NISTIR 7185

CPM 2: A REVISED CORE PRODUCT MODEL FOR REPRESENTING DESIGN INFORMATION

Steven J. Fenves
Sebti Foufou
Conrad Bock
Rachuri Sudarsan
Nicolas Bouillon
Ram D. Sriram

NIST
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

NISTIR 7185

CPM 2: A REVISED CORE PRODUCT MODEL FOR REPRESENTING DESIGN INFORMATION

Steven J. Fenves
Sebti Foufou
Conrad Bock
Rachuri Sudarsan
Nicolas Bouillon
Ram D. Sriram

October, 2004

U.S. DEPARTMENT OF COMMERCE
Donald L. Evans, Secretary
TECHNOLOGY ADMINISTRATION
Phillip J. Bond, Under Secretary of Commerce for Technology
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
Arden L. Bement, Jr., Director

ABSTRACT

This report presents a revised version of the Core Product Model (CPM) initially reported in [1]. The initial CPM was intended to provide a base-level product model that is: not tied to any specific application or software; open; non-proprietary; simple; generic; expandable; independent of any product development process; and capable of capturing all product information shared throughout the product's lifecycle. The revisions presented continue to support these intentions.

The objectives of the report are: (1) to document the changes in the CPM relative to the initial version; (2) to describe in detail the revised CPM, represented as a UML class diagram; (3) to show, through Java and XML implementations, how the CPM can be used as the basis, or organizing principle, of a product information-modeling framework that can support the full range of product design information; and (4) to present a rational, model-based process for converting a CPM supporting the early conceptual phases of design into an implementation-level operational database support system.

UML, XML and Java representations of the model are presented so as to provide interoperability with other models. A case study example is discussed and its XML representation is presented and analyzed to illustrate the principal elements of the revised CPM.

Keywords
Product modeling, information modeling, data modeling, artifact, form, function, behavior, entity-relationship data model, core product model

TABLE OF CONTENTS

1 Introduction 6
1.1 Objectives 6
1.2 Historic background 6
2 The revised Core Product Model 7
2.1 Representation of attributes and class types 8
2.2 The CPM classes 9
2.2.1 Abstract classes 9
2.2.2 Object classes 10
2.2.3 Relationship classes 12
2.2.4 Utility classes 14
2.2.5 Class hierarchies 14
2.2.6 Associations and aggregations 15
3 Java and XML Implementations of CPM 15
3.1.1 The Java Artifact class 15
3.1.2 The XML Artifact complexType 16
3.1.3 Example: XML representation of a planetary gear 17
4 Model compilation for the CPM 20
5 Future work 23
6 Summary and conclusions 24
References 24
Appendix A: Java implementation of CPM 27
Appendix B: CPXS of the Core Product XML Schema 29
Appendix C: XML representation of a planetary gear 43

LIST OF FIGURES

Figure 1: Class diagram of the Core Product Model ____ 8
Figure 2: CPM abstract classes ____ 10
Figure 3: CPM object classes ____ 11
Figure 4: CPM relationship classes ____ 13
Figure 5: Relationships between object classes ____ 13
Figure 6: The Java Artifact class ____ 16
Figure 7: The XML Artifact complexType ____ 17
Figure 8: Example of a consistency constraint in the XML artifact schema ____ 17
Figure 9: The planetary gear system ____ 18
Figure 10: An XML element representing the planetary gear system ____ 19
Figure 11: Behavior element of the planetary gear system ____ 20
Figure 12: CPM instance of attributes ____ 20
Figure 13: CPM class diagram of attributes ____ 21
Figure 14: Java code generated from an attribute instance ____ 21
Figure 15: Derivation path of attributes ____ 22

1 Introduction

1.1 Objectives

This report presents a revised version of the Core Product Model (CPM) initially reported in [1]. The objectives of the report are: (1) to document the changes in the CPM relative to the first version; (2) to describe in detail the revised CPM, represented as a Universal Modeling Language (UML) class diagram; (3) to show, throughout Java and eXtensible Markup Language (XML) implementations, how the CPM can be used as the basis, or organizing principle, of a product information-modeling framework that can support the full range of product design information; and (4) to present a rational, model-based process for converting a CPM supporting early conceptual phases of design into an implementation-level operational database support system.

1.2 Historic background

As discussed in detail in [1], the initial direction of the work presented was an attempt to provide a common basis among four in-house research and development projects at NIST:

- the NIST Design Repository project
- the Design-Process Planning Integration project
- the Design for Tolerancing of Electro-Mechanical Assemblies project
- the Object-Oriented Distributed Design Environment project.

As the projects progressed the commonality of concepts became apparent and a more general direction was sought. This direction was provided by the perception of the need for new engineering design and analysis tools. Product development is increasingly performed by geographically and temporally distributed teams with a high level of outsourcing of many phases of the product development process. New tools will be needed to address the full spectrum of product development activities encompassed by Product Lifecycle Management (PLM) systems, rather than just the narrow range covered by traditional Computer Aided Design and Engineering (CAD/CAE) systems. Next-generation tools will require representations that allow all information used or generated in the various product development activities to be transmitted to other activities by way of direct electronic interchange. Furthermore, product development across companies, and even within a single company, will almost invariably take place within a heterogeneous software environment.

The Core Product Model was conceived as a representation for product development information which can form the basis of future systems that respond to the demands sketched above and provide for improved interoperability among software tools in the future [2]. The model focuses on an artifact representation 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, mappings between function and form, and various kinds of relationships among these concepts.

CPM follows the tradition of work in the area of artifact representation. The division of artifact information into the categories of form, function, and behavior has its roots in earlier work in intelligent design systems. The model is most directly descended from the representation developed as part of the NIST Design Repository project [3], [4]. The model presented here shares both conceptual and representational aspects with that developed by the MOKA (Methodology and tools Oriented to Knowledge based engineering Applications) Consortium, an ESPRIT-funded collaborative project of the European Union [5]. The initial Core Product Model was completed in the Fall of 2000 and is documented in [1].

2 The revised Core Product Model

The Core Product Model is heavily influenced by the entity-relationship data model [6]. Accordingly, the CPM consists of two sets of classes, called *object* and *relationship*. The two sets of classes are equivalent to the Unified Modeling Language (UML) terms of *class* and *association class*, respectively [7].

In the text that follows, names of classes have initial capitalization (e. g., **Information**) and names of attributes do not (e. g., **information**).

A UML class diagram of the CPM is shown in Figure 1.

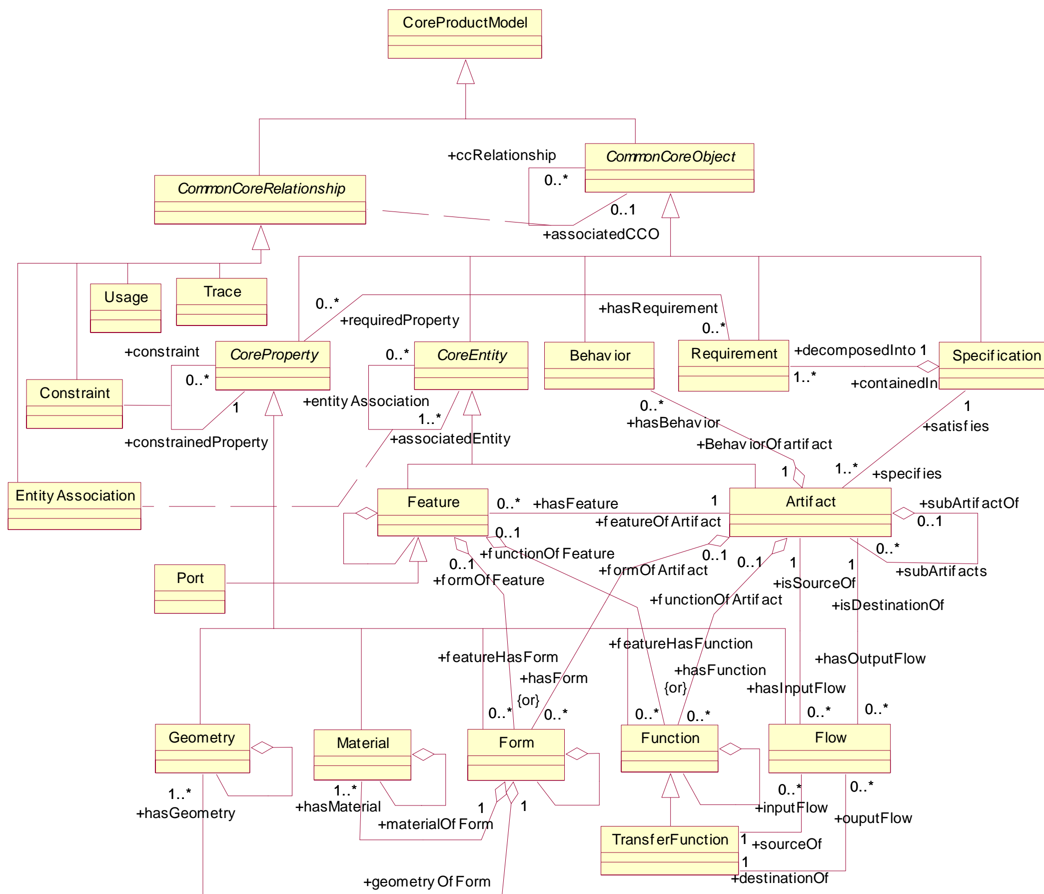

Figure 1: Class diagram of the Core Product Model

The general characteristics of the classes are discussed first. Then, the semantics of each class of objects and relationships is presented. Finally, the hierarchies and relationships among the classes are presented.

2.1 Representation of attributes and class types

In order to make the representation as robust as possible without having to predefine attributes that might be relevant only in a given domain, the CPM is limited to attributes required to capture generic product information and to create relationships among the classes. The representation intentionally excludes attributes that are domain-specific (e.g., attributes of mechanical or electronic devices) or object-specific (e.g., attributes specific to function, form or behavior). For the representation of this information, two generic information modeling concepts have been adopted.

First, each object and relationship has an **information** attribute. The class **Information** is a container consisting of:

- a textual **description** slot;

- a textual **documentation** string (e.g., a file path or URL referencing more substantial documentation); and
- a **properties** slot that contains a set of attribute-value pairs stored as strings representing all domain- or object-specific attributes.

This lack of specialization results in a small number of broadly applicable classes.

Second, all object and relationship classes, except for the abstract classes and the utility classes **Information**, **ProcessInformation** and **Rationale**, have an attribute called **type**, the value of which is a string that acts as a symbolic classifier[1]. Each object and relationship class may have a distinct hierarchical taxonomy of terms associated with that class. The value of the **type** attribute corresponds to one of the terms within the taxonomy for the given class. For example, "convert" is one of numerous types of transfer functions and the term can serve as the value of the **type** attribute of an instance of the class. Thus, all object and relationship classes in the representation may have their own individual generic engineering classification hierarchies that are independent of any other hierarchy (eventually, these taxonomies may be expanded into full ontologies of the terms and their semantic relationships). Implementations based on the CPM may use the **type** attributes, their underlying taxonomies and the attribute-value pairs stored in the entities' **Information** container to provide the means for model compilation of domain-specific specializations of the CPM classes, as discussed in Section 4.

Extensions and implementations of the CPM may explicitly assign attributes to specializations of the CPM objects and relationships. This has been done, for example, in the Open Assembly Model (OAM) discussed in a companion report [9], so as to provide interoperability with new systems, legacy data models such as STEP, or existing CAD programs.

2.2 The CPM classes

The classes comprising the CPM are grouped below into four categories: abstract classes, object classes, relationship classes and utility classes.

2.2.1 Abstract classes

In UML and in object-oriented programming, abstract classes are classes for which all instances are instances of a subclass. Abstract classes are used in the top of class hierarchies to store common methods or attributes. Figure 2 shows the four abstract classes in the CPM.

CoreProductModel

This class represents the highest level of generalization; all CPM classes are specialized from it according to the class hierarchy shown in Figure 2 and further discussed in Section 2.2.5. The common attributes **type**, **name** and **information** for all CPM classes are defined for this class.

[1] The semantics of the term **type** used in this report differs from that of the term "data type" commonly used in computer science data structure definitions. The use of the term **type** in this report is consistent with the definition used in the FRISCO report: "Type (Synonym: 'Category'): A type of things is a specific characterisation (e.g., a predicate) applying to all things of that type" [8]

CommonCoreObject
This is the base class for all the object classes. **CommonCoreRelationship** and its specializations, the **EntityAssociation**, **Constraint Usage** and **Trace** relationships, may be applied to instances of classes derived from this class.

CommonCoreRelationship
This is the base class from which all association classes are specialized according to the class hierarchy presented in Section 2.2.5. As stated above, the **CommonCoreRelationship** class serves as an association to the **CommonCoreObject** class.

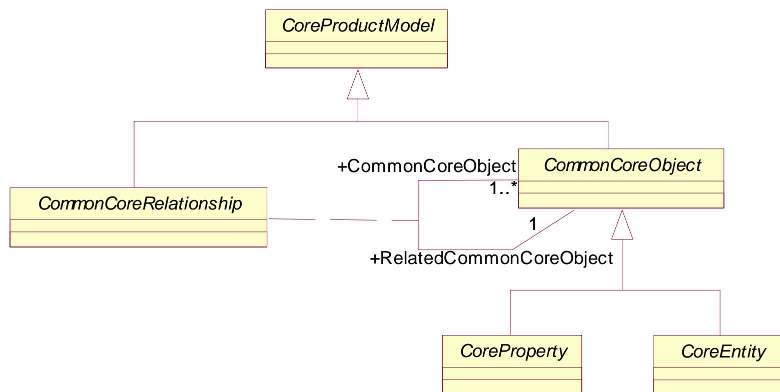

Figure 2: CPM abstract classes

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

CoreProperty
This 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.

2.2.2 Object classes

Figure 3 gives an abstract view of the CPM where only object classes are shown. The containment relationship **subArtifacts/subArtifactOf** is illustrated in the figure as an example.

Artifact
The key object class in the CPM is **Artifact**. **Artifact** 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 discussed in Section 2.2.5. The **Artifact's** attributes, other than the common ones described in Section 2.1, refer to the **Specification** that specifies the **Artifact**, the **Form**, **Function** and **Behavior** objects comprising the **Artifact**, i.e., in UML terminology, forming an aggregation with the **Artifact**, and the **Features** that may comprise the **Artifact.**

Feature
A **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., as determined by their respective functions. **Feature** has its own containment hierarchy, so that compound features can be created out of other features (but not artifacts).

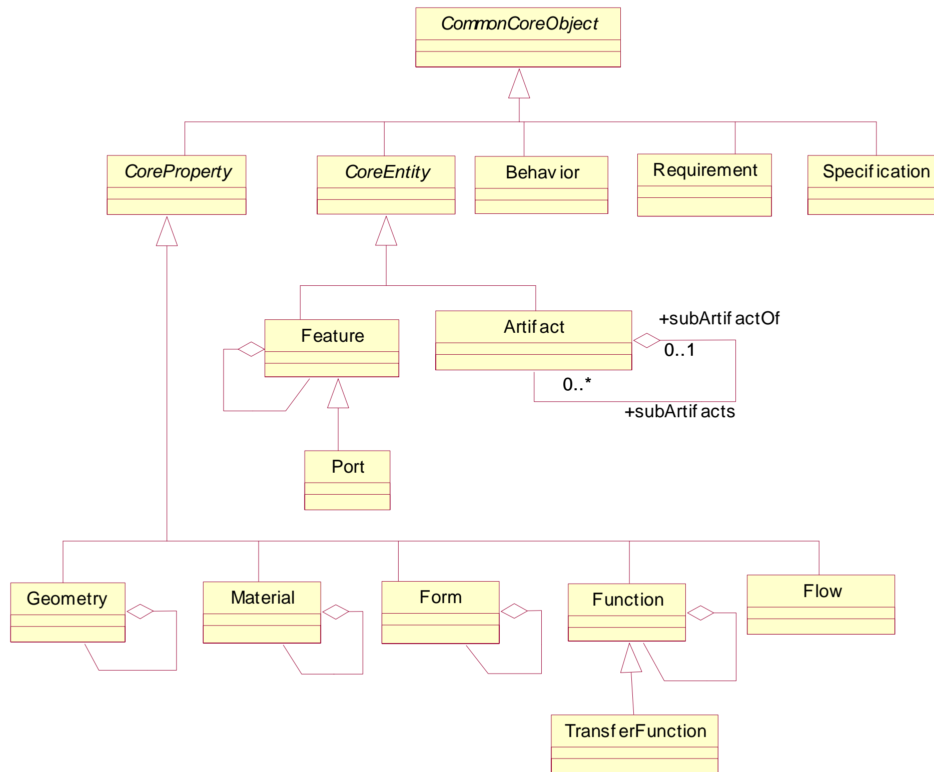

Figure 3: CPM object classes

Port
A **Port**, a specialization of **Feature**, is a specific kind of feature (sometimes referred to as an interface feature) through which the artifact is connected to (or interfaces with) other artifacts. The semantics of the term port are deliberately left vague: in some contexts ports only denote signal, control or display connection points, while in other contexts ports are equivalents of assembly features through which components mate.

Specification
A **Specification** represents the collection of information relevant to the design of an **Artifact** deriving from customer needs and/or engineering requirements. The **Specification** is a container for the specific requirements that the function, form, geometry and material of the artifact must satisfy.

Requirement
A **Requirement** is a specific element of the specification of an artifact that governs some aspect of its function, form, geometry or material. Conceptually, requirements should

only affect the function, i.e., the intended behavior, of the artifact; in practice, some requirements tend to affect the design solution directly, i.e., the form, geometry or material of the artifact. Requirements cannot apply to behavior, which is strictly determined by the behavioral model.

Function

A **Function** represents one aspect of what the artifact is supposed to do. The artifact satisfies customer needs and/or engineering requirements largely through its function. The term function is often used synonymously with the term *intended behavior*.

TransferFunction

A **TransferFunction** is a specialized form of **Function** involving the transfer of an input flow into an output flow. Examples of transfer functions are "transmit" a flow of fluid, current, or messages, etc., or "convert" from one energy flow to another or from a message to an action.

Flow

A **Flow** is the medium (fluid, energy, message stream, etc.) that serves as the output of one or more transfer function(s) and the input of one or more other transfer function(s). A flow is also identified by its source and destination artifacts.

Behavior

Behavior describes how the artifact implements its function. **Behavior** is governed by engineering principles which are 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. The observed behavior can then be examined with respect to the requirements to yield the *evaluated behavior*. In the evaluation process, unintended behaviors, i. e., that do not contribute to the intended function, can be identified and evaluated. **Behavior** has three specialized attributes or slots to hold the **behavioralModel**, **the observedBehavior** and the **evaluatedBehavior** (typically, URLs to the executable analysis program that embodies the behavioral model, the output of the behavioral model and the output of the external evaluation, respectively).

Form

The **Form** of the artifact can be viewed as the proposed design solution for the design problem specified by the function. In the CPM, the artifact's physical characteristics are represented in terms of its geometry and material properties. This subdivision was introduced because some of the intended applications tend to treat these two aspects quite differently (e. g., the product development process may have a separate task of material selection for a given function and geometry).

Geometry

Geometry is the spatial description of the artifact.

Material

Material is the description of the internal composition of the artifact.

2.2.3 Relationship classes

Relationship classes are derived from the **CommonCoreRelationship** class. They are shown in Figure 4**.** Relationships between object classes are shown in Figure 5.

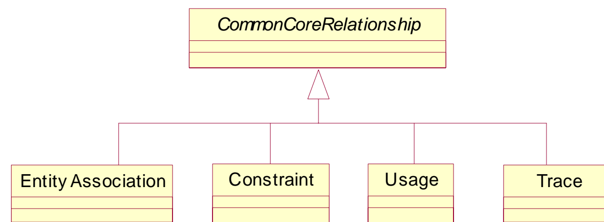

Figure 4: CPM relationship classes

Constraint

A **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. If it is intended to represent a mathematical equality or inequality constraint, the **properties** slot of the **Information** element associated with the constraint can hold the names of the attributes that enter in the constraint as well as the relational operator linking them.

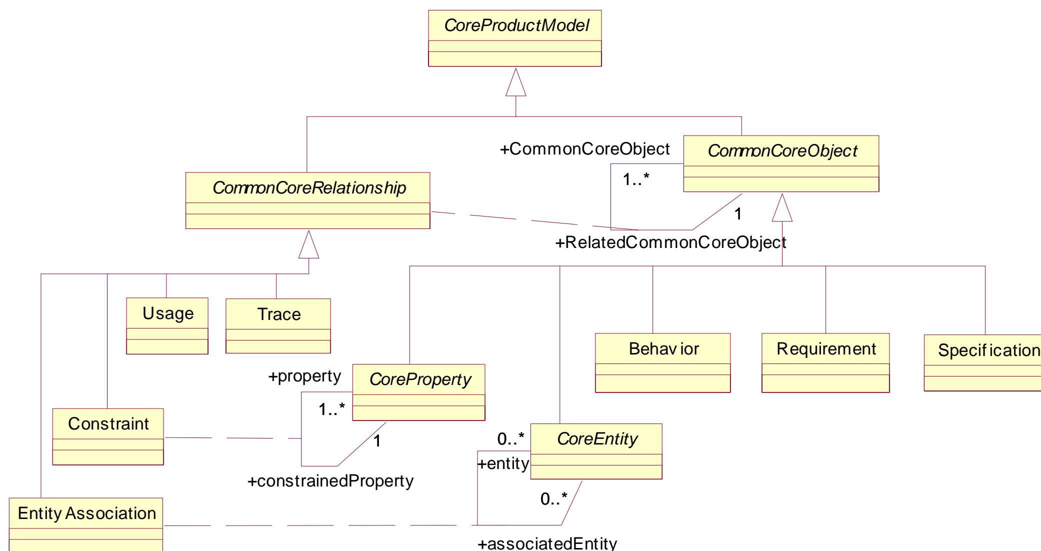

Figure 5: Relationships between object classes

EntityAssociation

EntityAssociation is a simple set membership relationship among artifacts, features and ports. In applications of the CPM this relationship can be specialized; for example, in the Open Assembly Model, **EntityAssociation** is specialized to **ArtifactAssociation** [9].

Usage

Usage is a mapping from **CommonCoreObject** to **CommonCoreObject**. The relationship is particularly useful when constraints apply to the specific "target" entity but not to the generic "source" entity, or when the source entity resides in an external catalog or design repository.

Trace

Trace is structurally identical to **Usage**. The relationship is particularly useful when the "target" entity in the current product description depends in some way on a "source" entity in another product description. The **type** attribute of **Trace** specifies the nature of the dependence, as follows:

(a) *Alternative_of*: this link points from one alternative to another at the highest level of the artifact decomposition hierarchy where the two alternatives differ (it is assumed that multiple alternatives may be in the product development process simultaneously and that they respond to the same set of requirements);
(b) *Version_of*: this link points from one version to another at the highest level of the artifact decomposition hierarchy where two versions differ (it is assumed that a new version supercedes a previously designed and approved version; changes in requirements leading to the new version can be represented by the same mechanism, i. e., as versions of the original requirements);
(c) *Derived_from*: similar to *Version_of,* this link allows family derivations to be represented;
(d) *Is_same_as*: at any level of the artifact decomposition hierarchy below the one where the alternative, version or derivative diverges, this link identifies a sub-artifact of the original artifact that this alternative, version or derivative is identical to;
(e) *Is_based_on*: at any level of the artifact decomposition hierarchy below the one where the alternative, version or derivative diverges, this link identifies a sub-artifact of the original artifact on which the sub-artifact of this alternative, version or derivative is based, modified so as to accommodate the new sub-artifacts of the alternative, version or derivative.

2.2.4 Utility classes

The utility package of the CPM contains the following three classes.

Information

The class **Information** is a container consisting of: (i) a textual **description** slot; (ii) a textual **documentation** string (e. g., a file path or URL referencing more substantial documentation); and (iii) a **properties** slot that contains a set of attribute-value pairs stored as strings representing all domain- or object-specific attributes. **Information** is an attribute of **CoreProductModel** and all its specializations.

ProcessInformation

The class **ProcessInformation** represents attributes related to the product development process, such as state and level, as used in [10], alternative and/or version designation or other product development process parameters that may be used in a PLM environment. **ProcessInformation** is an attribute of **Artifact**.

Rationale

The class **Rationale** represents attributes that record explanatory information on the reasons for or justifications of a particular decision in the product development process. **Rationale** is an attribute of **CoreProperty** and all its specializations.

2.2.5 Class hierarchies

All *object* classes are specializations of the abstract class **CommonCoreObject**. The attributes of **CommonCoreObject** are linkages to the **CommonCoreRelationship**, **Usage** and **Trace** relationships.

Specializations of **CommonCoreRelationship** are the **Constraint**, **EntityAssociation**, **Usage** and **Trace** classes.

2.2.6 Associations and aggregations

First, all *object* classes, i. e., specializations of the abstract class **CommonCoreObject**, except **Flow**, have their own separate, independent decomposition hierarchies, also known as "partOf" relationships or containment hierarchies[2]. Decomposition hierarchies are represented by attributes such as **subArtifacts/subArtifactOf** for the **Artifact** class.

Second, there are associations between:

(a) a **Specification** and the **Artifact** that results from it
(b) a **Flow** and its source and destination **Artifact**s and its input and output **Function**s
(c) an **Artifact** and its **Feature**s

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

(a) **Function**, **Form** and **Behavior** aggregate into **Artifact**
(b) **Function** and **Form** aggregate into **Feature**
(c) **Geometry** and **Material** aggregate into **Form**
(d) **Requirement** aggregates into **Specification**.

3 Java and XML Implementations of CPM

In order to illustrate how the CPM can be implemented and used, we have generated the equivalent Core Product XML Schema (CPXS:) and a set of Java classes. We have also developed a Java graphical user interface to input product data and generate XML documents according to the grammar of CPXS.

Appendix A and Appendix A contain, respectively, the Java classes and the CPXS.

3.1.1 The Java Artifact class

As example of the implementation, Figure 6 shows the Java **Artifact** class. In addition to the attributes shown in the figure, the **Artifact** class is participating in eight associations playing nine different roles, inherited from **CoreEntity**. These roles are converted into attributes of the Java **Artifact** class; these attributes are either simple valued or multiple values (arrays) according to the multiplicity at the end of the association. Furthermore, the inherited attributes **name**, **type** and **information** are not shown.

[2] For clarity, only the **subArtifacts/subArtifact_of** containment hierarchy of **Artifact** is labeled in Figures 1 and 3.

```
public class Artifact extends CoreEntity {
      public Specification satisfies ;
      public Feature hasFeature[] ;

      public Artifact subArtifactOf ;
      public Artifact subArtifacts[] ;

      public Flow hasInputFlow[] ;
      public Flow hasOutputFlow[] ;

      public Function hasFunction[] ;
      public Form hasForm[] ;
      public Behavior hasBehavior[] ;

      public ProcessInformation processInfo ;
}
```

Figure 6: The Java Artifact class

3.1.2 The XML Artifact complexType

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. Figure 7 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., inside 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 constraints for XML document consistency, consider the element (tag) **satisfies** inside the **Artifact** element. This tag references the name or code of the **Specification** element that the artifact satisfies; moreover 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 7: The XML Artifact complexType

To ensure this consistency, we define the **name** element to be the key of the **specification** element (Figure 8) and indicate that the element **satisfies** of the **artifact** element is a reference to this key. The figure also shows that the element **containedIn** in **requirement** 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 8: Example of a consistency constraint in the XML artifact schema

3.1.3 Example: XML representation of a planetary gear

The planetary gear system (PGS) example considered in this section was presented in detail in the Open Assembly Model report [11], illustrating the representation of both

containment hierarchy and the assembly. Our interest in the example here is to show how the revised CPM can be used to capture design information about the product; thus, only data important from the design point of view are represented.

Figure 9 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 such as function, form, behavior and specification, related to these subartifacts are not considered in this example.

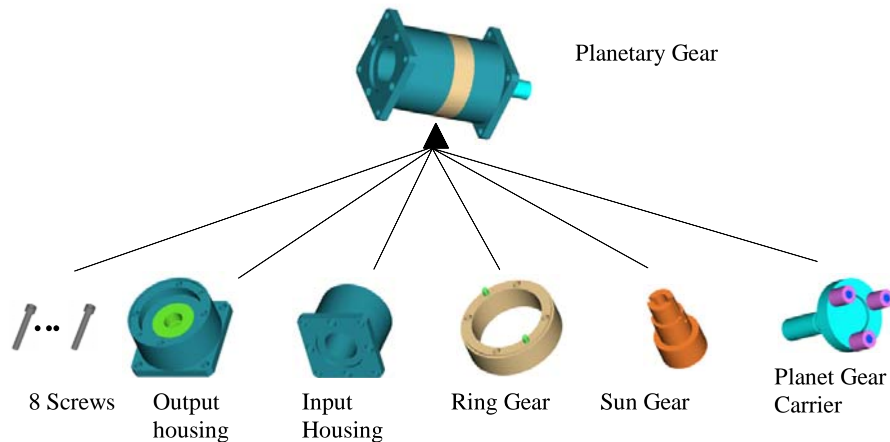

Figure 9: The planetary gear system

Figure 10 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
- **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 **from**, "*cylindricalForm*", and the **features**, "*fasteningHoles*, *outputShaftHole*" of the PGS.
- **satisfies**: this is the name of the XML element that gives the **Specification** that the PGS shall satisfy. Appendix C shows how this specification is decomposed into a set of requirements including: form, input speed, output speed, input torque and output torque requirements.
- **subartifacts**: this list gives the names of subartifacts of the current PGS artifact: planetGearCarrier, sunGear, ringGear, inputHousing, outputHousing and eight screws.

```
<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 10: An XML element representing the planetary gear system

Figure 11 shows the element named **pgsBehavior**. 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 11: Behavior element of the planetary gear system

The complete XML instance document of this example is presented in Appendix C, where more detail about the components, functions, forms, behaviors, etc. of the PGS can be found.

4 Model compilation for the CPM

CPM is a conceptual model intended for representing product design information from an engineer's point of view and not necessarily for the implementation of large-scale Product Lifecycle Management (PLM) information support systems.

Specifically, as discussed in Section 2.1, CPM uses the special attributes **type** and **properties** to record user-defined artifacts, exemplified in the instance diagram of Figure 12.

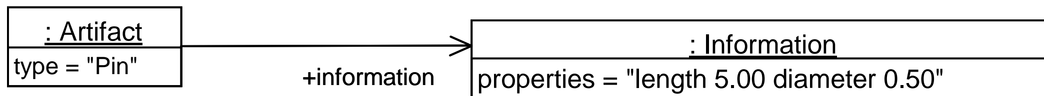

Figure 12: CPM instance of attributes

This figure is an instance of the class diagram shown in Figure 13, showing an artifact of type “pin” that has specified length and diameter attribute values. The values of these slots are delimited strings representing user-defined subtypes of **Artifact** and their properties.

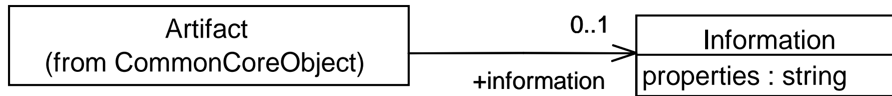

Figure 13: CPM class diagram of attributes

Such a representation will generally be appropriate and sufficient for the early conceptual phases of design, where typically there is a small number of instances and few attributes of interest for each instance. This representation will not scale to an implementation model, where thousands of instances may occur, each with a long list of application-specific attributes.

For application to industrial-scale systems, the conceptual model of CPM must be translated to an implementation model. This is called *model compilation* and is a part of the overall Model-driven Architecture (MDA) defined by the Object Management Group (OMG) [12]. MDA provides for translation of platform-independent models (PIMs), such as the CPM, to platform-specific models (PSMs) and for the generation of efficient implementation languages. For example, the **type/properties** representation above is inefficient because fast attribute value lookup can only be obtained with preparation at compile time. If the model compiler is able to tell where each attribute will be stored at runtime, it can compile each access to an attribute into retrieval from that predefined location, rather than repeating the runtime lookup at each retrieval. This requires user-defined attributes to be translated to a compilable language, such as Java. Specifically, the model compiler creates subclasses of **Artifact** from the specifications in the **type** slot, and defines attributes on the subclasses. These subclasses could be generated into a UML repository [13], as a PSM, then into a compilable language. This provides flexibility in choosing an implementation language. The end result in Java, for example, is shown in Figure 14.[3]

```
class Artifact
{
Form form;
Function function ;
…
}

class Pin extends Artifact
{
string length;
string diameter;
}
```

Figure 14: Java code generated from an attribute instance

Once the code above is compiled, a design tool can be used to instantiate the **Pin** class for specific pin designs, such as modeled in Figure 13, and insert values into the length and diameter slots. These values can be accessed efficiently because the attribute locations

[3] Other aspects of CPM's **Information** class can be translated to features on either **CommonCoreObject** or **Artifact**. For example, **documentation** can be an attribute of **CommonCoreObject**, while methods can be translated to to operations on classes such as **Pin**.

are compiled. The instances of the CPM used to begin the compilation process can be generated from graphical user interfaces, existing databases of designs, UML repositories, or whatever source has the information required. The model compiler can produce languages other than Java, for example, DDL for relational databases. There may be as many attribute-value implementations as there are implementations of the conceptual model of the CPM, which covers all of them because it is conceptual.

Another application of model compilation in the CPM is consistency maintainance of user-defined attribute values of an artifact with those of other instances of the CPM around it, such as instances of **Geometry**, **Material**, and so on. For example, Figure 15 shows a UML model for the Pin artifact, with derivation of user-defined attributes of an artifact from other instances of the CPM around it, expressed in UML's Object Constraint Language (OCL) [14]. This provides more efficient lookup of commonly used attribute values directly from an artifact, rather than navigating through the objects around it. The generated code in Figure 14 would include accessor operations that maintain consistency by propagating values from instances of **Geometry**, and so on, to the artifact attributes, or in the other direction, or both, obeying the constraints specified in OCL.[4]

Pin : Artifact
length = form.geometry.information.property.length diameter = form.geometry.information.property.diameter material = form.material.information.property.materialtype breakingStength = function.information.property.breakingStrength

Figure 15: Derivation path of attributes

Finally, model compilation can be used to translate CPM's delegation-style of reusing designs to the type/instance style of computational modeling. CPM uses **Artifact** for the representation of three things:

1. Description of classes of physical objects. For example, the design of a particular kind of gear box.

2. Use of these descriptions in composing designs for other physical objects, for example, the use of a particular gear box design in the description of a certain model of car.

3. Descriptions of physical objects conforming to the designs above. For example, maintenance record for an individual, physical gear box, with serial number 3463, installed in a particular car with VIN number 92345645.

The use of **Artifact** for all three reflects the engineer's viewpoint that they represent different stages in the lifecycle of the same artifact. These stages are related by associations in the CPM, such as the **Usage** association, which relates stages 1 and 2 above. Each stage may have different attribute values and even different attributes. For

[4] If a parameterized CAD model is available, an **Artifact** can refer to that model and its parameter values, rather than to the rest of the CPM model.

example, a stage 1 artifact might have an attribute for the designer's name, a stage 2 artifact will describe its relations to other artifacts of the design in which is being composed, and a stage 3 artifact might have attributes about its date of manufacture, the owner, and physical wear characteristics.

Computational models, on the other hand, usually have distinct elements for each of the above stages, called *type* (or *class*), *usage* (or *role*), and *instance*.[5] These reflect common information system construction practices of using program development environments to define the shapes of data structures (types, stage 1), and monitoring the execution of those programs in a separate debugging environment to find the actual data stored in those structures (instances, stage 3). Modern modeling techniques introduce usages or roles to more reliably compose designs (usages, stage 2) [16].

Model compilers can bridge the engineering and computational viewpoints by storing the rules by which the three stages are distinguished in the engineering model, using these to categorize artifacts, and generate the corresponding computational models. For example, in the CPM, an artifact that is used by other, but not used itself, is a stage 1 artifact. A model compiler can use this rule to determine which CPM artifacts should be translated to types in the computational models. Likewise, a stage 3 artifact is one that uses other artifacts that in turn use other unused artifacts. A model compiler can translate artifacts matching this rule to instances. The complete rule set and comparison of engineering and computational approaches will be the topic of future work.

5 Future work

The revisions leading from the initial CPM [1] to the model described above have all arisen from the experience gained in using the CPM as the basis for a number of extensions and applications, notably, the Design-Analysis Integration project [17] and the Open Assembly Model project described in [11] and in the companion report [9].

It is anticipated that a major part of the future work will continue to involve (a) extensions and applications that attempt to use the CPM as the basic organizing principle; and (b) generalizations, revisions, modifications and extensions of the CPM proper in light of the experience gained from such extensions and applications. The CPM is by no means mature or complete, as indicated by the need to produce the present revised version three years after the initial version.

Future work will also include review of the research literature on design and product modeling, as well as reviews of existing product modeling systems, aimed at identifying additional objects, relationships and shared attributes that may be added to the next revision of CPM. The overall objective is to find ways of extending the CPM so that it can eventually serve as the central model for collecting, correlating and organizing all product information throughout the entire product lifecycle from conception to disposal.

[5] Some computational models use one element as CPM does, but distinguish the three stages by a special attribute, for example, MOOD in the Health Level 7 Reference Information Model [15].

6 Summary and conclusions

The report documents the revised version of the Core Product Model initially presented in [1], together with brief descriptions of two pilot implementations and some preliminary thoughts about the use of a Model-driven Architecture for converting conceptual product models based on the CPM to robust implementation models. This report is to be viewed as a progress report, as it is expected that experience with further extensions and applications will give rise to further generalizations, revisions, modifications and extensions of the CPM .

References

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

2. Szykman, S., Fenves, S. J., Keirouz, W. T., and Shooter, S., *A foundation for interoperability in next-generation product development systems*, *Computer-Aided Design*, Vol. 33, No. 7, 2001, pp. 545-559.

3. Szykman, S., Racz, J. W., and Sriram, R. D., *The Representation Of Function In Computer-Based Design , Las Vegas, Nevada*, 1999.

4. Szykman, S., Racz, J. W., Bochenek, C., and Sriram, R. D., *A Web-based System for Design Artifact Modeling, Design Studies*, Vol. 21, No. 2, 2000.

5. *MOKA: A Framework for structuring and representing engineering knowledge*. http://www.kbe.coventry.ac.uk/moka/miginfo.htm . 1999.

6. Chen, P. P., The Entity-Relationship Model: Toward a Unified View of Data, *ACM Transactions on Database Systems*, Vol. 1, No. 1, 1976, pp. 9-36.

7. Booch, G., Rumbaugh, J., and Jacobson, I., *The Unified Modeling Language User Guide*, Addison-Wesley 1997.

8. Verrijn-Stuart A A. FRISCO - A framework of information system concepts - The Revised FRISCO Report (Draft January 2001), IFIP WG 8.1 Task group FRISCO. 2001.

9. Sudarsan, R., Baysal, M. M., Roy, U., Foufou, S., Bock, C., Fenves, S. J., and Sriram, R. D., *Information Models for Product Representation: Core and Assembly Models,* National Institute of Standards and Technology, NISTIR 7173, Gaithersburg, MD 20899, USA, 2004.

10. Shooter, S. B., Keirouz, W. T., Szykman, S., and Fenves, S. J., *A Model For Information Flow In Design*, Proceedings of DETC2000: ASME International Design Engineering Technical Conferences, Baltimore, Maryland, 2000.

11. 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.

12. *OMG: Model-driven Architecture*. 2004. http://www.omg.org/mda/

13. Bock, C., *UML without Pictures,* IEEE Software Special issue on Model-driven Development, 2004.

14. *OMG: UML 2.0 OCL Specification*. 2004. http://www.omg.org/cgi-bin/doc?ptc/2003-09-15

15. Health Level 7. HL7 Reference Model. 2004.

16. *OMG: UML 2.0 Superstructure Specification*. 2003. http://www.omg.org/cgi-bin/doc?ptc/2003-08-02 http://www.hl7.org/

17. Fenves, S. J., Choi, Y., Gurumoorthy, B., Mocko, G., and Sriram, R. D., *Master Product Model for the Support of Tighter Design-Analysis Integration*, National Institute of Standards and Technology, Gaithersburg, MD 20899, NISTIR 7004, 2003.

Appendix A : Java implementation of the CPM

This is not a complete working implementation; the following Java classes are provided only to give an idea of possible implementations.

CPM2 Java Classes

```
package cpm2 ;
import utility.Information ;
import utility.ProcessInformation;

public class TransferFunction extends Function {
      public Flow inputFlow[] ;
      public Flow outputFlow[] ;
}

public class Artifact extends CoreEntity {
      public Specification satisfies ;
      public Feature hasFeature[] ;

      public Artifact subArtifactOf ;
      public Artifact subArtifacts[] ;

      public Flow hasInputFlow[] ;
      public Flow hasOutputFlow[] ;

      public Function hasFunction[] ;
      public Form hasForm[] ;
      public Behavior hasBehavior[] ;

      public ProcessInformation processInfo ;
}

public class Behavior extends CommonCoreObject{
      public Artifact behaviorOfArtifact ;
}

public abstract class CommonCoreObject extends CoreProductModel {
      public CommonCoreRelationship ccRelation[] ;
}

public abstract class CoreEntity extends CommonCoreObject{
      public EntityAssociation entityAssociation[]  ;
}

public abstract class CoreProductModel {
      public String type ;
      public String name ;
      public Information information ;
}

public abstract class CoreProperty extends CommonCoreObject{
      public Constraint constraint[];
      public Requirement hasRequirement[] ;
}
```

```
public class Feature extends CoreEntity {
      public Artifact artifact;
      public Feature subFeatureOf ;
      public Feature subFeatures[] ;

      public Function featureHasFunction[] ;
      public Form featureHasForm[] ;
}

public class Flow extends CoreProperty {
      public Artifact isDestinationOf  ;
      public Artifact isSourceOf  ;
      public TransferFunction destinationOf  ;
      public TransferFunction sourceOf  ;
}

public class Form extends CoreProperty {
      public Feature formOfFeature;
      public Artifact formOfArtifact  ;
      public Form subForms[] ;
      public Form subFormOf  ;
      public Geometry hasGeometry[] ;
      public Material hasMaterial [] ;
}

public class Function extends CoreProperty {
      public Feature functionOfFeature ;
      public Artifact functionOfArtifact ;
      public Function subFunctions[] ;
      public Function subFunctionOf ;
}

public class Geometry extends CoreProperty {
      public Form geometryOfForm;
      public Geometry subGeometries[] ;
      public Geometry subGeometryOf  ;
}

public class Material extends CoreProperty {
      public Form materialOfForm;
      public Material subMaterials[] ;
      public Material subMaterialOf ;
}

public class Port extends Feature {

}

public class Requirement extends CommonCoreObject{
      public Specification containedIn;
      public CoreProperty requiredProperty[] ;
}

public class Specification extends CommonCoreObject{
      public Requirement decomposedInto[] ;
      public Artifact specifies[] ;
```

```
}

package utility ;

public class Information {
      public String description ;
      public String documentation ;
      public String Properties ;
}

public class ProcessInformation {

}

public class Rationale {

}
```

CPM2 association classes converted into Java classes

```
public class Usage extends CommonCoreRelationship {

}

public abstract class CommonCoreRelationship extends CoreProductModel {
      public CommonCoreObject associatedCCO[] ;
}

public class Constraint extends CommonCoreRelationship {
      public CoreProperty ConstrainedProperty[] ;
}

public class EntityAssociation extends CommonCoreRelationship {
      public CoreEntity associatedEntity[] ;
}

public class Trace extends CommonCoreRelationship {

}
```

Appendix B : CPXS of the Core Product XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
targetNamespace="http://namespace.nist.gov/msid/cpm"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:e="http://namespace.mist.gov/msid/ext"
xmlns="http://namespace.nist.gov/msid/cpm"
xmlns:cpm="http://namespace.nist.gov/msid/cpm"
elementFormDefault="qualified"
attributeFormDefault="unqualified">

<xsd:element name="model" type="Model">
```

```
<!-- ====================================
      for Behavior the name is a key
========================================= -->
<xsd:key name="PKBeh">
      <xsd:selector xpath="cpm:behavior"/>
      <xsd:field xpath="cpm:name"/>
</xsd:key>

<!-- Elements that shall reference a Behavior name -->
<xsd:keyref name="behRef" refer="PKBeh">
      <xsd:selector
xpath="./cpm:artifact/cpm:hasBehavior/cpm:Behavior"/>
      <xsd:field xpath="@name"/>
      </xsd:keyref>

<!-- ====================================
      for Requirement the name is a key
========================================= -->
<xsd:key name="PKReq">
      <xsd:selector xpath="cpm:requirement"/>
      <xsd:field xpath="cpm:name"/>
</xsd:key>

<!-- Elements that shall reference a Requirement name -->
<xsd:keyref name="reqRef" refer="PKReq">
      <xsd:selector
         xpath="cpm:specification/cpm:decomposedInto/cpm:Requirement"/>
      <xsd:field xpath="@name"/>
</xsd:keyref>

<!-- ====================================
      for Specification the name is 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>

<!-- =======================================
      for Artifact the name is a key
======================================= -->
<xsd:key name="PKArt">
      <xsd:selector xpath="cpm:artifact"/>
      <xsd:field xpath="cpm:name"/>
</xsd:key>

<!-- Elements that shall reference an Artifact name -->
```

```
<xsd:keyref name="art1Ref" refer="PKArt">
      <xsd:selector xpath="cpm:function"/>
      <xsd:field xpath="cpm:functionOfArtifact"/>
</xsd:keyref>
<xsd:keyref name="art2Ref" refer="PKArt">
      <xsd:selector xpath="cpm:form"/>
      <xsd:field xpath="cpm:formOfArtifact"/>
</xsd:keyref>
<xsd:keyref name="art3Ref" refer="PKArt">
      <xsd:selector xpath="cpm:flow"/>
      <xsd:field xpath="cpm:isSourceOf"/>
</xsd:keyref>
<xsd:keyref name="art4Ref" refer="PKArt">
      <xsd:selector xpath="cpm:flow"/>
      <xsd:field xpath="cpm:isDestinationOf"/>
</xsd:keyref>
<xsd:keyref name="art5Ref" refer="PKArt">
      <xsd:selector xpath="cpm:feature"/>
      <xsd:field xpath="cpm:featureOfArtifact"/>
</xsd:keyref>
<xsd:keyref name="art6Ref" refer="PKArt">
      <xsd:selector xpath="cpm:behavior"/>
      <xsd:field xpath="cpm:behaviorOfArtifact"/>
      </xsd:keyref>
<xsd:keyref name="art7Ref" refer="PKArt">
      <xsd:selector
xpath="./cpm:specification/cpm:specifies/cpm:Artifact"/>
      <xsd:field xpath="@name"/>
</xsd:keyref>
<xsd:keyref name="art8Ref" refer="PKArt">
    <xsd:selector xpath="cpm:artifact"/>
    <xsd:field xpath="cpm:subArtifactOf"/>
</xsd:keyref>
<xsd:keyref name="art9Ref" refer="PKArt">
    <xsd:selector
xpath="./cpm:artifact/cpm:subArtifacts/cpm:Artifact"/>
    <xsd:field xpath="@name"/>
</xsd:keyref>

<!-- ========================================
   for Feature the name is a key
======================================== -->
<xsd:key name="PKFeat">
    <xsd:selector xpath="cpm:feature"/>
    <xsd:field xpath="cpm:name"/>
</xsd:key>

   <!-- Elements that shall reference a Feature name -->
   <xsd:keyref name="featRef" refer="PKFeat">
    <xsd:selector xpath="cpm:function"/>
    <xsd:field xpath="cpm:functionOfFeature"/>
   </xsd:keyref>
   <xsd:keyref name="feat2Ref" refer="PKFeat">
    <xsd:selector xpath="cpm:form"/>
    <xsd:field xpath="cpm:formOfFeature"/>
   </xsd:keyref>
   <xsd:keyref name="subFeatRef" refer="PKFeat">
```

```
     <xsd:selector xpath="cpm:feature"/>
     <xsd:field xpath="cpm:subFeatureOf"/>
    </xsd:keyref>
    <xsd:keyref name="featNameRef" refer="PKFeat">
     <xsd:selector xpath="./cpm:feature/cpm:subFeatures/cpm:Feature"/>
     <xsd:field xpath="@name"/>
    </xsd:keyref>
    <xsd:keyref name="feat2NameRef" refer="PKFeat">
     <xsd:selector xpath="./cpm:artifact/cpm:hasFeature/cpm:Feature"/>
     <xsd:field xpath="@name"/>
    </xsd:keyref>



    <!-- ================== ==================
     for TransferFunction the name is a key
    ======================================== -->
    <xsd:key name="PKTFun">
     <xsd:selector xpath="cpm:transferFunction"/>
     <xsd:field xpath="cpm:name"/>
    </xsd:key>

    <!-- Elements that shall reference a TransferFunction name -->
    <xsd:keyref name="tfunOfRef" refer="PKTFun">
     <xsd:selector xpath="cpm:flow"/>
     <xsd:field xpath="cpm:sourceOf"/>
    </xsd:keyref>
    <xsd:keyref name="tfun2OfRef" refer="PKTFun">
     <xsd:selector xpath="cpm:flow"/>
     <xsd:field xpath="cpm:destinationOf"/>
    </xsd:keyref>

             <!-- ====================================
     for Flow the name is a key
    ======================== ================== -->
    <xsd:key name="PKFlow">
     <xsd:selector xpath="cpm:flow"/>
     <xsd:field xpath="cpm:name"/>
    </xsd:key>

    <!-- Elements that shall reference a Flow name -->
    <xsd:keyref name="flow0NameRef" refer="PKFlow">
     <xsd:selector xpath="./cpm:artifact/cpm:hasInputFlow/cpm:Flow"/>
     <xsd:field xpath="@name"/>
    </xsd:keyref>
    <xsd:keyref name="flow1NameRef" refer="PKFlow">
     <xsd:selector xpath="./cpm:artifact/cpm:hasOutputFlow/cpm:Flow"/>
     <xsd:field xpath="@name"/>
    </xsd:keyref>
    <xsd:keyref name="flow2NameRef" refer="PKFlow">
     <xsd:selector
xpath="./cpm:transferFunction/cpm:inputFlow/cpm:Flow"/>
     <xsd:field xpath="@name"/>
    </xsd:keyref>
    <xsd:keyref name="flow3NameRef" refer="PKFlow">
     <xsd:selector
xpath="./cpm:transferFunction/cpm:outputFlow/cpm:Flow"/>
```

```
 <xsd:field xpath="@name"/>
</xsd:keyref>

         <!-- ====================================
 for Form the name is a key
========================================- -->
<xsd:key name="PKForm">
 <xsd:selector xpath="cpm:form"/>
 <xsd:field xpath="cpm:name"/>
</xsd:key>

<!-- Elements that shall reference a Form name -->
<xsd:keyref name="formOfRef" refer="PKForm">
 <xsd:selector xpath="cpm:geometry"/>
 <xsd:field xpath="cpm:geometryOfForm"/>
</xsd:keyref>

<xsd:keyref name="fromOfRef" refer="PKForm">
 <xsd:selector xpath="cpm:material"/>
 <xsd:field xpath="cpm:materialOfForm"/>
</xsd:keyref>
<xsd:keyref name="subFromRef" refer="PKForm">
 <xsd:selector xpath="cpm:form"/>
 <xsd:field xpath="cpm:subFormOf"/>
</xsd:keyref>
<xsd:keyref name="formNameRef" refer="PKForm">
 <xsd:selector xpath="./cpm:form/cpm:subForms/cpm:Form"/>
 <xsd:field xpath="@name"/>
</xsd:keyref>
<xsd:keyref name="form2NameRef" refer="PKForm">
 <xsd:selector xpath="./cpm:artifact/cpm:hasForm/cpm:Form"/>
 <xsd:field xpath="@name"/>
</xsd:keyref>
<xsd:keyref name="form3NameRef" refer="PKForm">
 <xsd:selector xpath="./cpm:feature/cpm:featureHasForm/cpm:Form"/>
 <xsd:field xpath="@name"/>
</xsd:keyref>

<!-- ========================================-
 for Geometry the name is a key
========================================- -->
<xsd:key name="PKGeom">
 <xsd:selector xpath="cpm:geometry"/>
 <xsd:field xpath="cpm:name"/>
</xsd:key>

<!-- Elements that shall reference a Geometry name -->
<xsd:keyref name="geomRef" refer="PKGeom">
 <xsd:selector xpath="./cpm:form/cpm:hasGeometry/cpm:Geometry"/>
 <xsd:field xpath="@name"/>
</xsd:keyref>
<xsd:keyref name="subGeomRef" refer="PKGeom">
 <xsd:selector xpath="cpm:geometry"/>
 <xsd:field xpath="cpm:subGeometryOf"/>
</xsd:keyref>
<xsd:keyref name="geomNameRef" refer="PKGeom">
```

```
    <xsd:selector
xpath="./cpm:geometry/cpm:subGeometries/cpm:Geometry"/>
    <xsd:field xpath="@name"/>
   </xsd:keyref>

   <!-- ========================================-
    for Material the name is a key
   ========================================- -->
   <xsd:key name="PKMat">
    <xsd:selector xpath="cpm:material"/>
    <xsd:field xpath="cpm:name"/>
   </xsd:key>

   <!-- Elements that shall reference a Material name -->
   <xsd:keyref name="matRef" refer="PKMat">
    <xsd:selector xpath="cpm:form/cpm:hasMaterial/cpm:Material"/>
    <xsd:field xpath="@name"/>
   </xsd:keyref>
   <xsd:keyref name="subMatRef" refer="PKMat">
    <xsd:selector xpath="cpm:material"/>
    <xsd:field xpath="cpm:subMaterialOf"/>
   </xsd:keyref>
   <xsd:keyref name="matNameRef" refer="PKMat">
    <xsd:selector
xpath="./cpm:material/cpm:subMaterials/cpm:Material"/>
    <xsd:field xpath="@name"/>
   </xsd:keyref>

   <!-- ========================================-
    for Function the name is a key
   ======================================= -->
   <xsd:key name="PKFun">
    <xsd:selector xpath="cpm:function"/>
    <xsd:field xpath="cpm:name"/>
   </xsd:key>

   <!-- Elements that shall reference a Function name -->
   <xsd:keyref name="subFunRef" refer="PKFun">
    <xsd:selector xpath="cpm:function"/>
    <xsd:field xpath="cpm:subFunctionOf"/>
   </xsd:keyref>
   <xsd:keyref name="funNameRef" refer="PKFun">
    <xsd:selector
xpath="./cpm:function/cpm:subFunctions/cpm:Function"/>
    <xsd:field xpath="@name"/>
   </xsd:keyref>
   <xsd:keyref name="funName2Ref" refer="PKFun">
    <xsd:selector xpath="./cpm:artifact/cpm:hasFunction/cpm:Function"/>
    <xsd:field xpath="@name"/>
   </xsd:keyref>
   <xsd:keyref name="funName3Ref" refer="PKFun">
    <xsd:selector
xpath="./cpm:feature/cpm:featureHasFunction/cpm:Function"/>
    <xsd:field xpath="@name"/>
   </xsd:keyref>
  </xsd:element>
```

```
 <!-- This is the main type containing all other types -->
 <xsd:complexType name="Model">
  <xsd:sequence>
   <xsd:choice minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="form" type="Form"/>
    <xsd:element name="geometry" type="Geometry"/>
    <xsd:element name="material" type="Material"/>
    <xsd:element name="function" type="Function"/>
    <xsd:element name="transferFunction" type="TransferFunction"/>
    <xsd:element name="flow" type="Flow"/>
    <xsd:element name="artifact" type="Artifact"/>
    <xsd:element name="feature" type="Feature"/>
    <xsd:element name="specification" type="Specification"/>
    <xsd:element name="requirement" type="Requirement"/>
    <xsd:element name="behavior" type="Behavior"/>
   </xsd:choice>
  </xsd:sequence>
 </xsd:complexType>

 <!-- Core Product Model-->
 <xsd:complexType name="CoreProductModel" abstract="true">
  <xsd:sequence>
   <xsd:element name="type" type="xsd:string" minOccurs="0"/>
   <xsd:element name="name" type="xsd:string"/>
   <xsd:element name="information" type="Information" minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>

 <!-- Information -->
 <xsd:complexType name="Information">
  <xsd:sequence>
   <xsd:element name="description" type="xsd:string" minOccurs="0"/>
   <xsd:element name="documentation" type="xsd:string" minOccurs="0"/>
   <xsd:element name="properties" type="Properties" minOccurs="0"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="Properties">
  <xsd:sequence>
   <xsd:element name="property" type="Property" minOccurs="1"
maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="Property">
  <xsd:simpleContent>
   <xsd:extension base="xsd:string">
    <xsd:attribute name="name" type="xsd:string" use="required"/>
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>

 <!--
====================================================================
In  the following classes (CommonCoreObject, CommonCoreRelationship,
Constraint, EntityAssociation, CoreEntity, CoreProperty, Requirement)
some elements are commented. This is to avoid errors
```

```
during schema validations, these elements are of type ListOfxx, xx
stands for an abstract class. As XML doesn't allow any instanciations
of abstract classes, the commented elements can not be used, but are
provided to show how some object oriented programming of the UML model
can not be faithfully represented using XML schema.
======================================================================-->

 <!-- CommonCoreObject -->
 <xsd:complexType name="CommonCoreObject"  abstract="true">
  <xsd:complexContent>
   <xsd:extension base="CoreProductModel">
    <xsd:sequence>
     <!--<xsd:element name="commonCoreRelationship"
type="ListOfCCRelations" minOccurs="0"/>-->
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- CommonCoreRelationship -->
 <xsd:complexType name="CommonCoreRelationship"  abstract="true">
  <xsd:complexContent>
   <xsd:extension base="CoreProductModel">
    <xsd:sequence>
    <!--<xsd:element name="associatedCCObject" type="ListOfCCObjetcs"
minOccurs="0"/>-->
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- Constraint -->
 <xsd:complexType name="Constraint">
  <xsd:complexContent>
   <xsd:extension base="CommonCoreRelationship">
    <xsd:sequence>
     <!--<xsd:element name="constrainedProperty"
type="ListOfCoreProperties" minOccurs="0"/>-->
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- EntityAssociation -->
 <xsd:complexType name="EntityAssociation">
  <xsd:complexContent>
   <xsd:extension base="CommonCoreRelationship">
    <xsd:sequence>
     <!--<xsd:element name="associatedEntity" type="ListOfCoreEntities"
minOccurs="0"/>-->
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- CoreEntity -->
 <xsd:complexType name="CoreEntity"  abstract="true">
```

```
  <xsd:complexContent>
   <xsd:extension base="CommonCoreObject">
    <xsd:sequence>
     <!--<xsd:element name="entityAssociation"
type="ListOfEntityAssociations" minOccurs="0"/>-->
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- CoreProperty -->
 <xsd:complexType name="CoreProperty"  abstract="true">
  <xsd:complexContent>
   <xsd:extension base="CommonCoreObject">
    <!--<xsd:element name="constraint" type="ListOfConstraints"
minOccurs="0"/>-->
    <!--<xsd:element name="hasRequirement" type="ListOfRequirements"
minOccurs="0"/>-->
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- Requirement -->
 <xsd:complexType name="Requirement">
  <xsd:complexContent>
   <xsd:extension base="CommonCoreObject">
    <xsd:sequence>
     <xsd:element name="containedIn" type="xsd:string" minOccurs="1"/>
     <!--<xsd:element name="requiredProperty"
type="ListOfCoreProperties" minOccurs="0"/>-->
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- Specification -->
 <xsd:complexType name="Specification">
  <xsd:complexContent>
   <xsd:extension base="CommonCoreObject">
                <xsd:sequence>
                    <xsd:element name="specifies"
type="ListOfArtifacts"
                        minOccurs="0"/>
                    <xsd:element name="decomposedInto"
type="ListOfRequirements"
                        minOccurs="0"/>
                </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- Behavior -->
 <xsd:complexType name="Behavior">
  <xsd:complexContent>
   <xsd:extension base="CommonCoreObject">
    <xsd:sequence>
```

```
     <xsd:element name="behaviorOfArtifact" type="xsd:string"
minOccurs="1"/>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- Artifact -->
 <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>

 <!-- Feature -->
 <xsd:complexType name="Feature">
  <xsd:complexContent>
   <xsd:extension base="CoreEntity">
    <xsd:sequence>
     <xsd:element name="featureOfArtifact" type="xsd:string"
minOccurs="1"/>
     <xsd:element name="featureHasFunction" type="ListOfFunctions"
minOccurs="0"/>
     <xsd:element name="featureHasForm" type="ListOfForms"
minOccurs="0"/>
     <xsd:element name="subFeatures" type="ListOfFeatures"
minOccurs="0"/>

     <xsd:element name="subFeatureOf" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>
```

```
 <!-- Port -->
 <xsd:complexType name="Port">
  <xsd:complexContent>
   <xsd:extension base="Feature">
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- Form -->
 <xsd:complexType name="Form">
  <xsd:complexContent>
   <xsd:extension base="CoreProperty">
    <xsd:sequence>
     <xsd:element name="hasGeometry" type="ListOfGeometries"
minOccurs="0"/>
     <xsd:element name="hasMaterial" type="ListOfMaterials"
minOccurs="0"/>
     <xsd:element name="subForms" type="ListOfForms" minOccurs="0"/>
     <xsd:element name="subFormOf" type="xsd:string" minOccurs="0"/>
     <xsd:choice minOccurs="0" maxOccurs="1">
      <xsd:element name="formOfArtifact" type="xsd:string"
minOccurs="0"/>
      <xsd:element name="formOfFeature" type="xsd:string"
minOccurs="0"/>
     </xsd:choice>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- Geometry -->
 <xsd:complexType name="Geometry">
  <xsd:complexContent>
   <xsd:extension base="CoreProperty">
    <xsd:sequence>
     <xsd:element name="subGeometries" type="ListOfGeometries"
minOccurs="0"/>
     <xsd:element name="subGeometryOf" type="xsd:string"
minOccurs="0"/>
     <xsd:element name="geometryOfForm" type="xsd:string"
minOccurs="1"/>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- Material -->
 <xsd:complexType name="Material">
  <xsd:complexContent>
   <xsd:extension base="CoreProperty">
    <xsd:sequence>
     <xsd:element name="subMaterials" type="ListOfMaterials"
minOccurs="0"/>
     <xsd:element name="subMaterialOf" type="xsd:string"
minOccurs="0"/>
     <xsd:element name="materialOfForm" type="xsd:string"
minOccurs="1"/>
```

```
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- Function -->
 <xsd:complexType name="Function">
  <xsd:complexContent>
   <xsd:extension base="CoreProperty">
    <xsd:sequence>
     <xsd:element name="subFunctions" type="ListOfFunctions"
minOccurs="0"/>
     <xsd:element name="subFunctionOf" type="xsd:string"
minOccurs="0"/>
     <xsd:choice minOccurs="0" maxOccurs="1">
      <xsd:element name="functionOfArtifact" type="xsd:string"
minOccurs="0"/>
      <xsd:element name="functionOfFeature" type="xsd:string"
minOccurs="0"/>
     </xsd:choice>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- TransferFunction -->
 <xsd:complexType name="TransferFunction">
  <xsd:complexContent>
   <xsd:extension base="Function">
    <xsd:sequence>
     <xsd:element name="inputFlow" type="ListOfFlows" minOccurs="0"/>
     <xsd:element name="outputFlow" type="ListOfFlows" minOccurs="0"/>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- Flow -->
 <xsd:complexType name="Flow">
  <xsd:complexContent>
   <xsd:extension base="CoreProperty">
    <xsd:sequence>
     <xsd:element name="sourceOf" type="xsd:string" minOccurs="1"/>
     <xsd:element name="destinationOf" type="xsd:string"
minOccurs="1"/>
     <xsd:element name="isSourceOf" type="xsd:string" minOccurs="1"/>
     <xsd:element name="isDestinationOf" type="xsd:string"
minOccurs="1"/>
    </xsd:sequence>
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>

 <!-- ListOfFlows -->
 <xsd:complexType name="ListOfFlows">
  <xsd:sequence minOccurs="1">
   <xsd:element name="Flow" minOccurs="1" maxOccurs="unbounded">
```

```
    <xsd:complexType>
     <xsd:attribute name="name" type="xsd:NCName" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <!-- ListOfForms -->
 <xsd:complexType name="ListOfForms">
  <xsd:sequence minOccurs="1">
   <xsd:element name="Form" minOccurs="1" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:attribute name="name" type="xsd:NCName" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <!-- ListOfGeometries -->
 <xsd:complexType name="ListOfGeometries">
  <xsd:sequence minOccurs="1">
   <xsd:element name="Geometry" minOccurs="1" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:attribute name="name" type="xsd:NCName" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <!-- ListOfMaterials -->
 <xsd:complexType name="ListOfMaterials">
  <xsd:sequence minOccurs="1">
   <xsd:element name="Material" minOccurs="1" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:attribute name="name" type="xsd:NCName" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <!-- ListOfFunctions -->
 <xsd:complexType name="ListOfFunctions">
  <xsd:sequence minOccurs="1">
   <xsd:element name="Function" minOccurs="1" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:attribute name="name" type="xsd:NCName" use="required"/>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <!-- ListOfBehaviors -->
 <xsd:complexType name="ListOfBehaviors">
  <xsd:sequence minOccurs="1">
   <xsd:element name="Behavior" minOccurs="1" maxOccurs="unbounded">
    <xsd:complexType>
     <xsd:attribute name="name" type="xsd:NCName" use="required"/>
```

```
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<!-- ListOfFeatures -->
<xsd:complexType name="ListOfFeatures">
 <xsd:sequence minOccurs="1">
  <xsd:element name="Feature" minOccurs="1" maxOccurs="unbounded">
   <xsd:complexType>
    <xsd:attribute name="name" type="xsd:NCName" use="required"/>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<!-- ListOfArtifacts -->
<xsd:complexType name="ListOfArtifacts">
 <xsd:sequence minOccurs="1">
  <xsd:element name="Artifact" minOccurs="1" maxOccurs="unbounded">
   <xsd:complexType>
    <xsd:attribute name="name" type="xsd:NCName" use="required"/>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<!-- ListOfRequirements -->
<xsd:complexType name="ListOfRequirements">
 <xsd:sequence minOccurs="1">
  <xsd:element name="Requirement" minOccurs="1" maxOccurs="unbounded">
   <xsd:complexType>
    <xsd:attribute name="name" type="xsd:NCName" use="required"/>
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>

<!-- ProcessInformation -->
<xsd:complexType name="ProcessInformation">
 <xsd:sequence>
 </xsd:sequence>
</xsd:complexType>

<!-- Rational -->
<xsd:complexType name="Rational">
 <xsd:sequence>
 </xsd:sequence>
</xsd:complexType>

 <!-- Trace -->
<xsd:complexType name="Trace">
 <xsd:complexContent>
  <xsd:extension base="CommonCoreRelationship">
  </xsd:extension>
 </xsd:complexContent>
</xsd:complexType>
```

```
 <!-- Usage -->
 <xsd:complexType name="Usage">
  <xsd:complexContent>
   <xsd:extension base="CommonCoreRelationship">
   </xsd:extension>
  </xsd:complexContent>
 </xsd:complexType>


</xsd:schema>
```

Appendix C : XML representation of a planetary gear

```
<?xml version="1.0" encoding="UTF-8"?>
<model xmlns="http://namespace.nist.gov/msid/cpm">

<!-- Artifact -->
<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>
      <hasBehavior>
            <Behavior name="pgsBehavior"/>
      </hasBehavior>
      <hasFunction>
            <Function name="changeSpeedOfRotation"/>
      </hasFunction>
      <hasForm>
            <Form name="cylindricalForm"/>
      </hasForm>
            <satisfies>pgsSpecification</satisfies>
      <hasFeature>
            <Feature name="fasteningHoles"/>
            <Feature name="outputShaftHole"/>
      </hasFeature>
      <subArtifacts>
            <Artifact name = "planetGearCarrier"/>
            <Artifact name = "sunGear"/>
            <Artifact name = "ringGear"/>
            <Artifact name = "inputHousing"/>
            <Artifact name = "outputHousing"/>
            <Artifact name = "screw1"/>
            <Artifact name = "screw2"/>
            <Artifact name = "screw3"/>
            <Artifact name = "screw4"/>
            <Artifact name = "screw5"/>
            <Artifact name = "screw6"/>
            <Artifact name = "screw7"/>
            <Artifact name = "screw8"/>
```

```
      </subArtifacts>
</artifact>

    <!-- Features -->
    <feature>
        <name>fasteningHoles</name>
        <information>
            <description>8 holes to contain screws</description>
            <documentation>4 holes are in the output housing side,
                the other 4 are in the input housing side. They are
                used to fasten the output housing and the input
                housing to the ring gear.
            </documentation>
        </information>
        <featureOfArtifact>PlanetaryGearSystem</featureOfArtifact>
    </feature>
    <feature>
        <name>outputShaftHole</name>
        <information>
            <description>the cylindrical stem of the output shaft
                will be inserted into this hole</description>
        </information>
        <featureOfArtifact>PlanetaryGearSystem</featureOfArtifact>
        <featureHasFunction>
            <Function name="stemInsertion"/>
        </featureHasFunction>
    </feature>

    <!-- subartifacts: These artifacts are incomlete. Their
specifications,
        behaviors, functions, forms are TBD -->
    <artifact>
        <name>planetGearCarrier</name>
        <satisfies>pgsSpecification</satisfies>
        <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>
    <artifact>
        <name>sunGear</name>
        <satisfies>pgsSpecification</satisfies>
        <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>
    <artifact>
        <name>ringGear</name>
        <satisfies>pgsSpecification</satisfies>
        <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>
    <artifact>
        <name>inputHousing</name>
        <satisfies>pgsSpecification</satisfies>
        <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>
    <artifact>
        <name>outputHousing</name>
        <satisfies>pgsSpecification</satisfies>
         <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>
    <artifact>
```

```
        <name>screw1</name>
        <satisfies>pgsSpecification</satisfies>
         <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>
    <artifact>
        <name>screw2</name>
        <satisfies>pgsSpecification</satisfies>
        <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>
    <artifact>
        <name>screw3</name>
        <satisfies>pgsSpecification</satisfies>
        <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>
    <artifact>
        <name>screw4</name>
        <satisfies>pgsSpecification</satisfies>
        <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>
    <artifact>
        <name>screw5</name>
        <satisfies>pgsSpecification</satisfies>
        <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>
    <artifact>
        <name>screw6</name>
        <satisfies>pgsSpecification</satisfies>
        <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>
    <artifact>
        <name>screw7</name>
        <satisfies>pgsSpecification</satisfies>
        <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>
    <artifact>
        <name>screw8</name>
        <satisfies>pgsSpecification</satisfies>
        <subArtifactOf>PlanetaryGearSystem</subArtifactOf>
    </artifact>

    <!-- Functions -->
    <function>
        <name>changeSpeedOfRotation</name>
        <information>
            <description>this the main function of the
PlanetaryGearSystem.
                it provides adequate and variabel 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>
        <function>
            <name>stemInsertion</name>
            <functionOfFeature>outputShaftHole</functionOfFeature>
      </function>

    <!-- Specifications -->
    <specification>
        <name>pgsSpecification</name>
        <specifies>
            <Artifact name="PlanetaryGearSystem"/>
        </specifies>
        <decomposedInto>
            <Requirement name="formSpecification"/>
            <Requirement name="inputSpeed"/>
            <Requirement name="outputSpeed"/>
            <Requirement name="inputTorque"/>
            <Requirement name="outputTorque"/>
        </decomposedInto>
    </specification>

    <!-- Requirements -->
    <requirement>
        <name>inputSpeed</name>
        <information>
            <description>input speed shall be between 900 and 1800 rpm
            </description>
        </information>
        <containedIn>pgsSpecification</containedIn>
    </requirement>
        <requirement>
        <name>outputSpeed</name>
        <information>
            <description>output speed shall be between 300 and 600 rpm
            </description>
        </information>
        <containedIn>pgsSpecification</containedIn>
    </requirement>
    <requirement>
        <name>inputTorque</name>
        <information>
           <description>input torque shall be around 2.26
N.m</description>
        </information>
        <containedIn>pgsSpecification</containedIn>
    </requirement>
    <requirement>
        <name>outputTorque</name>
        <information>
          <description>output torque shall be around 6.78
N.m</description>
        </information>
        <containedIn>pgsSpecification</containedIn>
    </requirement>
```

```
    <requirement>
        <name>formSpecification</name>
        <information>
            <description>length less than 100mm, width and height less
than
                60mm, weight less that 150g
            </description>
        </information>
        <containedIn>pgsSpecification</containedIn>
    </requirement>

    <!-- Form -->
      <form>
            <name>cylindricalForm</name>
            <information>
            <description>the form of the PlanetaryGearSystem
                shall be cylindrical
            </description>
            <documentation>length less than 100mm, width and height
less
                  than 60mm, weight less that 150g. No more details
            about
                  the geometry of this form are available
            </documentation>
            </information>
            <formOfArtifact>PlanetaryGearSystem</formOfArtifact>
      </form>

   <!-- Behavior -->
      <behavior>
            <name>pgsBehavior</name>
            <information>
            <description>the behavior of the 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>

      <behaviorOfArtifact>PlanetaryGearSystem</behaviorOfArtifact>
      </behavior>
</model>
```