriram, R. D., "*CPM2: A Revised Core Product Model for Representing Design Information* ," National Institute of Standards and Technology, NISTIR 7185, Gaithersburg, MD 20899, USA, 2004.

10. Sudarsan, R., Young-Hyun H, Feng, S. C., Roy, U., Fujun W., Sriram, R. D., and Lyons, K. W., "*Object-oriented Representation of Electro-Mechanical Assemblies Using UML*," National Institute of Standards and Technology, NISTIR 7057, Gaithersburg, MD 20899, USA, 2003.