A Product Information Modeling Framework For Product Lifecycle Management

S. J. Fenves, R. D. Sriram, R. Sudarsan and F. Wang*
Design and Process Group
Manufacturing Systems Integration Division
Manufacturing Engineering Laboratory
National Institute of Standards and Technology
Gaithersburg MD, USA

Abstract

We describe a framework for representing products based on the NIST Core Product Model (CPM) and its extensions, the Open Assembly Model (OAM), the Design-Analysis Integration model (DAIM), and the Product Family Evolution Model (PFEM). These are abstract models with general semantics, with the specific semantics about a particular domain to be embedded within the usage of the models for that domain. CPM represents the product's function, form and behavior, its physical and functional decompositions, and the relationships among these concepts. An extension of CPM provides a way to associate design rationale with the product. OAM defines a system level conceptual model and the associated hierarchical assembly relationships. DAIM defines a Master Model of the product and a series of abstractions called Functional Models - one for each domain-specific aspect of the product – and two transformations, called idealization and mapping, between the master model and each functional model. PFEM extends the representation to families of products and their components; it also extends design rationale to the capture of the rationale for the evolution of the families.

The framework is intended to: (1) capture product, design rationale, assembly, and tolerance information from the earliest conceptual design stage - where designers deal with the function and performance of products – to the full lifecycle; (2) facilitate the semantic interoperability of next-generation CAD/CAE/CAM systems; and (3) capture the evolution of product families. The relevance of our framework to Product Lifecycle Management (PLM) is that any data component in the framework can be accessed directly by a PLM system, providing fine-grained access to the product's description and design rationale.

1 Introduction

PLM is generally defined as "a strategic business approach for the effective management and use of corporate intellectual capital." PLM systems are gaining acceptance for managing all information about a corporation's products throughout their full lifecycle, from conceptualization to operations and disposal.

PLM systems form the apex of the corporate software hierarchy and as such depend on subsidiary systems for detailed information capture and dissemination. PLM systems tend to delegate the task of managing the information describing the product itself to Product Data Management (PDM) systems. Furthermore, in many organizations, only the geometric description of products generated by Computer-Aided Design (CAD) systems is managed directly; in these organizations PDM systems rely on the CAD systems for managing product descriptions.

The above segmentation of PLM and subsidiary software systems results in two shortcomings. First, while PLM can track changes through the products' lifecycle from conception to disposal, the information that describes the actual changes can be found only through the subsidiary PDM systems, and the reason for the changes may not be recorded in computer-processable form anywhere. Thus, there is a need to make product descriptions and their design rationale directly accessible from PLM systems, with

* *{sfenves,sriram,sudarsan,fuwang}@cme.nist.gov*

no intermediary layers of software. Second, CAD representations arise only at later stages of design, after a form has been assigned to the product concept; therefore, PLM systems tied only to CAD representations of products cannot be used before the form is assigned. In order to realize PLM's potentials, PLM systems need to interact with product information used in the early stages of conception and ideation, where designers and planners deal with the function and performance of products, and not yet with their form.

The Product Engineering Program at NIST has as its goal to "establish a semantically-based, validated product representation scheme as a standard that supports the seamless interoperability among current and next generation computer-aided design systems (CAD) and between CAD systems and other systems that generate and use product data. Specifically, the primary needs for the next generation of CAD/Computer-Aided Manufacturing (CAM)/Computer-Aided Engineering (CAE) software systems are interoperability among software tools, collaboration among distributed designers and design teams, integration of data and knowledge across the product development cycle (from design to analysis to manufacturing and beyond), as well as knowledge capture, exchange and reuse." [http://www.mel.nist.gov/msid/pe.htm]. The PLM philosophy and supporting software systems aim at providing support to an even broader range of engineering and business activities. The aim of this paper is to demonstrate that the Product Engineering Program's approach can: (1) support the full range of PLM information needs; and (2) overcome the two shortcomings of the PLM software segmentation discussed above.

2 Objectives for the Interoperability Framework

The exchange of part and assembly information between heterogeneous modeling systems is critical for collaborative design and manufacturing. Interchange standards for product geometry are in wide use. However, little has been done in terms of developing standard representations that specify the full range of design information and product knowledge. The NIST interoperability framework is intended to address this issue.

The conceptual information architecture under development at NIST has the following key attributes: (1) it is based on formal semantics, and will eventually be supported by an appropriate ontology to permit automated reasoning; (2) it is generic: it deals with conceptual entities such as artifacts and features, and not specific artifacts such as motors, pumps or gears; (3) it is to serve as a repository of a rich variety of information about products, including aspects of product description that are not currently incorporated; (4) it is intended to foster the development of novel applications and processes that were not feasible in less information-rich environments; (5) it incorporates the explicit representation of design rationale, considered to be as important as that of the product description itself; and (6) there are provisions for converting and/or interfacing the generic representation schemes into a production-level interoperability framework. An interoperability framework resulting from the application of the conceptual information architecture will: (1) provide a generic depository of all product information at all stages of the design process; (2) serve all product description information to the PLM system and its subsidiary systems using a single, uniform information exchange protocol; and (3) support direct interoperability among CAD, CAE, CAM and other interrelated systems where high bandwidth, seamless information interchange is needed.

3 Components of the Interoperability Framework

The NIST interoperability framework consists of the four major components described below.

3.1 The Core Product Model

The primary objective of the Core Product Model (CPM)[1] is to provide a base-level product model that is open, non-proprietary, generic, extensible, independent of any one product development process

and capable of capturing the full engineering context commonly shared in product development. The CPM model consists of two sets of classes, called object and relationship, equivalent to the UML [2] class and association class, respectively (throughout the paper we use the notation and class diagrams of the Unified Modeling Language (UML), we also use bold face font for UML classes and packages, where a package is a collection of classes [2] that can be used as a namespace).

Core Product Model Entities. Figure 1 illustrates the entities comprising the CPM. All entities are specializations of the abstract class **CommonCoreObject**. **CoreEntity** and **CoreProperty** are abstract classes. The former specializes into **Artifact** and **Feature**, and the latter into **Function**, **Form**, **Geometry,** and **Material**. A **DesignRationale** class (discussed in Section 3.4) is associated with **CoreProperty**.

Artifact is the aggregation of **Function**, **Form**, and **Behavior**. **Form** in turn is the aggregation of **Geometry** and **Material**. In addition, an **Artifact** has a Specification and is an aggregation of **Features**. The **Specification** is a container for the design requirements pertaining to the artifact's function or form. **Feature** represents any information in the **Artifact** that is an aggregation of **Function** and **Form.** **Artifact**, **Feature**, **Function**, **Form**, **Geometry** and **Material** are each aggregates of their own containment hierarchies (part-of relationships).

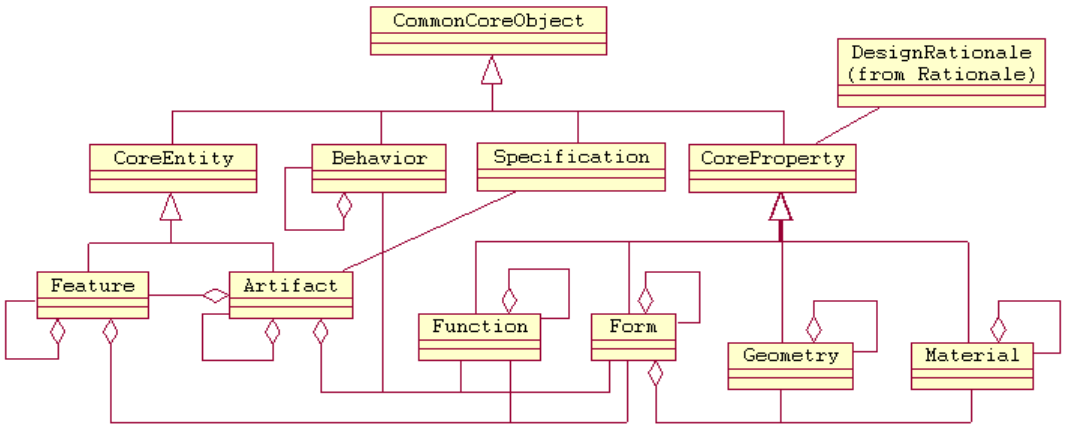

Figure 1: Entities in the Core Product Model

Semantically, **Artifact** represents a distinct entity in a design, whether that entity is the entire product or one of its subsystems or components. **Function** represents what the artifact is intended to do. The separate representation of Function renders the DAI and its extensions capable of supporting functional reasoning in the absence of any information on the artifact's form, thus providing support for the conceptual phases of design. **Form** may be viewed as the proposed design solution to the problem specified by the function and consists of the artifact's **Geometry** (shape and structure may be synonymous in some contexts) and the **Material** it is composed of. **Behavior** represents how the artifact's form implements its function and is evaluated by a causal model, such as a Finite Element Analysis (FEA). **Feature** represents a subset of the form that has some function assigned to it. CPM does not treat pure form elements as features nor does it support the independent behavior of features.

Core Product Model Relationships. Figure 2 shows the relationships in the CPM. All relationships are subclasses of the abstract class **CommonCoreRelationship** and are all UML association classes. **Requirement** is an association class between the **Specification** and a **CoreProperty** of the artifact; each requirement applies to some aspect of the function, form, geometry, material of the artifact. **Constraint** is a set of properties that share an attribute that must hold in all cases.

There are two specializations of **SetRelationship**: **UndirectedSetRelationship** groups objects into a set, while **DirectedSetRelationship** groups them into two subsets with different roles (e. g., a *controlling* subset and a *controlled-by* subset). **AssemblyRelationship** is implemented in the CPM as an undirected set of artifacts and features; it is specialized in the Open Assembly Model described below. Finally, **Reference** links or cross-references entities.

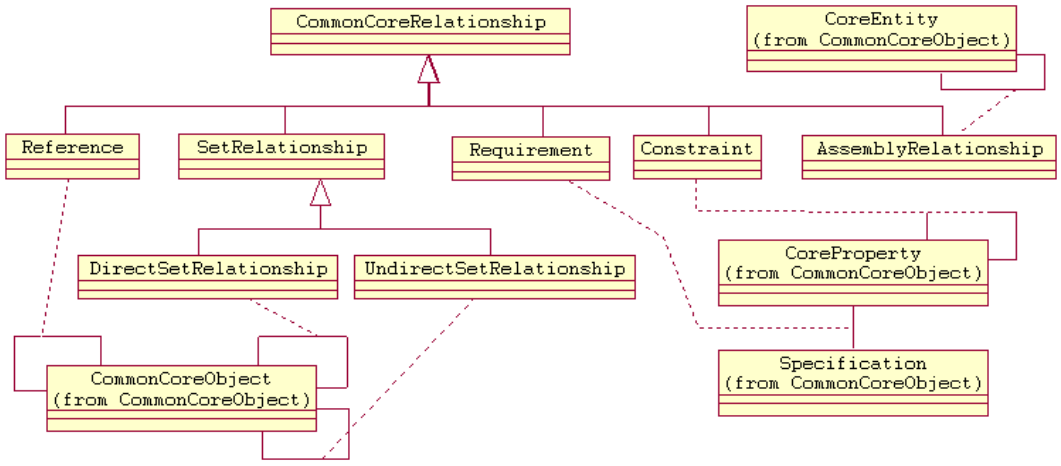

Figure 2: Relationships in the Core Product Model

3.2 The Open Assembly Model

Most electromechanical products are assemblies of components. The aim of the Open Assembly Model (OAM) is to provide a standard representation and exchange protocol for assembly and system-level tolerance information. OAM is extensible; it currently provides for tolerance representation and propagation, representation of kinematics, and engineering analysis at the system level. The assembly information model emphasizes the nature and information requirements for part features and assembly relationships. The model includes both assembly as a concept and assembly as a data structure. For the latter it uses the model data structures of ISO 10303, informally known as the STandard for the Exchange of Product model data (STEP)[3].

Figure 3 shows the main schema of the Open Assembly Model. The schema incorporates information about assembly relationships and component composition; the former is represented by the class **AssemblyAssociation** and the latter is modeled using part-of relationships. An **Assembly** is decomposed into subassemblies and parts. A **Part** is the lowest level component. Each assembly component, whether a sub-assembly or part, is made up of one or more features, represented in the model by **OAMFeature**. The **Assembly** and **Part** classes are subclasses of the CPM **Artifact** class and **OAMFeature** is a subclass of the CPM **Feature** class.

ArtifactAssociation is specialized into the following classes: **PositionOrientation, RelativeMotion** and **Connection**. **PositionOrientation** represents the relative position and orientation between two or more artifacts that are not physically connected and describes the constraints on the relative position and orientation between them. **RelativeMotion** represents the relative motions between two or more artifacts that are not physically connected and describes the constraints on the relative motions between them. **Connection** represents the connection between artifacts that are physically connected. **Connection** is further specialized as **FixedConnection, MovableConnection,** or **IntermittentConnection**. **FixedConnection** represents a connection in which the participating artifacts are physically connected and describes the type and/or properties of the fixed joints. **MovableConnection** represents the connection in which the participating artifacts are physically connected and movable with respect to one

another and describes the type and/or properties of kinematic joints. **IntermittentConnection** represents the connection in which the participating artifacts are physically connected only intermittently.

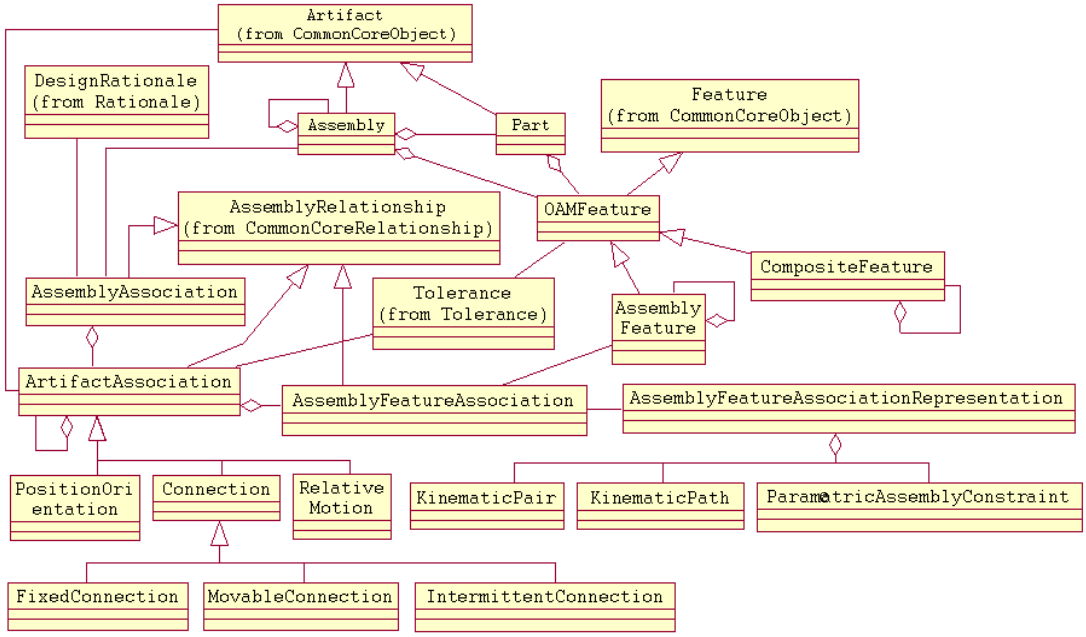

Figure 3: Main Schema of Open Assembly Model

OAMFeature has tolerance information, represented by the class **Tolerance**, and subclasses **AssemblyFeature** and **CompositeFeature**. **AssemblyFeature** specifies the relationship between assembled components. **CompositeFeature** represents a composite feature that can be decomposed into multiple simple features. **AssemblyFeatureAssociation** represents the association between mating assembly features through which relevant artifacts are associated. The assembly relationship between two or more assembly features is represented by the class **AssemblyFeatureAssociationRepresentation,** an aggregation of parametric assembly constraints, a kinematic pair, and/or a relative motion between assembly features.

ParametricAssemblyConstraint specifies explicit geometric constraints between artifacts of an assembled product, intended to control the position and orientation of artifacts in an assembly. Parametric assembly constraints are defined in ISO 10303-108 [4]). This class is further specialized into specific types: **Parallel**, **ParallelWithDimension**, **SurfaceDistanceWithDimension**, **AngleWithDimension**, **Perpendicular**, **Incidence**, **Coaxial**, **Tangent**, and **FixedComponent**.

KinematicPair defines the kinematic constraints between two adjacent artifacts (links) at a joint. The kinematic structure schema in ISO 10303-105 [5] defines the kinematic structure of a mechanical product in terms of links, pairs, and joints. The kinematic pair represents the geometric aspects of the kinematic constraints of motion between two assembled components. **KinematicPath** represents the relative motion between artifacts. The kinematic motion schema in ISO 10303-105 [5] defines kinematic motion. It is also used to represent the relative motion between artifacts.

Tolerance. Tolerancing is a critical issue in the design of electro-mechanical assemblies. Tolerancing includes both tolerance analysis and tolerance synthesis. Proactive approaches to assembly or tolerance analysis in the early design stages will involve making decisions with incomplete information. In order to support early tolerance synthesis and analysis in the conceptual product design stage, we include

function, tolerance, and behavior information in the assembly model to allow analysis and synthesis of tolerances, even with incomplete data sets.

DimensionalTolerance typically controls the variability of linear dimensions that describe location, size, and angle; it is also known as tolerancing of perfect form. This is included to accommodate the ISO 1101 standard [6]. **GeometricTolerance** is the general term applied to the category of tolerances used to control shape, position, and runout. It enables tolerances to be placed on attributes of features, where a feature is one or more pieces of a part surface; feature attributes include size (for certain features), position (certain features), form (flatness, cylindricity, etc.), and relationship (e.g. perpendicular-to). The class **GeometricTolerance** is further specialized into the following: (1) **FormTolerance**; (2) **ProfileTolerance**; (3) **RunoutTolerance**; (4) **OrientationTolerance**; and (5) **LocationTolerance**.

Datum is a theoretically exact or a simulated piece of geometry, such as a point, line, or plane, from which a tolerance is referenced. **DatumFeature** is a physical feature that is applied to establish a datum. **FeatureOfSize** is a feature that is associated with a size dimension, such as the diameter of a spherical or cylindrical surface or the distance between two parallel planes. **StatisticalControl** is a specification that incorporates statistical process controls on the toleranced feature in manufacturing.

3.3 The Design–Analysis Integration Model

The Computer-Aided Design of a product's geometry and Computer-Aided Engineering for the analysis of its behavior are in common use today. However, the integration of the efforts of the professionals in the two disciplines is not as complete as it should be, resulting in the limited interoperability of the two sets of tools. Typically, a product's behavior needs to be analyzed in several functional domains (e. g., structural, thermal, kinetics, economics) and the results of the analyses may suggest design changes for improving or optimizing the behavior.

The Design-Analysis Integration Model (DAIM) is a conceptual data architecture that provides the technical basis for tighter design-analysis integration than is possible with today's tools and information models. It is also intended to make analysis-driven design (often referred to as form-to-function reasoning) more practical. Eventually, it should also support opportunistic analysis, where the system tracks the geometric design process and notifies the designer when sufficient geometric information has been generated to initiate a functional analysis [7].

The class diagram of the DAIM is shown in Figure 4. The **MasterModel** and the **FunctionalModel** are both specializations of the CPM **Artifact** class. The Master Model serves as the global repository of information on a product being designed; it may be implemented as a centralized distributed database, or virtual database. Each **FunctionalModel** represents an abstraction of the product of interest to a specific functional domain. The figure shows three representative specializations: a **StrengthView** for finite element modeling and analysis; a **ShapeView** for classical CAD geometry modeling; and a **KinematicsView** for kinematic modeling and analysis. The two models are linked by two association classes. **Idealization** is the transformation that creates a functional model specific to a particular domain from the master model; this is typically an abstraction operation removing detail irrelevant to the particular function, but more general transformations may also be used. **Mapping** is the reverse transformation of updating the master model based on changes in the domain-specific functional model; it is conceptually the more difficult transformation to define and implement, as it is responsible for maintaining full consistency between the two models.

3.4 The Product Family Evolution Model

Many manufacturing concerns develop product families so as to offer a variety of products with reduced development costs [8]. The Product Family Evolution Model (PFEM) represents the evolution of product

families and of the rationale of the changes involved [9]. The model consists of three sub-models: family, evolution, and evolution rationale.

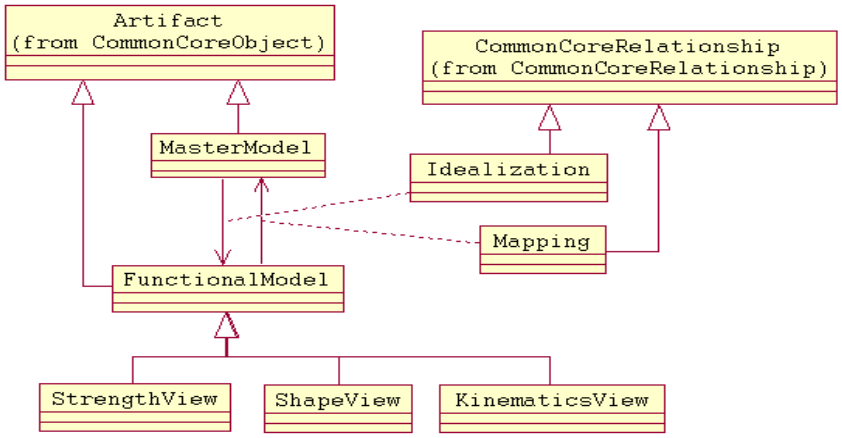

Figure 4: Design - Analysis Integration Model

Product and Component Families. A product is made up of components that usually have their own family definitions. Therefore, product and component families are modeled separately, and configuration relationships established between products and their components. In the model, The family naming space is represented by the classes **Family**, **Series**, and **Version**. The design information on an artifact in the family is represented by the class **PFEM_Artifact**, a subclass of the class **Artifact** defined in the CPM.

Figure 5 shows the class diagram. **Family**, **Series**, and **Version** are subclasses of **FamilyDesignation**. **Family** is the designation for an entire artifact family, which is a collection of **Series,** that may have sub-series. **Version** is a time-sequenced aspect of the family definition; versions form a chain structure. **ProductFamily** and **ComponentFamily**, **ProductSeries** and **ComponentSeries**, and **ProductVersion** and **ComponentVersion** are subclasses of **Family**, **Series**, and **Version**, respectively. **Configuration** is the association class between **ProductVersion** and **ComponentVersion** that defines the configuration between product and component versions.

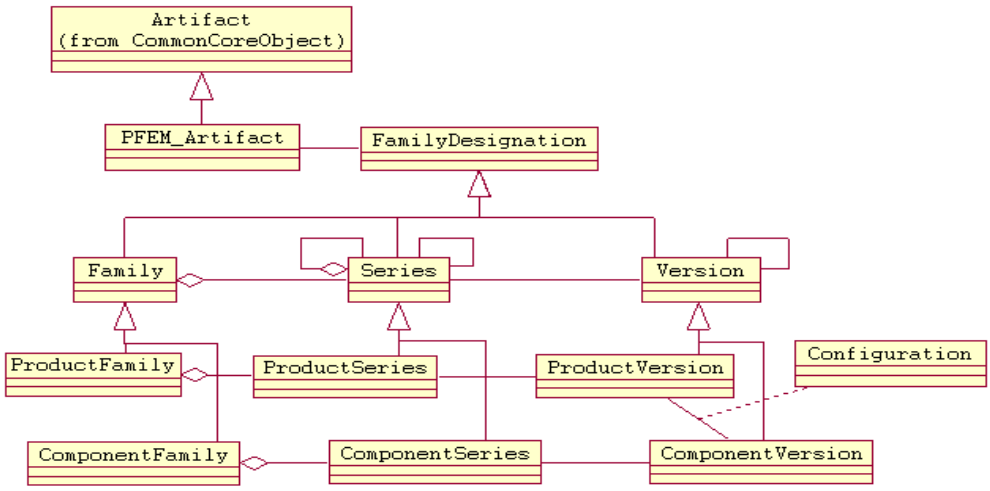

Figure 5: Product and Component Families

Family Evolution. Family Evolution consists of two aspects: Family Derivation and Design Evolution. Family Derivation refers to the set of precedence relationships between derivative series and versions in the evolution of the product line. Design Evolution contains the design information that changed between particular series or versions and their predecessor(s).

Figure 6 shows the class diagram of family evolution. The class **Evolution** is the aggregation of **FamilyDerivation** and **DesignEvolution**. **FamilyDerivation** is specialized into **SeriesDerivation** and **VersionDerivation**. **SeriesDerivation** is the association class between a series and its predecessor series, and **VersionDerivation** is the association class between a version and its predecessor version(s). **DesignEvolution** is the association class between a **PFEM_Artifact** of a series or version and that of its predecessor series or version.

Evolution Rationale. While Family Evolution captures what has changed, Evolution Rationale captures the reasons for the changes. The evolution rationale includes two aspects: Family Derivation Rationale and Design Evolution Rationale. Family Derivation Rationale captures the driving factors for the changes in the product line while Design Evolution Rationale records the reasons for the design changes.

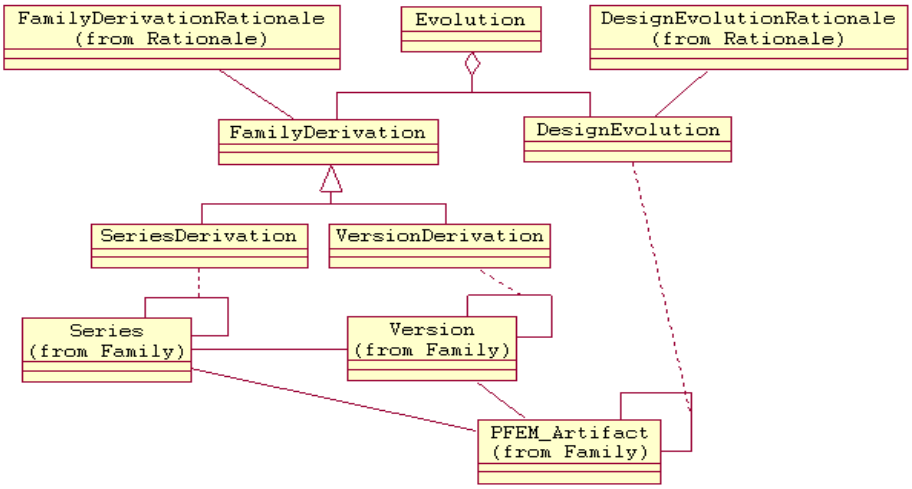

Figure 6: Family Evolution

The class **EvolutionRationale** is defined in the package **Rationale,** shown in Figure 7. The classes **DesignRationale** and **EvolutionRationale** are subclasses of **Rationale**. The class **DesignJustification** defines the justification of the design decision to use the associated artifact, and is the principal contents of the design rationale. The classes **DesignEvolutionRationale** and **FamilyDerivationRationale** are subclasses of **EvolutionRationale**, representing the design evolution rationale and family derivation rationale, respectively. The design evolution driving factors are the justifications of the changes in the design. The derivation driving factors are described by the class **DevelopmentSpecificationEvolution** which represents the evolution of **DevelopmentSpecification**. The classes **Requirement**, **Regulation**, and **Technology** are subclasses of **DevelopmentSpecification**.

4 Further Research Needs

A number of issues have to be investigated before implementation of an interoperability platform based on the proposed product information-modeling framework can begin. First, the framework presented is but a first step towards a complete product modeling architecture supporting the PLM philosophy. A search needs to be made to identify other framework components that need to be modeled and integrated.

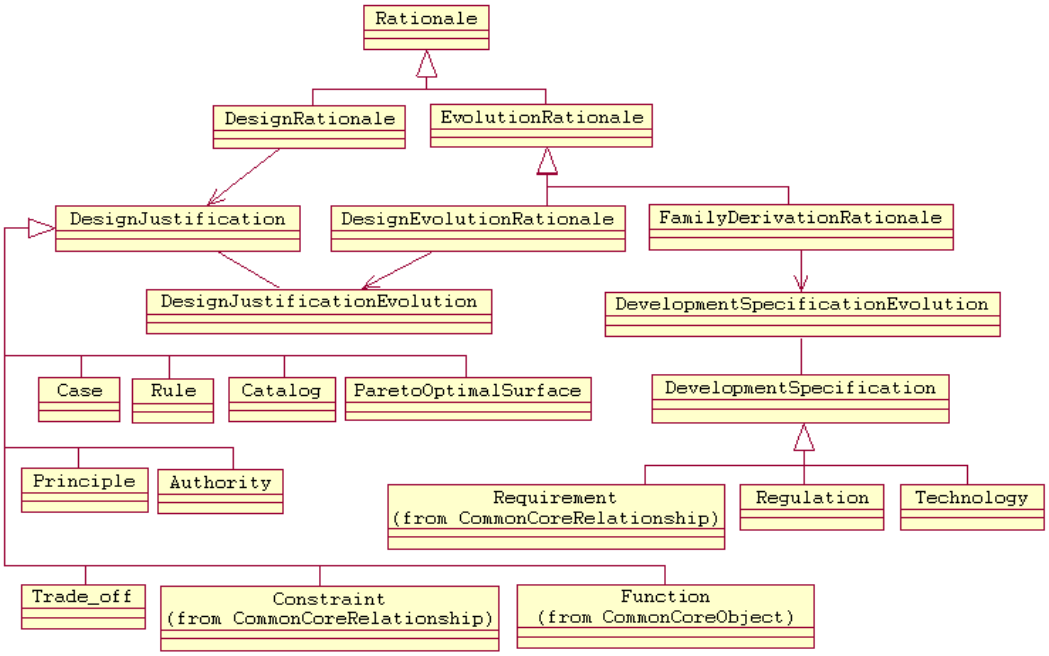

Figure 7: Rationale

Second, a focused search of the PLM literature and current PLM products needs to be made to clarify all product information needs throughout the PLM process to develop a conceptual Application Protocol Interface (API) that can serve all product information to all PLM process components. As part of such a conceptual interface specification, considerable attention needs to be given to the possible interactions between the *product data* served by the framework and *metadata about the products* maintained by the PLM system.

Third, recognizing that product information modeling frameworks of the scope contemplated here will be heterogeneous, rather than single-language, single-vendor homogeneous systems, research is needed to identify, and if necessary develop, information exchange standards that can provide the degree of interoperability that will be necessary.

5 Conclusions

Until quite recently, computer support for product development tended to cover a narrow slice of a product's lifecycle, typically the segment from the product's engineering specification to its physical embodiment. PLM promises to provide support for the product's entire lifecycle, from the first conceptualization to the disposal of its last instance. The volume, diversity, and complexity of information describing the product will increase correspondingly.

This paper makes a proposal for a single product information interoperability framework that can access, store, serve, and reuse all the product information throughout the entire lifecycle. The guiding principles for such a framework are outlined, and four components that constitute the kernel of such a framework are described. Further research is needed to identify and model the other components of the framework, to develop a conceptual Application Program Interface (API) between the PLM system and the framework, and to identify or develop standards for the information interchange. The proposed product information interoperability framework is contemplated to have a broader scope than just being a product information server to PLM systems. Design and manufacturing process components interoperate by

exchanging large volumes of product information, and the proposed product information modeling framework needs to support such "horizontal" information exchanges as readily as the "vertical" exchanges among process components, PLM systems and any intermediary systems, such as PDM and Enterprise Resource Planning (ERP) systems.

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

3. ISO/CD 10303-109*, Product data representation and exchange: Integrated application resource: Kinematic and geometric constraints for assembly models, 2002,* International Organization for Standardization (ISO), Geneva, Switzerland.

4. ISO/DIS 10303-108, Product data representation and exchange: Integrated application resource: Parameterization and constraints for explicit geometric product models, 2003, International Organization for Standardization (ISO), Geneva, Switzerland.

5. ISO10303-105: 1996, Industrial automation systems and integration -- Product data representation and exchange -- Part 105: Integrated application resource: Kinematics, 1996, International Organization for Standardization (ISO), Geneva, Switzerland.

6. ISO 1101:1983, Technical drawings -- Geometrical tolerancing -- Tolerancing of form, orientation, location and run-out -- Generalities, definitions, symbols, indications on drawings, 1983, International Organization for Standardization (ISO), Geneva, Switzerland.

7. 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 (under WERB), 2003.

8. Ho, T. and Tang, C., *Product Variety Management*, Kluwer Academic Publishers, Boston, MA, 1998.

9. Wang, F., Fenves, S. J., Sudarsan, R., and Sriram, R. D., Towards Modeling the Evolution of Product Families, Proceedings of the 2003 ASME Design Engineering Technical Conferences, Chicago, IL, 2003.