**NISTIR 7643**

# Ontological Product Modeling for Collaborative Design

Conrad Bock
XuanFang Zha
Hyo-Won Suh
Jae-Hyun Lee

National Institute of Standards and Technology
U.S. Department of Commerce

**NISTIR 7643**

# Ontological Product Modeling for Collaborative Design

Conrad Bock
XuanFang Zha
Hyo-Won Suh
Jae-Hyun Lee
*Manufacturing Systems Integration Division*
*Manufacturing Engineering Laboratory*

October 2009


U.S. Department of Commerce
*Gary Locke, Secretary*

National Institute of Standards and Technology
*Patrick D. Gallagher, Deputy Director*

# Ontological Product Modeling for Collaborative Design

Conrad Bock
XuanFang Zha
Hyo-Won Suh
Jae-Hyun Lee
October 28, 2009

This paper presents a product modeling language for collaborative design that has the benefits of ontology and expanded capabilities in conventional product modeling. The proposed approach uses ontology to increase flexibility and accuracy in combining, refining, and checking consistency of requirements and designs from multiple, disparate sources, and model-based approaches to develop more powerful, engineering-friendly languages for using ontology. It can capture partial and high-level models in early stages of design, as well as more complete and detailed models in later stages, and supports reliable interpretation of models across the product lifecycle. Examples are given in a proof-of-concept implementation.

## 1 Introduction

Collaborative product development increases efficiency, reliability, and innovation by ensuring the right knowledge is available at the right time [Sriram 2002] [Szykman 2001a,b]. It avoids backtracking and rework due to miscommunication, and delays due to unnecessary dependencies between designers. Integrated and concurrent design enables engineers to interact and reach agreement by sharing product information and knowledge. This reduces the cost of design, increases reliability, opens opportunities for innovation, and provides stronger competition in the international marketplace.

Collaborative product development is hampered by at least two factors:[1]

- Partial product descriptions developed independently by different designers cannot be reliably and consistently refined and combined. Significant effort is needed to create a complete description, because separate contributions rarely fit together and are often overlapping. This prevents teams of designers from contributing to the same product independently, and from building on existing products.

- Engineers and product design tools have only loose alignment on how to interpret different languages and terminologies. Engineers and tools interpret the same product description in different ways. This results in significant cost overruns due to rework when discrepancies are discovered, often after implementation and testing are already underway.

[1] These are issues in shared representation, one of several areas affecting collaborative engineering, including long term knowledge retention, coordination and transaction management, negotiation and conflict mitigation, organizational issues, virtual reality, and decision techniques [Sriram 2006].

Designers distributed geographically and organizationally in global economies worsen the above problems [Sriram 2006]. Multiple firms coordinate to develop and exchange product descriptions. In these situations, designers cannot rely on informal discussion to combine separate contributions to the same product definition, or resolve differences in how product models are interpreted. Product information assets become fragile and return less on investment.

This paper addresses the challenges above by applying ontological and model-based techniques to expanded capabilities in product modeling languages. Ontologies bring an "open world" approach that enables independently developed product models to partially describe the same products, with the precision needed to check consistency when the models are combined, and to ensure uniform interpretation. Model-based techniques provide engineering-friendly languages, freeing engineers from learning the specifics of ontology languages, while still having their benefits. This paper applies these techniques to expanded capabilities in product modeling, such as capturing the environment in which engineered devices are to be used, supporting taxonomies of products and behaviors, and giving interconnection of subassemblies and parts the same capabilities as subassemblies and parts. These improvements enable different or overlapping aspects of product information to be developed separately, or built on each other, then assembled rapidly and flexibly with the aid of automated consistency checking. They achieve this by bringing more relevant information about the product together in a uniform representation. The results are closer to engineering intention and applicable to a wider range of engineering tasks than earlier approaches.

Previous work in product modeling takes either an ontological or model-based approach, but usually not both. Those using only model-based techniques do not support independently developed product models for the same product (open world), or the precision needed to check consistency when the models are combined, while those taking only an ontological approach do not provide engineering-friendly modeling languages. In addition, previous work with ontology in product modeling does not take full advantage of ontology, such as open world semantics, while previous work using model-based techniques does not support many of the needed capabilities, such as interconnection of subassemblies and parts with the same capabilities as subassemblies and parts. The few previous combinations of ontology and model-based techniques support only some of their potential synergies.

The paper covers the most general aspects of product modeling within the above scope, such as product taxonomies, interconnections of parts and subassemblies, and relating behavior to structure. It does not address or restrict:

- Processes by which product models are developed, or representing such processes ("design" as a verb).
- More detailed topics such as kinematics, tolerances, and detailed behavior, geometry, and material models.
- Other stages of the product lifecycle besides design, such as fabrication, maintenance, and disposal.

The approaches of this paper will be applied to the above topics in future work, see Section 6.

Section 2 gives requirements on product models and the proposed language. Section 3 covers previous work on these requirements. Section 4 covers the language with brief introductions to ontology and model-based architecture. Section 5 describes a proof-of-concept implementation of the language. Section 6 gives future work and Section 7 concludes the paper.

# 2 Requirements on Product Models and Languages for Collaboration

Product modeling languages are engineered just as products are. Languages have requirements and alternative ways of satisfying them. Language requirements are naturally intertwined with the kinds of models expected to be constructed by engineers or other stakeholders. This section takes scenario of collaborative design described in Section 1 to determine characteristics of typical product models, and requirements on product modeling languages. Section 2.1 discusses product models as they are constructed by engineers or other stakeholders in a collaborative way, while Section 2.2 addresses requirements on languages used to construct these models. The requirements are addressed by the language proposed in Section 4, where ontological techniques are applied to satisfy the concerns of Section 2.1, and model-based approaches to satisfy those of Section 2.2.

## *2.1 Collaboration-enabled Product Models*

The ultimate goal of collaborative product modeling is to produce a consistent and complete model of a product, drawn from a wide variety of sources. The sources will necessarily be incomplete and potentially conflicting. This section gives several characteristics typical of product models created in this scenario. These are characteristics of product models as they are created by engineers or other stakeholders, rather than requirements on the languages used to create them, see Section 2.2.

Product models developed collaboratively can be intentionally incomplete. They can describe any aspect of a product, no matter how small. For example, a product model for a car might only give limits on how much pollution it may emit, perhaps determined by the manufacturer or a regulating government agency.[2] Models might only describe the materials used in the car. Or they might only give the overall shape of the car for marketing and aerodynamic analysis. All these are product models, they are just incomplete.

Multiple incomplete product models can exist for the same product at the same time. Product development typically has many engineers and stakeholders, each having something to say about the product, but none saying everything. For example, regulators might establish limits on pollution for cars, marketing departments might be concerned with their shape, and engineering departments with the materials to use. Each of these descriptions of the product is a product model in its own right.

[2] Requirements are part of product models, see Section 2.2.

Incomplete product models can be combined into more complete models, assuming they are not contradictory. For example, if one stakeholder requires the car to have a certain shape for marketing purposes, but another requires a different shape to meet mileage requirements, then the models cannot be immediately combined. The ultimate goal is to produce a set of models of the same product that are consistent and cover all aspects of the product.

Product models developed collaboratively can be about the physical product itself (*device*), or the things surrounding the product when it is operated (*environment*), or both. For example, a model of a car might describe the kinds of roads on which it can be driven, perhaps the altitudes and speeds at which it can be driven, and how much pollution it is expected to emit. A product model without these would tell us about the car itself, rather than about the requirements on it or how the car satisfies those requirements.

Product models developed collaboratively can be about *structure*, *behavior*, or both. Structure describes individual physical objects, such as a particular car with an identification number. Structural models describe some aspect of individual physical objects, which some objects might fit and others not. For example, a structural model might describe cars weighing less than 2000 kg. A particular car with an identification number weighing 1500 kg fits this description, while another car weighing 2500 kg does not. Behaviors describe changes in individual physical objects, or lack thereof, occurring during particular time intervals (*behavior occurrences*).[3] For example, a commuting behavior might be describe travel between workplace and home. The occurrence of John going to work in his car on March 3, 2007 between 8:05am and 8:30am Eastern U.S. Time would fit this description if John left from home, otherwise it would not. Behavior occurrences involve individual physical objects, for example, John, his car, and the road on which he is driving that particular day. The same object can be involved in many behavior occurrences. For example, John's car is involved in an occurrence of commuting on many days.

Since product models can be incomplete, they can be about portions of the device, environment, structure, or behavior, in any combination (*total system*). To cover all these cases, total systems are defined in the paper as behavior occurrences involving individual physical devices and objects in the environment when the device is used. Product models describe some aspect of total systems, which might be the interactions of the environment and the device, or how the device should be used. A product model might describe only a portion of the total system behavior, for example, only the objects in the environment, only their behavior, only the structure of the device, or only its behavior. A complete product model combines all these into a consistent description of all the relevant objects and behavior occurrences of the total system.

[3] The term "occurrence" is borrowed from the Process Specification Language (PSL), a constraint language for processes [ISO 2006].

Figure 1 illustrates the above approach. The rectangles are contributions of different engineers and stakeholders to specify the same product, each rectangle being a partially specified product model from one source. The dashed lines and ovals show which portion of the total systems each product model describes. Each model describes some aspect of structure or behavior, or both, of the environment or device, or both. Some of the models only describe the environment (Product Model 1 in the figure), some only the device (Models 3 and 4), and others describe both (Model 2). Models can describe overlapping portions of the total system (Models 1 and 3, 2 and 3), but must not be contradictory. In all cases product models describe some aspect of the total system. The final product model must incorporate these contributions in a consistent way.

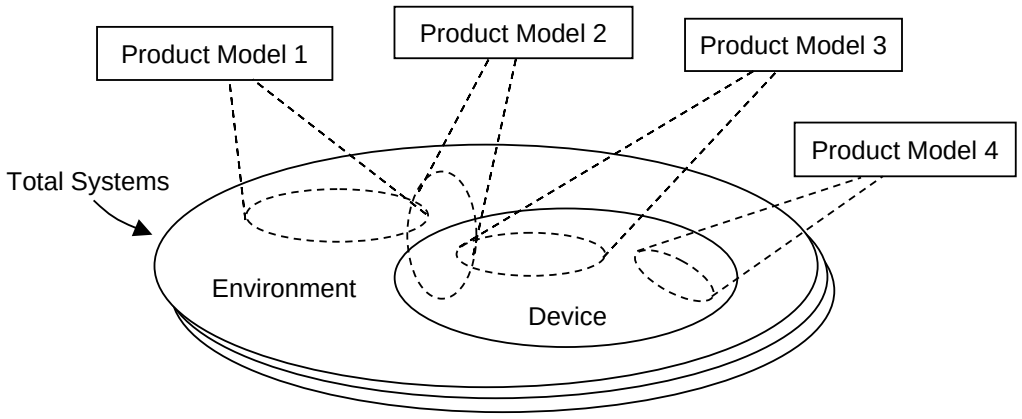


**Figure 1: Multiple Incomplete Product Models of the Same Product**

## *2.2 Requirements for Collaboration-enabled Product Modeling Languages*

This section describes the capabilities of product modeling languages following the principles of Section 1. These capabilities enable designers to express portions of a product model, and combine them with contributions of other designers, see Section 2.1. The capabilities fall into several categories:

- Generalization (taxonomies, refinement).
- Interconnections as components (reuse, inheritance, decomposition, interconnections between interconnections).
- Behavior as relations and interconnections.
- Models of device and environment (designs and requirements, total systems).

Generalization is a technique for combining product models by specifying that everything fitting the description of one product model also fits the description of another. Figure 2 illustrates this with a model of a car generalizing a model of a small car (generalization is notated by the hollow-headed arrow).[4] The models are only concerned with the weight of the car. The general model describes products weighing less than 2000 kg, and in the special model less than 1000 kg. All objects fitting the description of small cars will fit the description of cars in general.

[4] The hollow-headed arrow notation for generalization is borrowed from the Unified Modeling Language (UML) [OMG 2009b], where it has the same meaning as in this paper.

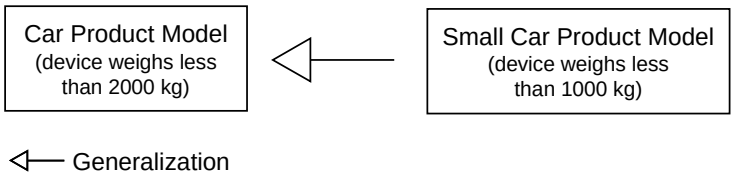


**Figure 2: Product Model Generalization**

Another technique for combining product models is interconnection of other models, according to roles those other models play in the new product, sometimes called "composition" or "assembly." Figure 3 illustrates this in a simplified car model that uses the wheel model multiple times in different ways. The colon notation specifies the reused subassembly to the right of the colon, and the way it is used to the left.[5] Two wheel usages are powered by the engine and two are not, as shown by the lines between the engine and two of the wheels. One of the lines connects the engine and a wheel directly, and the other connects them through subassemblies of the engine and wheel (the connection directly between engine and wheels is composed of other subassemblies, see Figure 5 and following). The resulting product will have four subassemblies specified by the wheel model, two of which will be connected to a subassembly specified by the engine model.[6]

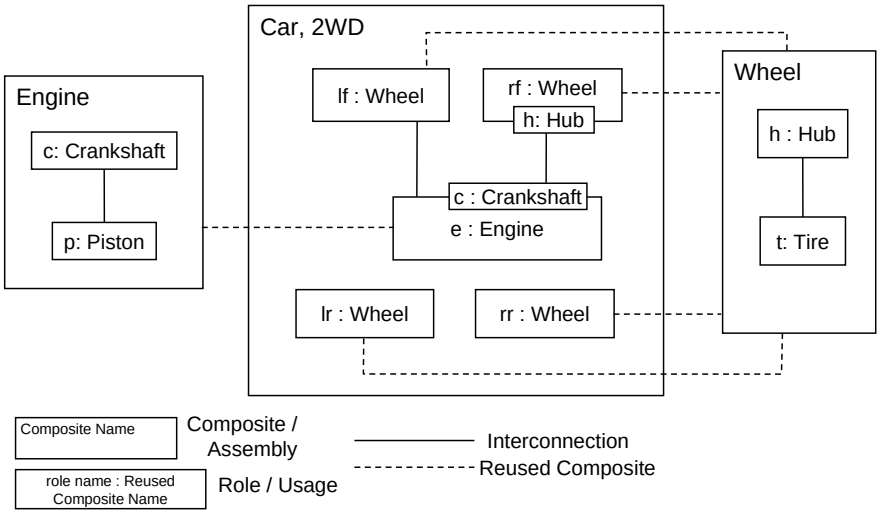


**Figure 3: Interconnected Elements in a Composite**

Generalization applies to refinement of interconnections and subassemblies, as illustrated in Figure 4 (role names omitted for brevity). Vehicles generalize cars and boats. Anything described by the car or boat models must also be described by the vehicle model, by the definition of generalization. Vehicles include an element that exerts force outside the vehicle (the impeller), which generalizes wheels and propellers, as shown in the upper right of Figure 4. Similarly, frames, which are used in vehicles, generalize

[5] This notation is adapted from the UML composite structure diagram [OMG 2009b] [Bock 2004].
[6] The environment can have interconnections also. The examples here are just for devices.

chasses and hulls, used in cars and boats, respectively. Boats add an element for representing the rudder with an additional connection to the hull. The interconnections can also be generalized. For example, a wheel attaches to a chassis differently from a propeller to a hull, but these are generalized by the common characteristic of limiting the movement between the frame and impeller.

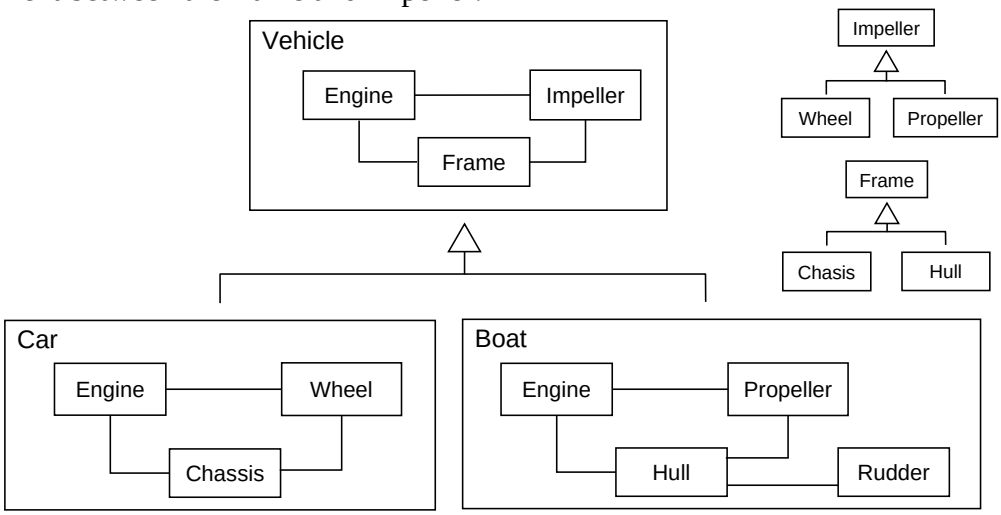


**Figure 4: Generalization of Composites**

Interconnections within an assembly or composed product model can be specified as a composition of other interconnected elements, as illustrated in Figure 5 . The connection between engine and wheel decomposes into a separately specified set of interconnected elements that includes a clutch and gearbox for manual transmissions. Things described by the car model will have individual clutches and gearboxes linked between the engine and wheels. The same connection decomposition might be reused many times, for example in chemical plant might have the same piping patterns between tanks used many times. This enables product designers to reuse complex interconnections defined by others.

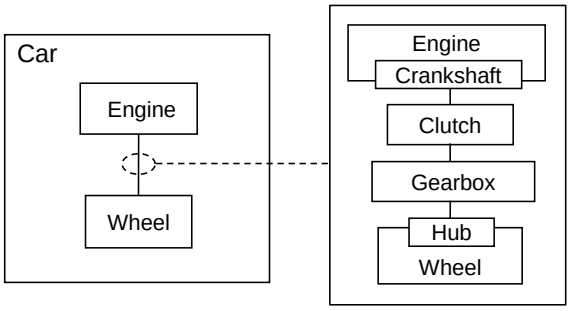


**Figure 5: Decomposition of Interconnections**

Alternative product model refinements using generalization and interconnection composition are illustrated in Figure 6. The car model generalizes two others that decompose the connection between engine and wheel in different ways. One decomposes it with a manual transmission, the other with an automatic transmission. The things described by the specialized product models are also described by the car model, by the definition of generalization, including any characteristics of the interconnection of engines and wheels, such as power delivery requirements, see example in Figure 8.

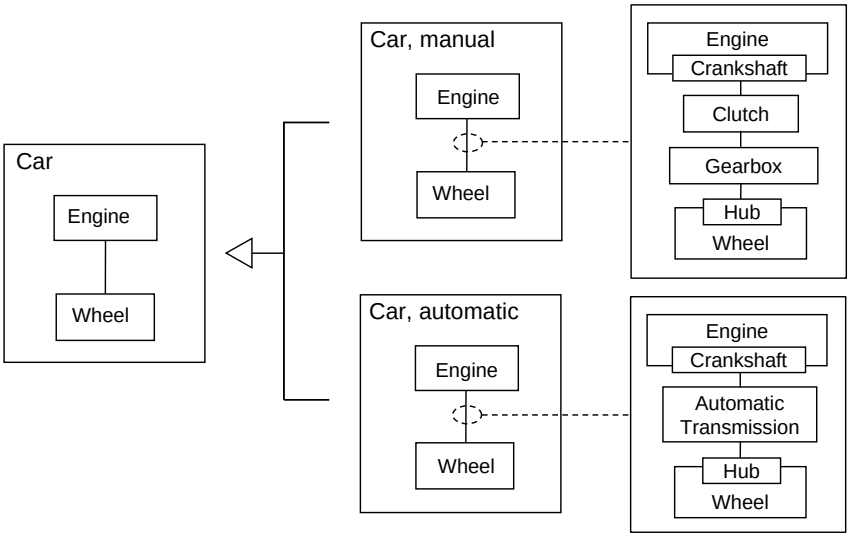


**Figure 6: Product Generalization and Interconnection Decomposition**

Interconnections can be interconnected in a product model, as illustrated in Figure 7. Two interconnections that decompose into piping elements are themselves interconnected. The individual physical devices described by the assembly will have two individual pipes between which there is heat transfer also described by the product model. This can be combined with generalization, for example, to specify fluid is transferred between the units, without specifying it is achieved with piping. Then the thermal connection applies to specializations of the assembly that transfer fluid in different ways.

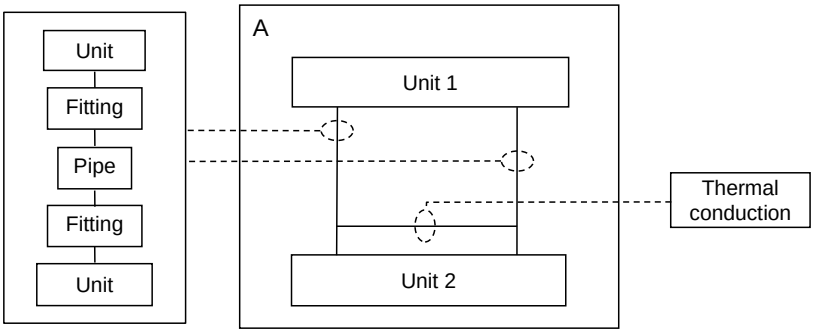


**Figure 7: Interconnections of Interconnections**

Behaviors can be taken as relations between the objects involved in them, enabling behaviors to interconnect elements in a composite or assembly.[7] For example, a behavior might specify that the objects involved in them have no relative movement. Figure 8 shows a model for this on the lower left, used by an interconnection in an assembly just above it. The result is an assembly that constrains a plate and bracket to remain in a fixed

[7] Some behaviors might appear to involve only one object, but even these are relative to other objects. For example, a rotating or moving object is only doing so in relation to other objects. The expansion of metal by heating is relative the environment in which heating occurs, which might prevent expansion.

relative position. The model generalizes two others in which the interconnection is decomposed in ways that are consistent with the general behavioral constraint of no relative movement. One specialization uses a bolt and nut to maintain the relative position of the bracket and plate, the other uses a rivet. These decompositions satisfy the general behavioral constraint, because the products described by the specialized models are also described by the general one, according to the definition of generalization. This example is elaborated in the proof-of-concept implementation in Section 5.

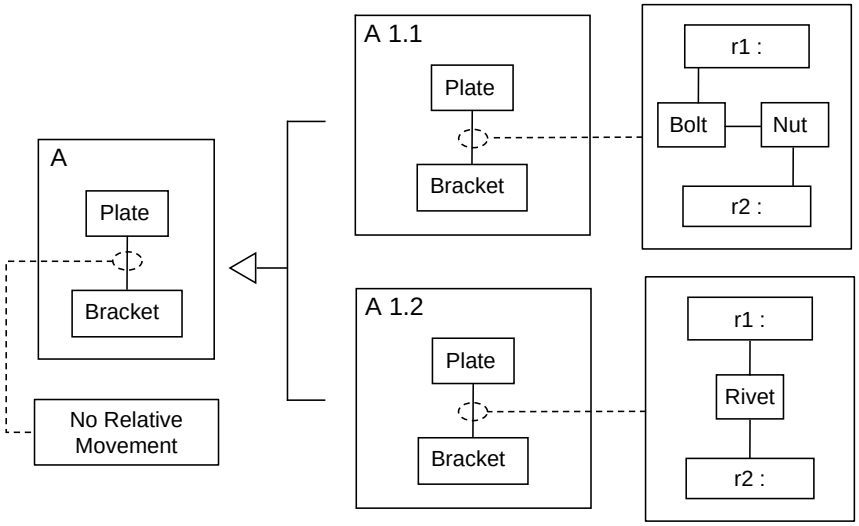


**Figure 8: Behavior Relations**

At least two kinds of product models are useful for modeling languages to identify. Since models can be about any aspect of a device or the environment in which is operates (and the structure or behavior of either one, see total system in Section 2.1), it is helpful if the language distinguishes at least these kinds of models:

1) *Designs* are product models that describe only engineered devices, which might include their intended behavior.

2) *Requirements* are product models that describe the environment in which the device is to be used [Zave 1997], including:
   a) The objects in the environment of the device when it is used, and the effect of the device on them.
   b) The behavior of the environment in the presence of the device, including the interactions of the environment and the device, for example, how the device should be used.

The definitions of design and requirement above are for exposition in this paper, and can be changed or adapted as needed. In particular, some communities use the term "requirement" to include descriptions of devices developed at early stages. For example, a marketing requirement might constrain the weight of lawn mowers, or the color. In the terminology of this paper, such constraints are part of design of the lawn mower. The requirement is to be able to lift the device or that it is pleasing to look at. Requirements in this paper are "operational" or "usage" requirements. The terminology can be adapted or changed as needed when the language is applied in particular communities.

# 3 Previous Work

Previous work in product modeling takes either a model-based or ontological approach, but usually not both, as needed to satisfy the requirements in Section 2. Those using only model-based techniques do not support independently developed product models for the same product (open world), or the precision needed to check consistency when the models are combined, as required in Section 2.1, while those taking only an ontological approach do not provide engineering-friendly modeling languages, as described in Section 2.2. In addition, previous work with ontology in product modeling does not take full advantage of it as described in Section 2.1, such as open world semantics, while previous work using model-based techniques does not support many of the capabilities described in Section 2.2, such as giving interconnection of subassemblies and parts the same capabilities as subassemblies and parts. The few previous combinations of ontology and model-based techniques support only some of the synergies described in Section 2. This section reviews these two areas of previous work in product modeling, with model-based approaches in Section 3.1, and ontologies in Section 3.2.

## *3.1 Model-based Product Information*

Model-based approaches to product information were used long before ontological techniques. They provide engineering-oriented languages with some of the capabilities in Section 2.2, but specify their meaning informally, and lack the benefits of ontology described in Section 2.1. These languages include ISO 10303 (STEP, STandard for the Exchange of Product model data [Owen 1993]), the Unified Modeling Language (UML) [OMG 2009b] and its extensions in the Systems Modeling Language (SysML) [OMG 2008a], the U.S. National Institute of Standards and Technology (NIST) Core Product Model (CPM) [Fenves 2008] and its extension in the Open Assembly Model (OAM) [Rachuri 2006], and Methodology and tools Oriented to Knowledge-based engineering Applications (MOKA) [Stokes 2001]. Table 1 summarizes their support for the capabilities described in Section 2.2, see details given in the rest of this section. The last column (OPML, Ontological Product Modeling Language) is the language proposed in this paper.

| | | STEP | UML 2 / SysML | CPM 2 / OAM 2 | MOKA | OPML |
|---|---|---|---|---|---|---|
| **Generalization / refinement** | | X+ | S | X | X+ | S |
| **Composition / assembly** | | | | | | |
| | **Interconnection of elements** | S- | S | S- | X | S |
| | **Multiple usages of the same kind** | X+ | S | v1: X<br>v2 : S- | X | S |
| | **Relation / connector decomposition** | X+ | UML 2 : X<br>SysML : S- | X+ | X | S |
| | **Interconnections of interconnections** | X | UML 2 : X<br>SysML : S- | X | X | S |
| **Behaviors as relations / connectors** | | X | X | X | X | S |
| **Total system ( device / environment )** | | X | X+ | X | X | S |

S : Full support
S- : Support with exceptions
X+ : Does not support, with exceptions
X : Does not support

**Table 1: Support for Capabilities of Section 2.2**

STEP is a comprehensive standard, covering a wide range of manufacturing product data, some crossing subdomains (generic and integrated resources) and some specific to subdomains, such as electrical amid mechanical products (application protocols). STEP gives a standard terminology for product management system vendors [Gu 1995].[8] STEP models have assembly support within some of the application protocols, but limited assembly representations across subdomains. For example, ISO 10303-44 [ISO 2000] specifies the assembly structure of a product and manages its configuration during its lifecycle. One of its primary features is that it provides a mechanism to generate and maintain various kinds of product data structures, such as bills of material and parts lists, using the same primitive entities, such as components. ISO 10303-109 [ISO 2004] defines kinematic and geometric constraints for mechanical assemblies, including assembly feature relationships and constraint schemas. ISO 10303-239 [ISO 2003b] provides capabilities for product life cycle support, including product lifecycle activity management, product description, operational feedback, and support solution and environment. The Organization for the Advancement of Structured Information Standards Product Life Cycle Support Technical Committee is currently developing Data Exchange Specifications using ISO 10303-239. This includes product breakdown for support, fault states, task specification, maintenance plan, operational feedback, product as individual, requirements, core models, and infrastructure maintenance.

STEP currently does not adequately address the capabilities of Section 2.2, in some cases only in particular application protocols. ISO 10303-109 supports hierarchical relations and interconnections among components via part properties such as assembly features, but it does not cover assembly generalization and relation decomposition for configuration management of assemblies and components, nor behaviors as relations and

[8] STEP uses EXPRESS as a metalanguage, see footnote 17 in Section 4.3.

interconnections. Extensions or STEP-compatible models have been proposed to address some of the missing capabilities in STEP assembly models [Liu 1992] [Zha 2002], for example, introducing entities such as assembly, subassembly, part, and connector for integrated assembly design and process planning, and also to combine EXPRESS [ISO 2003a] and Extensible Markup Language [W3C 2006] [Zha 2006]. In addition, ISO working group TC184/SC4/WG12 is updating STEP's assembly representations. However, these proposals are not currently adopted into STEP.

UML is a standardized, general-purpose, wide lifecycle modeling language defined at the Object Management Group (OMG), under their Model-driven Architecture [OMG 2009c]. It is designed for visualizing and interchanging models of many kinds of systems and the entities in their environment. It is used for business processes, organizational structures, and hardware. It covers both structural and dynamic models, including composite objects and multiple behavior models. UML provides two extension mechanisms, profiles and metamodel specialization. SysML extends UML to a modeling language for manufactured systems engineering, which is a concerned with the accurate capture of customer requirements and reliable translation of these to high-level designs for hardware, software, and organizations. SysML is defined as a UML 2 profile by the OMG, in cooperation with the International Council on Systems Engineering (INCOSE). UML and SysML fully support generalization, composition and assemblies, except only SysML supports relation decomposition and connectors between connectors. Neither supports behavior as relations or interconnections. SysML and UML do support total system modeling very well, because of the limited nature of requirements in SysML and use cases in UML.

The NIST research effort on foundations for next generation CAD and product development systems suggests a core representation for product information, CPM, and extensions to this model, such as OAM, and Design-Analysis Integration [Fenves 2003]. Heterogeneous Material Model [Biswas 2007], Mechatronic Device Model [Xu 2005], the Product Family Evolution Model [Wang 2003], and the Embedded System Model [Zha 2005a]. CPM is intended to unify and integrate product and assembly information, providing a base-level, knowledge-oriented model that is open and non-proprietary, extensible, independent of product development process, and capturing engineering context most commonly shared in product development activities. It focuses on artifact representation including function, form (including geometry and material), behavior, physical and functional decompositions, and relationships among these concepts. OAM extends CPM to include assembly, tolerance and propagation, kinematics, and engineering analysis at the system level. OAM is integrated with the EXPRESS/XML schema based assembly model [Zha 2002] to completely capture the detailed geometric information using XML schema [Zha 2006]. Thus, the STEP-based integrated product information model comprises not only geometry but also function, behavior, form feature and product structure information.

CPM and OAM currently do not adequately address the capabilities of Section 2.2. OAM supports interconnections between parts, but not involving elements of the same kind (except in version 2), and only ports for assembly. It also does not support

generalization, decomposition of relations and connectors between connectors, or behavior relations and connectors. CPM does not clearly distinguish behavior of the device and environment, though some work exists in this area [Zha 2005b] [Xu 2005].

MOKA is perhaps the most powerful and widely known methodology for product design modeling and development of knowledge-based engineering applications, including process knowledge. It is intended for routine design tasks, rather than conceptual design that includes requirements. The MOKA product model supports five views or perspectives on the underlying product model. Structure is the hierarchical decomposition of a product into parts, assemblies, and features (structures). It can be physical or logical. Function is the functional decomposition of the product and principles of solution. Behavior includes a state model of the various states of a product and of the transition from one state to another. Technology includes materials and manufacturing process information. User-defined technological information includes alternate representations of the physical structure. MOKA provides a kernel for various kinds of applications, using base classes as the key integration elements. MOKA is based on UML 1.0 class modeling, and applies UML stereotypes.

MOKA shares the same theme as the approach proposed in this paper, but many of the potential capabilities are not drawn out in the documentation and articles. For example, MOKA has the potential to support generalization and relation decomposition, but this is largely unexplored in the documentation. In many cases, it is unclear how the semantics of the stereotype applies to the MOKA concepts. For example, state and transition in MOKA behaviors are stereotypes of UML Class, but it is not explained what they classify (what the instances of states and transitions are). MOKA does not consider the environment of devices (total system modeling), or requirements modeling generally. This makes it difficult to apply MOKA during conceptual design. MOKA supports interconnection of elements in a composition, but not behavior as relations, connectors between connectors, multiple usages of the same kind.

## *3.2 Product Ontologies*

Ontology has been recently applied to product modeling, usually to improve the precision of modeling languages, such as those in Section 3.1 [Metzger 1996]. Some of the applications follow the typical definition of ontology as being about real world things, or things intended for the real world, rather than linguistic constructions such as modeling languages. This approach gives more precise meaning to product models by interpreting them as physical things, satisfying some of the requirements in Section 2.1, but does not tie ontologies to modeling languages for ease of use in the engineering community, as required in Section 2.2. Other work applies ontology languages to the syntax of existing or new modeling languages, which improves the specification of how those languages appear to the engineer, but does not give them ontological meaning in terms of individual, physical objects or occurrences of behavior. The few efforts that combine ontology and modeling languages do not take advantage of the open world semantics of ontology for collaboration, and provide only a portion of the expressiveness needed in modeling languages.

Some of the previous work applying ontology to product modeling classifies physical objects or occurrences of behavior, for example, built products with serial numbers, or the operation of a particular pump on a particular day. Some of the work identifies general concepts for individual physical objects, such as relations between parts and wholes, and integrity within physical objects, and congruence, with higher-level concepts are built on the fundamental ones, such as convexity and perpendicularity [Guarino 1997]. Other work extends these general physical concepts to include dynamics, then relates dynamics to mathematical equations for use in simulation [Borst 1997]. Some work standardizes taxonomies of physical objects in particular manufacturing domains, such as pumps and heat exchangers, specialized further into classes defined by industry associations, all specialized from generic classes supporting assembly of physical things and how they are involved in occurrences of behavior [Leal 2005]. Some research explores how these product taxonomies can be used in knowledge management and configuration systems [Chan 2007] [Yang 2008], while another effort gives a methodology for defining product family ontologies [Nanda 2006]. All these lines of work can describe individual physical things, such as a particular car with an identification number, but do not provide a modeling language for engineers. For example, these approaches can classify physical things, such as pumps, but an engineer cannot specify which models are about the environment of a product as it is operated (requirements), and which are about the product itself (designs).

Other previous work applies ontology to elements of existing or new product modeling languages, rather than classifying physical objects or occurrences of behaviors.[9] Some research gives taxonomies of functions, such as providing material, preventing a side-effect, or driving a process [Kitamura 2002]. These classify domain-specific functions that appear in a product model, such as heating and generating torque, rather than occurrences of these functions that happen at particular times. Other research gives a taxonomy of requirements and their relations, such as explicit and derived requirements, and requirement decomposition [Lin 1996]. Other research uses ontology to formalize an assembly language that was defined in UML, and integrate it into design tools [Kim 2006]. Some research formalizes the syntax of CPM using ontology languages [Patil 2005] [Fiorentini 2008]. These lines of work precisely specify how to use a modeling language properly, but do not have the benefits of ontologies of physical objects and occurrences, such as checking consistency of requirements and designs.

A few previous efforts combine ontology and modeling languages, using the same or related architectural techniques as in this paper, and applying them in product lifecycles [Ackermann 2004] [Lee 2008]. These efforts do not draw out the implications of ontology for collaborative design (partial and refined product models combined and checked for consistency), and their modeling languages do not integrate interconnected elements with generalization, decomposable relations, and behaviors.

[9] This work uses ontology to describe linguistic entities rather than real things (syntax rather than semantics), see metalanguages in footnote 21 in Section 4.3. Sometimes these linguistic entities are treated as real things, for example, a model of a car is given a particular weight, such as 1000 kg, but this is a shorthand for specifying that actual cars built according to the model will have a particular weight.

# 4 An Ontological Product Modeling Language

This section describes a product modeling language for the capabilities described in Section 2, addressing those of Section 2.1 with ontological techniques, and those of Section 2.2 with model-based approaches. Section 4.1 briefly introduces ontology in general. Section 4.2 shows how ontology applies to product modeling. Section 4.3 introduces a technique for making ontology more accessible to engineers, by integrating modeling languages with ontology. Section 4.4 explains the benefits of this technique to product modeling. Section 4.5 covers models of generic concepts of the language proposed in this paper, such as relations, while Section 4.6 covers extensions for engineering concepts. Together these last two sections describe the Ontological Product Model Language (OPML), which uses the technique of Section 4.3 to combine ontology and modeling languages and satisfy the requirements of Section 2.

## *4.1 Introduction to Ontology*

Ontology focuses on descriptions of real or intended things, especially partial descriptions that can be combined and checked for consistency separately from the many real or intended things being described.[10] In this paper, engineers and stakeholders provide descriptions of intended products that are ultimately manifested in individual physical things and their behavior. Ontology enables product engineers and stakeholders to independently develop partial descriptions of the same product and check consistency when the descriptions are combined.[11]

Ontology is usually formalized with set theory, where members of the sets are actual or intended things, and rules for membership (called *classes*) capture expert knowledge and specifications. In product modeling, the members of the sets might be individual physical products or their behavior occurrences, and membership rules are engineering and stakeholder specifications about the product. Classes only characterize the members of sets, they do not identify the sets or individual members directly. For example, a class might require the members of a set to be all and only cars weighing less than 2000 kg. The set described by this class will have individual cars in it, each with particular identification number, but the class will not identify any of the cars directly. Sets are defined just by listing their members (extension). Classes are defined by rules determining members of sets (intension).[12] The members of the set are said to *conform* to the class, be *classified* by the class, be *consistent* with the class, or informally, be "described" by the class. Classes are sometimes informally called "categories" or "types."

[10] See footnote 21 in Section 4.3 regarding ontologies of linguistic rather than real world entities.

[11] This is analogous to a merging style of change management in document systems, where multiple developers can edit copies of the same document, and merge them together with conflict checking. It is an alternative to the check-in/out style of document management, where only one developer can edit a document at a time.

[12] Membership rules can determine that individuals are members (*sufficient* conditions) or are not members (*necessary* conditions). For example, a membership rule for small cars is that they are cars, which is a necessary condition. The rule would determine that a truck is not a small car, but could not determine whether a sports car is a small car, because not all cars are small cars.

Classes are very expressive because they are only rules for set membership, rather than actual members of sets. Classes can describe:

- one, some, or all aspects of things (open world).
- things from the past, present, future.
- intended things that are actually built or imagined things that are never realized.
- physically possible or impossible things.
- things with very little in common, or things that are very similar.

This expressiveness arises from separating classes from the sets of things they describe. Membership rules exist independently of sets, which may be empty or contain only things that do not or cannot exist in reality, such as perpetual motion machines. The goal of the engineering process is for these sets to contain real and useful things, but classes provide a way to capture engineering knowledge at stages before and after the members of the sets become real or useful.

Ontological reasoning examines classes to determine results of operations on the corresponding sets, without using the members of the sets directly.[13,14] For example, Figure 9 on the upper right shows a class for cars weighing less than 2000 kg, and on the upper left a class for cars weighing more than 2000 kg. Example members of the sets described by these classes are shown at the bottom. Ontological reasoning applied to these classes reveals whether they can be consistently combined into a single class. In this example, reasoning shows the sets corresponding to the combination of classes will never have any members in common, because an individual car cannot weigh more and less than 2000 kg at the same time. The classes being combined are inconsistent. Reasoning determines this using only the class descriptions (weighing more and less than 2000 kg), not by intersecting the sets conforming to the classes. This enables ontological reasoning to operate without enumerating all possible members of the sets described by classes.

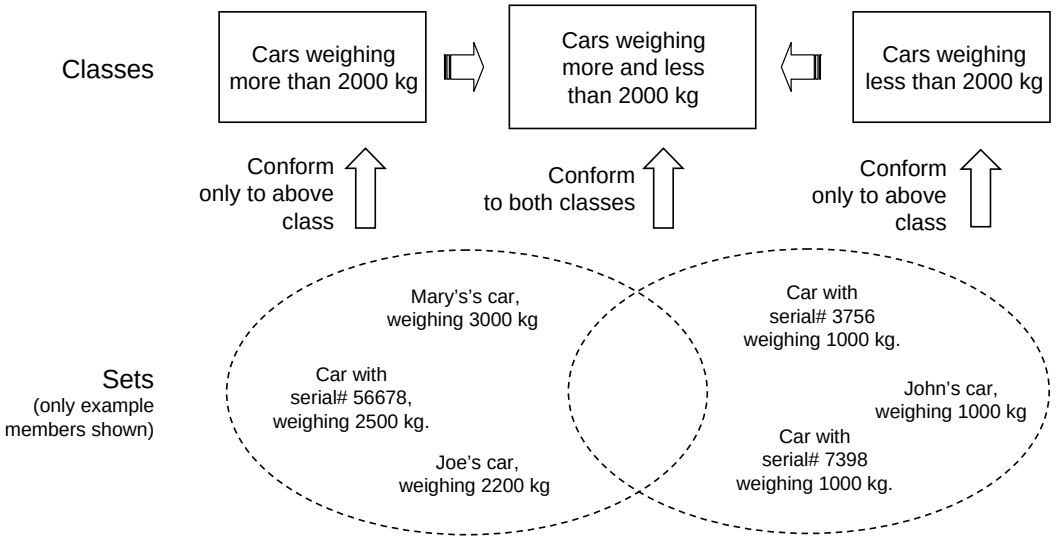


**Figure 9: Classes and Sets**

[13] This distinguishes ontological reasoning from rule systems, which require instances to perform inference, due to closed world semantics.

[14] Ontological reasoning can also determine whether an individual is in the sets described by classes (instance checking), but this is not as applicable to engineering design, because the products do not usually exist before the models.

## *4.2 Ontological Product Modeling*

Product models can be represented as ontological classes because product models describe:

- existing or potentially existing things, and occurrences of behavior of those things. For example, a product model for a car describes actual cars with identification numbers, or occurrences of behavior, such as a car being driven on a particular day by a particular person, or simulated versions of these. Product models do not describe documents or other engineering data recording requirements and designs for cars. These documents and data are the product models, and product models do not describe themselves.[15]
- some aspects of the same product (open world). For example, two product models can describe the same car, where one specifies the elevations at which is can be operated, while another specifies its weight limit. Partial product models can be combined into a more complete ones, until a model is reached that is complete enough for manufacturing.

Sets described by product models (as classes) must have members that are the same kind of thing, to enable their descriptions to be combined into one model and tested for consistency. Otherwise, a model might describe members that are ruled out by other models, even if the models are consistent in every other way. For example, if the members of sets described by product models are taken to be either physical objects or behavior occurrences, the intersections might be missing some members just because they are objects, or just because they are behavior occurrences, even if the objects are involved in the occurrences in a consistent way. Product models that describe sets of the same kind of things can support logical operations on models developed independently in a collaborative setting, regardless of their source.

One option is to take product models as classes of physical objects, for example, cars with vehicle identification numbers, or simulated versions of these. Requirements and designs could describe individual cars and be tested for consistency. However, cars behave in many different ways at different times, in part because of variations in how they are operated and in what environment. The description of just the objects themselves cannot capture the variety of these situations.

Another option, used in this paper, is to take product models as classes of behavior occurrences in which objects are involved, for example, the driving of a car during a particular timeframe, as shown at the bottom of Figure 10. The product model on the upper left limits operation of the car to below 5000 m. This describes the set of behavior occurrences encircled in the lower left ellipse, each an operation of an individual car. Physical objects are involved in these occurrences, because there must be something that is behaving. A product model might describe only objects involved in the occurrences, as much product data does, is illustrated on the upper right of Figure 10. This model limits the weight of the car involved in the occurrence to less than 2000 kg. This is just an

[15] This avoids well-known paradoxes caused by of sets containing themselves.

intentionally incomplete model that does not cover the dynamic aspects of occurrences. It describes the occurrences encircled in the lower right ellipse. The two original models describe occurrences involving the engineered device (design) and the environment in which it is operating (requirement), respectively (total system behavior occurrences, see Section 2.1), while the combined model covers both.

Since these two product models describe sets of the same kinds of things (behavior occurrences) logical operations can apply to them, as shown in Figure 10. Classes descriptions are combined to cover sets of occurrences described by both models at the same time. In this example, these are occurrences of using a car weighing under 2000 kg at an elevation under 5000 m. The set described by this combined model is illustrated in the intersection of the two ellipses (only examples are shown for brevity). The occurrences not in this set involve a car under the weight limit, but used at an unintended elevation, as shown by the occurrences on the far lower right, or involve a car over the weight limit, but used at a proper elevation, as shown on the far lower left.

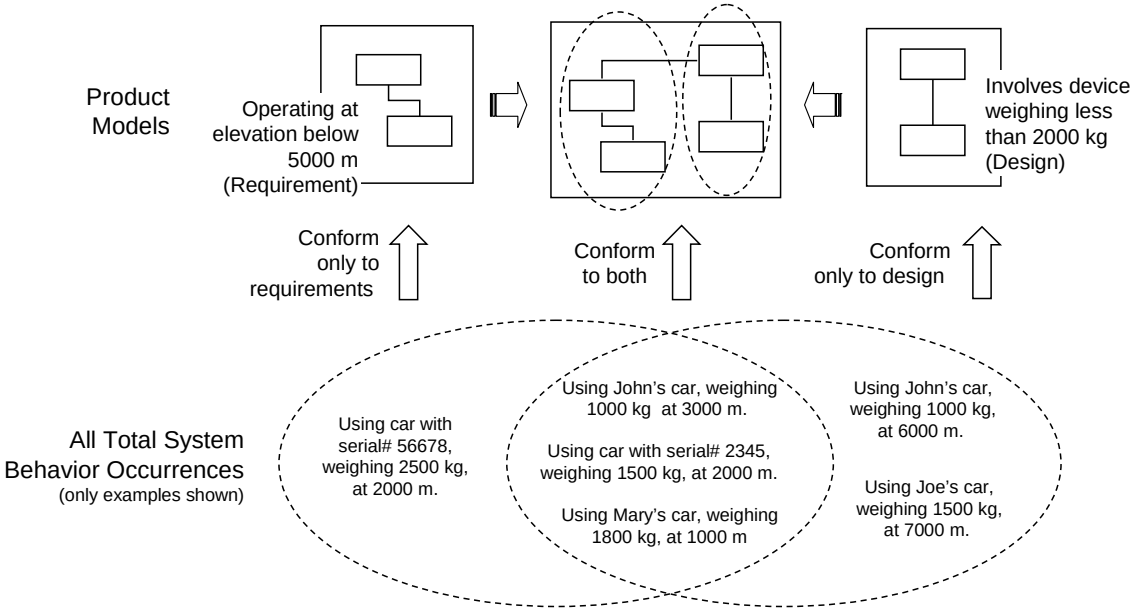


**Figure 10: Product Models Describing Behavior Occurrences**

## *4.3 Languages for Ontological Product Modeling*

A barrier to adoption of ontological product modeling is ontology languages are not specific to engineering. Ontology languages take time to learn and relate to engineering applications. For example, the Web Ontology Language (OWL) uses the terms "class", "property", "restriction", "subclass", "domain", "range", all with set-theoretic meanings not familiar to engineers [W3C 2004a]. This is illustrated under the left box in Figure 11, with one ontology language element, Class, shown at the at the *M2 level*, which is where

languages are defined.[16,17] Product models are shown at the *M1 level*, such as a model of cars. This is where engineers and other stakeholders define products. Product models describe members of sets at the *M0 level*, such as using a particular car with a particular identification number. These are the total systems that product models are ultimately about, see Section 2.1.[18]

Using only ontology languages at M2, as under the left box of Figure 11, the engineer must know to use CLASS at M2 and other ontological concepts when defining product models. This enables product models to generalize each other, for example CAR being more general than SMALL CAR,[19] and to classify actual cars at M0 (notated with a dashed arrow),[20] and to be checked for consistency when combined. However, these benefits would only be available to engineers trained in ontology. Also engineering information about the models cannot be recorded, such as the engineers responsible for them, or which models are requirements and which designs, because CLASS cannot define these properties. CLASS is part of the ontology language, rather than an engineering language. One attempt to address this might be to introduce PRODUCT at M1, as a generalization of all product models. This would give some guidance to engineers, but does not support information about product models, as in the examples above.

Engineering modeling languages typically do not have the benefits of ontology described earlier. For example, under the middle box of Figure 11, the M2 level has an engineering term, PRODUCT MODEL, which is used to specify particular product models at M1, such as CAR and SMALL CAR. This enables engineering information about the models to be recorded, such as the engineers responsible for them, or which models are requirements and which designs, because PRODUCT MODEL is part of an engineering language. However, the M1 product models are not ontological classes, because they are not specified with CLASS at M2.[21] This prevents CAR from generalizing SMALL CAR, from classifying actual cars at M0, and from being checked for consistency when combined.

The approach of this paper combines the ones above to provide the capabilities of ontology and the ease of use of modeling languages, as illustrated under the right box in

[16] The levels in Figure 11 are numbered by the Object Management Group's (OMG) Model-Driven Architecture (MDA) [OMG 2009a,c], inherited from the Electronic Industries Alliance's Computer-Aided Software Engineering Data Interchange Format [Flatscher 2002].

[17] Modeling languages are specified in *metalanguages*, which appear at an *M3 level*. These languages are not shown for brevity, but are usually a small subset of domain-independent M2 modeling languages, such as UML class diagrams. See footnote 21.

[18] From a software viewpoint, the M0 level is divided into physical objects or behavior occurrences, and measurements made on those recorded in a manufacturing information system. This paper treats these as the same for brevity, and to reflect the engineering design viewpoint that is not concerned with manufacturing information.

[19] See footnote 4 in Section 2.2 about notation.

[20] The conformance link from CAR and SMALL CAR to CLASS is omitted for brevity.

[21] Ontology languages can be used as metalanguages (M3 level, see footnote 17), describing language elements at M2 such as PRODUCT MODEL, rather than models such as CAR at M1. This ensures precise specification of modeling languages, as in this paper and some previous work, but does not provide the benefits of ontology to modelers working at M1, as in the techniques on the left and right of Figure 11, because ontology at M3 only enables classes to be specified at M2.

Figure 11. It includes CLASS as a generalization of PRODUCT MODEL in the modeling language at M2.[22] Engineers can use a term they know to define product models at M1, and record engineering information on product models, while still having the benefits of ontology. Product models can generalize each other, classify actual cars at M0, be checked for consistency when combined, and carry information such as which engineers are responsible for them, or which product models are requirements and which are designs.

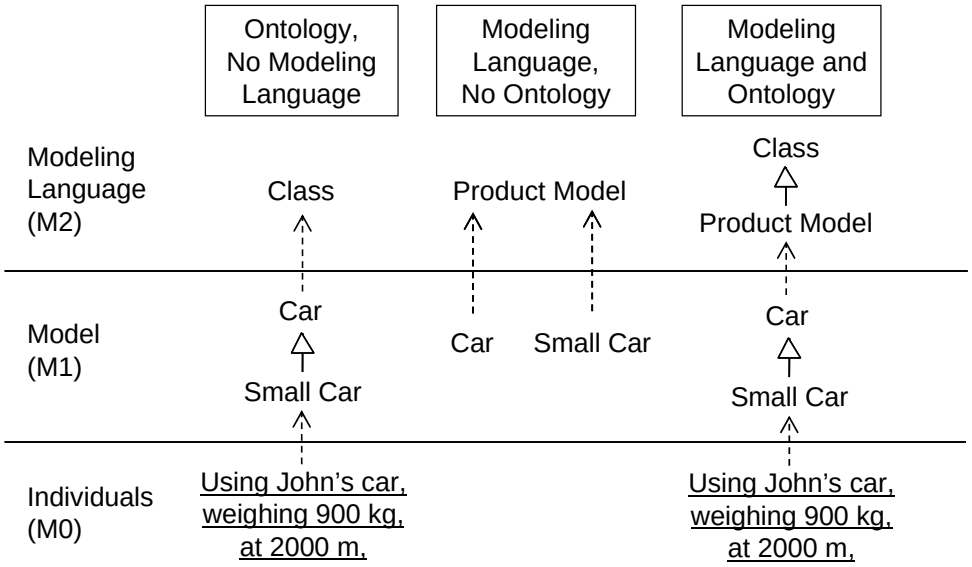


**Figure 11: Ontology and Product Modeling Languages**

The modeling technique illustrated on the right of Figure 11 does not dictate or restrict design processes. In particular, the hollow-headed arrow notates generalization, rather than a sequence of steps in a design process. For example, the model for small cars might be developed before the model of cars in general. The design process might also proceed the other way, from cars to small cars. Or the individual M0 elements might exist before the model is developed, as occurs when the documentation is lost for long-lived products such as ships, and must be regenerated from real world artifacts, or in prototype-based design processes.

The language modeling techniques in Figure 11 cover the terminology (*abstract syntax*) of the language, but not concrete syntax, such as punctuation and graphical shapes. This enables multiple concrete syntaxes to be described a single language specification. Developing concrete syntax involves many issues of visual ergonomics and conventions that are not addressed by abstract syntax. In the rest of this paper, "language" will refer to abstract syntax, as in the M2 level of Figure 11, and the implication of M1 models for M0 individuals (*semantics*).

[22] Generalization can be used at M2 when ontology languages are used as metalanguages, see footnote 21.

## *4.4 Benefits of Ontological Product Modeling Languages*

Product modeling languages using the technique of Section 4.3 support partial, refinable, and combinable product models. For example, generalization can refine requirements with alternative designs, as illustrated in Figure 12. The most general model at M1 only requires that water is moved somehow, without specifying the device achieving this. For example, the MOVING WATER model might specify a minimum rate at water should be moved. The two specialized models introduce devices, a pump and Archimedes screw, respectively.[23] They must meet the requirement of moving water at a minimum rate, because all of their conforming total systems also conform to the model of moving water, by the definition of generalization, see Section 2.2.

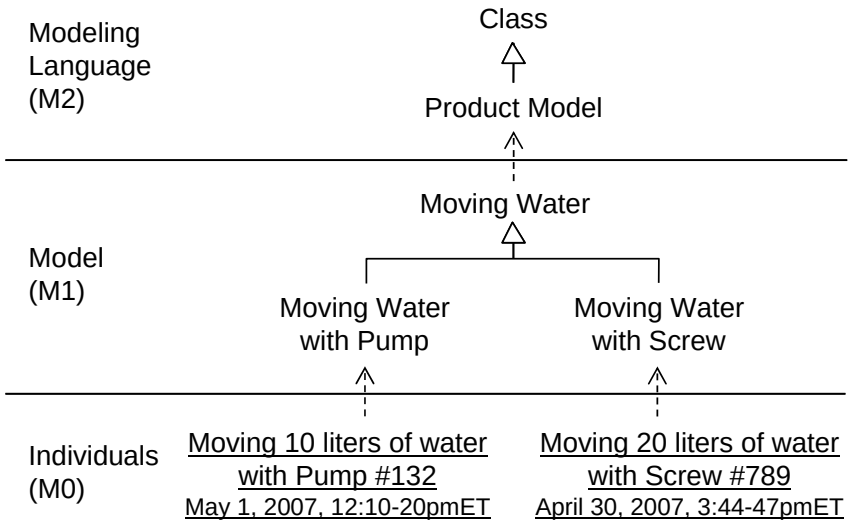


**Figure 12: Alternative Designs for the Same Requirement**

Requirements and designs are kinds of product models, as shown at the M2 level in Figure 13. Requirements are models that describe the environment of devices as they are operated, including the proper use and expected effect of the device, while designs are models that constrain the devices being operated, see Section 2.2. The taxonomy of requirements and designs at M1 describes narrower sets of individuals in the more specialized classes. For example, vehicles are described as weighing less than 10000 kg, and small vehicles as less than 2000 kg. The individual total system at the lower right conforms to the small vehicle design, because the operated car weighs 1500 kg, while the total system at the lower left does not, though it meets the safety requirement for small vehicles. The same model can describe both environment and device, whereupon it is both a requirement and a design, and the same M0 total system can conform to multiple M1 models, which might be requirements and designs, see Figure 10 in Section 4.2.

The technique of Section 4.3 facilitates cross-checking of requirements and designs at multiple levels of abstraction, to identify potential errors early in the design cycle and

[23] Conformance links from the specialized M1 models to PRODUCT MODEL are omitted for brevity.

increase the accuracy and completeness of testing built products against requirements. Figure 13 illustrates this with safety requirements generalized on the left, designs on the right, and generalizations between them in the middle. Descriptions are added in the specialized classes. For example, safe, small, dry land vehicles have a traction requirement that is not present for all safe, small vehicles. Analysis can be applied to check that specialized models imply the more general ones when requirements and designs are combined [Shooter 2000].[24] For example, analysis can predict whether the traction requirement for safe, small, dry land vehicle designs will produce the required stopping distance for safe small vehicles. This determines whether it is possible for M0 elements to conform to the design for safe, small, dry land vehicles, because these must conform to the more general requirements also, by the definition of generalization. Tests can be developed to verify generalizations in practice. For example, tests on the produced vehicle might verify traction is within the specified limits, and that this results in the specified stopping distance.

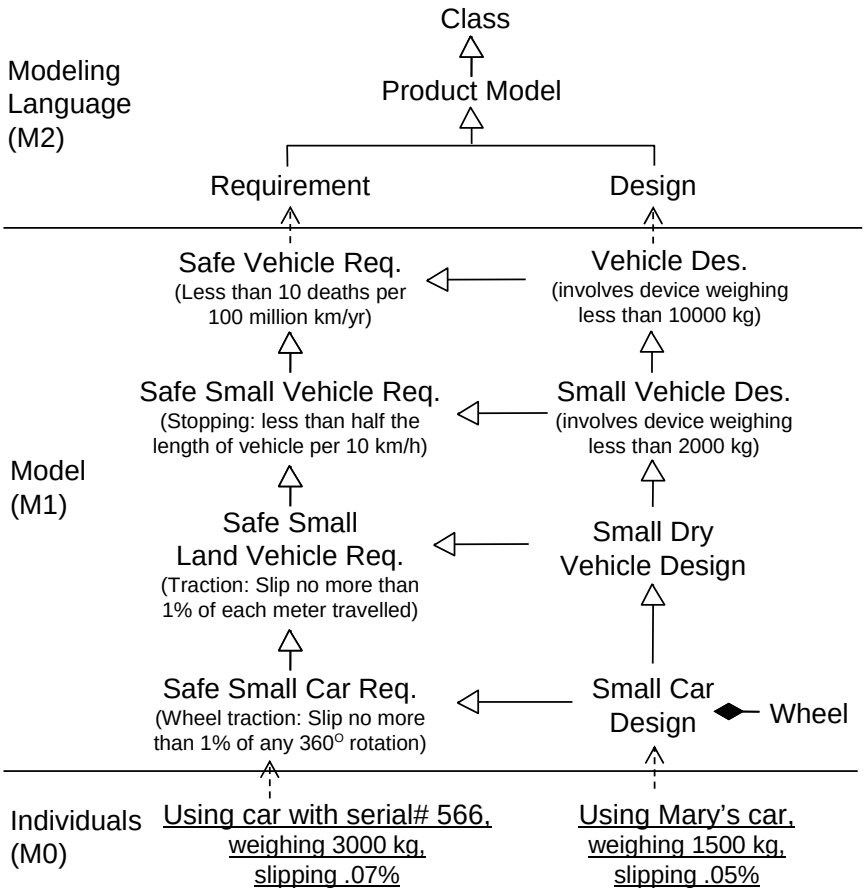


**Figure 13: Requirements and Designs**

[24] This assumes requirements only describe the expected effects under proper operation of the device. This prevents contradictions caused by total systems conforming to designs and not the operational requirements. These total systems are not of concern to requirements, which only capture that the device behaves properly when operated properly.

## *4.5 Modeling Language for Relations, Interconnections, and Behavior*

This section gives abstract syntax for a language covering basic concepts such as relations, behaviors, compositions, and combinations of these (see end of Section 4.3 about abstract syntax).[25] Conformance between M0 and M1 levels is specified to provide semantics. The base modeling language of this section is used as the foundation for engineering terminology in Section 4.6.

Relations are introduced into the modeling language (M2) as a specialization of CLASS. This enables M1 relations to be generalized at M1, and to have conforming *links* at M0. For example, in Figure 14, LINKAGE has conforming links at M0 that are categorized into fixed and moving.[26] Relations are always between other things, modeled at M2 with RELATES to identify the kinds of things being related. There are at least two of these, as indicated by the minimum multiplicity of two. For example, LINKAGE is defined at M1 as relating one LINKED THING to another, which is introduced just to categorize M0 individuals that can be linked.

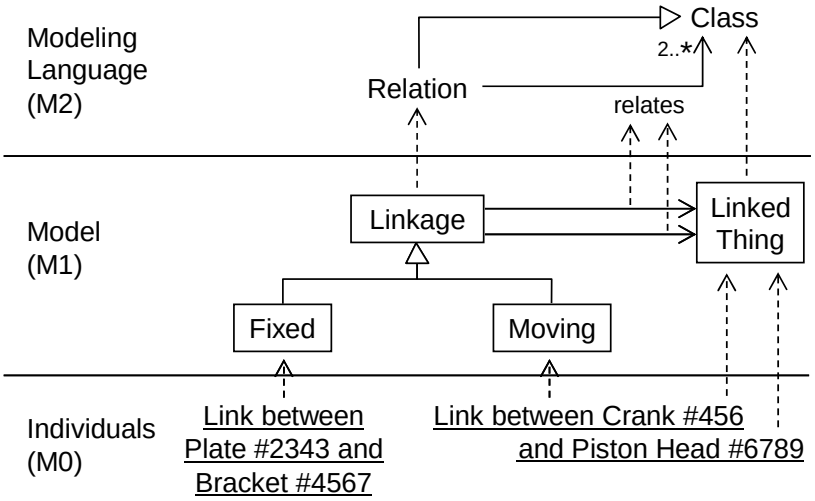


**Figure 14: Relations**

Relations by themselves are insufficient for capturing the composition of interconnected elements, for example, the interconnections between engines and wheels in cars in Figure 3 in Section 2.2 [Bock 2004]. A relation between the ENGINE and WHEEL classes is not restricted to the context in which it is used, for example, it would allow an engine in one car to power the wheels in another, or a propeller in a boat, a spare wheel in the same car. Defining specialized classes for interconnections, for example car and boat engines, and for powered and spare wheels, is cumbersome (introducing classes for every "role" played by engines and wheels), and still allows the engine in one car to power the wheels in another. The only way to capture designer intent fully is to define classes for the individual engine in every individual car, for example, classes for John's car engine, Mary's car wheel, to separate the engines and wheels in John's and Mary's cars. This of

[25] This syntax is extended from the full level of [Bock 2009] and has the same semantics.
[26] Rectangles are used at M1 in the rest of the paper to distinguish it from the other levels.

course is unacceptably burdensome for the designer, in part because it is not usually known at design time what the individual M0 elements are. These problems suggest the modeling language should be extended to support interconnection between elements of composition and assembly.

A solution to the above problems is illustrated in Figure 15, notated with an adapted form of UML class diagrams,[27] with some of the conformance links shown in Figure 16. The primary aspects of the solution are:

1. Identify connected elements by relations between a composite and the elements inside it. For example, identify the engine and wheels in each individual car using the relations ENGINEINCAR and POWEREDWHEELINCAR, between cars and their engines and powered wheels, respectively. This ensures the POWERS relation is applied only with each individual car, not between cars, see next item.
2. Treat interconnections like any other element of the composite, with the additional capability of connecting the relations in the previous item (between a composite and its elements). For example, the powers relation between a car's engine and wheels is treated like an element of the car, just like the engines and wheels. It is given a relation to the composite, just like the relations for engines and wheels in the first item above. In Figure 15 and Figure 16 this is the POWERSINCAR relation between CAR and POWERS conforming to CONNECTOR, a new element at M2. The POWERSINCAR relation identifies M0 powers links between the engines and wheels in each individual car. This enables interconnections to be generalized, like any other element of the composite, as in Figure 4 in Section 2.2, as well as connected by other connectors, as in Figure 7.

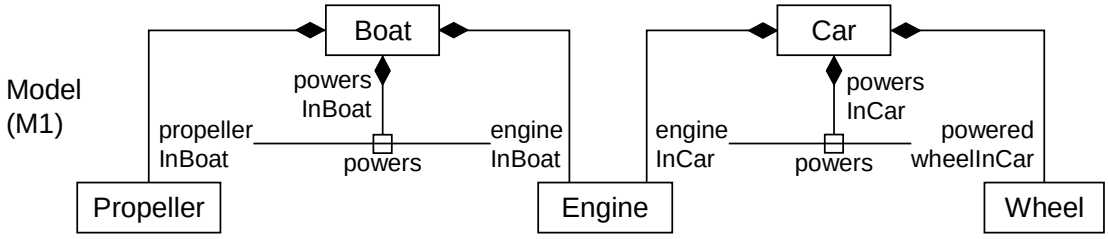


**Figure 15: Interconnected Elements (M1)**

[27] UML composite structure diagrams are a compact and scalable notation for this, see Figure 3 in Section 2.2 [OMG 2009b] [Bock 2004]. The large rectangle is the class CAR, while the smaller, nested rectangles are relations from the class CAR to elements inside it. The names of the relations are shown to the left of the colon. The interconnections are shown as lines between the rectangles inside the class.

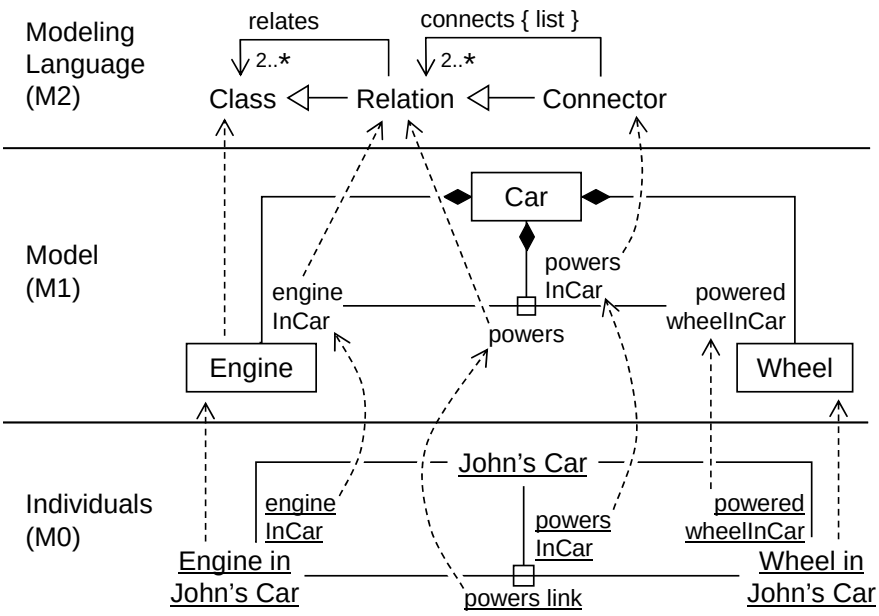


**Figure 16: Interconnected Elements (Three Levels)**

Interconnections can be specified between subassemblies of subassemblies, for example, connecting hubs within wheels of a car to the crankshaft in its engine, as illustrated in Figure 3 in Section 2.2. It is not sufficient to model this with a connector between the hubs and the crankshaft, because not all the hubs are in powered wheels, and not all engines are in cars. The modeling language (M2) specifies the ends of connectors with a series of relations from the whole, as shown in Figure 16 with the "{list}" annotation on CONNECTS. In this example, one end of the connector refers to a list of POWEREDWHEELINCAR and HUBINWHEEL to power only the hubs in powered wheels, while the other end refers to a list of ENGINEINCAR and CRANKSHAFTINENGINE to limit the connection to crankshafts in car engines, rather than boat engines.

Relations between a composite and its elements can include elements that are "outside" the composite. For example, a car model might have a relation for the owner. Relations like these can tie device models to models of their environments. For example, a car model can have a relation for the roads on which it is supposed to be operated, and the operator. Elements of the environment and device can be interconnected as needed for the intended use, for example, the wheels of the car are on the road, and the hands of the operator are on the steering wheel. An alternative is to model the total system of car and its environment. In this approach, the car model does not refer directly to road and driver, so all its relations between cars and their elements are to things contained in cars.

Relations can be composed of interconnected elements, like classes in general. For example, the powers relation between engines and wheels might be composed of a clutch and gearbox that are related to each other and to the engines and wheels. The model for this example is similar to Figure 16, except the composite element is a relation, as illustrated in Figure 17. The POWERS relation has four elements, each identified by its own relation, for example, the E relation to ENGINE, and the GB relation to GEARBOX (impellers generalize wheels and propellers, see Figure 4 in Section 2.2). The relations

identifying the linked objects do not have a diamond because they are not contained in POWERS. The four elements are interconnected by other relations, such as EPOWERSC and CPOWERSGB, as specified by the connectors (relations from POWERS to the connectors are omitted for brevity). Individual links at M0 conform to POWERS by having interlinked elements, such as the engine and clutch in John's car linked in conformance to EPOWERSC. Individual M0 links for composite relations can be established by M1 connectors. For example, the powers link in Figure 17 is established by the connector at M1 in Figure 16.

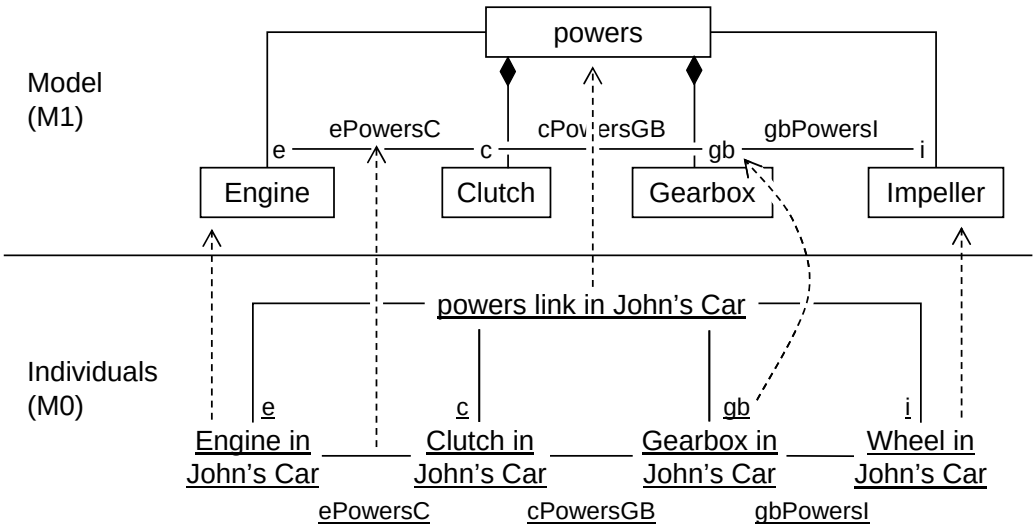


**Figure 17: Relation Decomposition**

Relations can generalize others that have alternative decompositions, like classes in general. For example, the POWERS relation of Figure 17 might actually be just for manual transmissions, and another relation might have a decomposition for automatic transmissions, as illustrated in Figure 6 in Section 2.2. The alternatives can be generalized by an abstract version of the POWERS relations that does not have any connectors between the engine and wheels, as shown in Figure 18 (M2 and M0 omitted for brevity). The relations between ABSTRACTPOWERS and its elements, E and I, are available in the specializations AUTOMATICPOWERS and MANUALPOWERS (was POWERS in Figure 17), because M0 links conforming to AUTOMATICPOWERS or MANUALPOWERS also conform to ABSTRACTPOWERS, by the definition of generalization. The specialized relations each have their own way of interconnecting the engine and wheels.

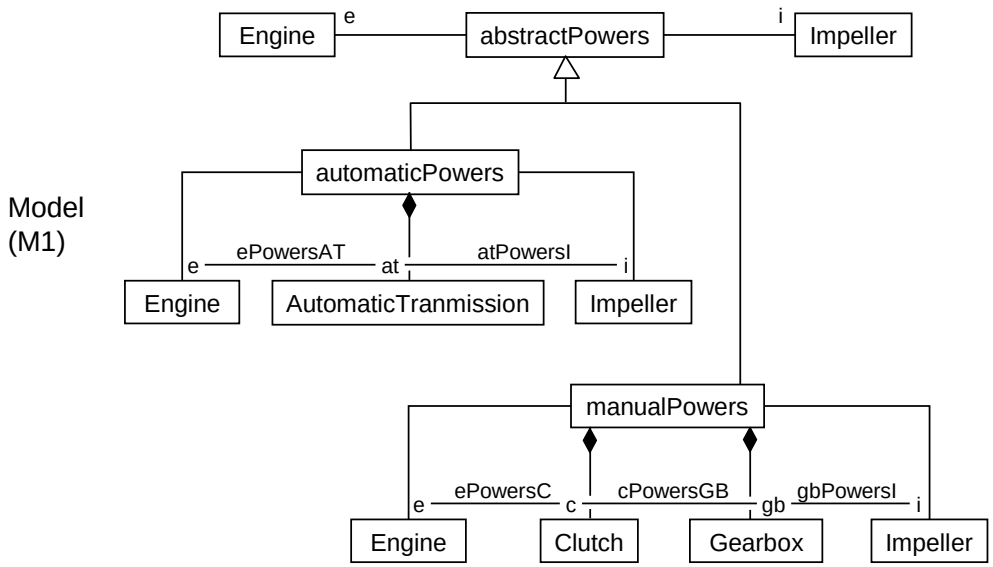


**Figure 18: Relation Generalization with Alternative Decompositions**

Specialized relations can be used to specialize interconnections, for example, the connector between engines and wheels in Figure 6 in Section 2.2. The model for this is shown in Figure 19. Cars in general use the abstract powers relation, while specialized cars use the manual and automatic powers relations. The POWERSINCAR connector is also generalized in cars from the specialized connectors in manual and automatic cars, AUTOMATICPOWERSINCAR and MANUALPOWERSINCAR, as shown in the upper right of Figure 19. This means M0 links conforming to AUTOMATICPOWERSINCAR and MANUALPOWERSINCAR also conform to POWERSINCAR. For example, if John's car is a manual, then the link between the engine and wheels in it will conform to MANUALPOWERS and ABSTRACTPOWERS, and the link will be identified by both MANUALPOWERSINCAR and ABSTRACTPOWERSINCAR.

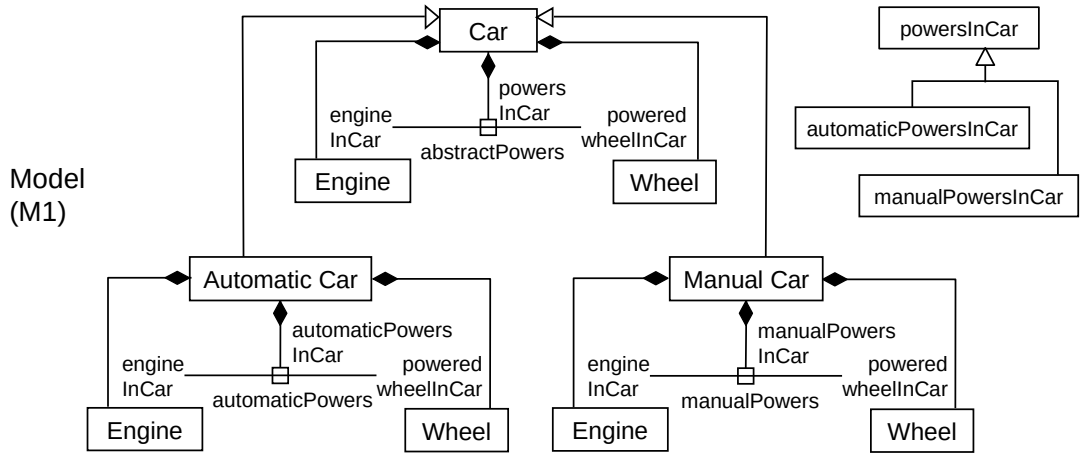


**Figure 19: Generalization of Interconnected Elements using Relation Generalization**

Interconnections can be interconnected, for example, the connector between the pipes in Figure 7 in Section 2.2. The model for this is shown in Figure 20. The assembly A has four elements, two of which are connectors using the PLUMBING relation. Since connectors are relations between a composite and its elements, they can be also connected (in this example by a connector using THERMAL relation). Portions of this model can generalized, for example, to specify fluid is transferred between the units, without specifying it is achieved with piping. The interconnections would apply to specialized assemblies, which specialize the connectors between the units for the various ways to achieve fluid transfer. The thermal connection would inherit to the specialized assemblies also, like the rest of the elements, and apply to the specialized kinds of fluid transfer between the units.

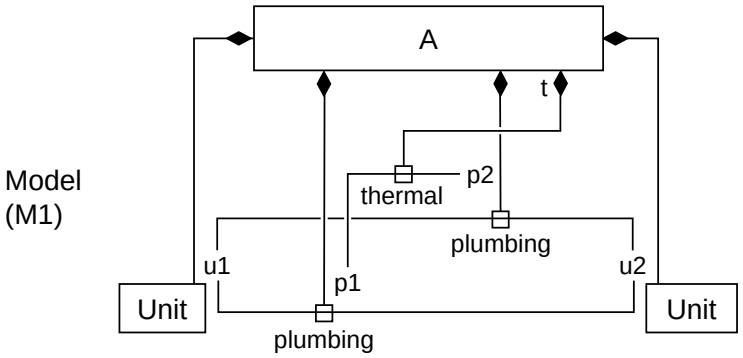


**Figure 20: Interconnection of Interconnections**

Behaviors are introduced into the modeling language (M2) as a specialization of CLASS, as shown in Figure 21. This enables M1 behavior models to be generalized at M1, and to have conforming behavior occurrences at M0. For example, in Figure 21 the rotation behavior has conforming occurrences at M0 that are categorized into fast and slow rotations. Behaviors always have at least one element that is behaving, modeled at M2 with INVOLVES, to identify the kinds of things involved in the behavior. For example, the M1 ROTATION element is defined involves ROTATING THING, which is introduced just to categorize those M0 individuals that can rotate.

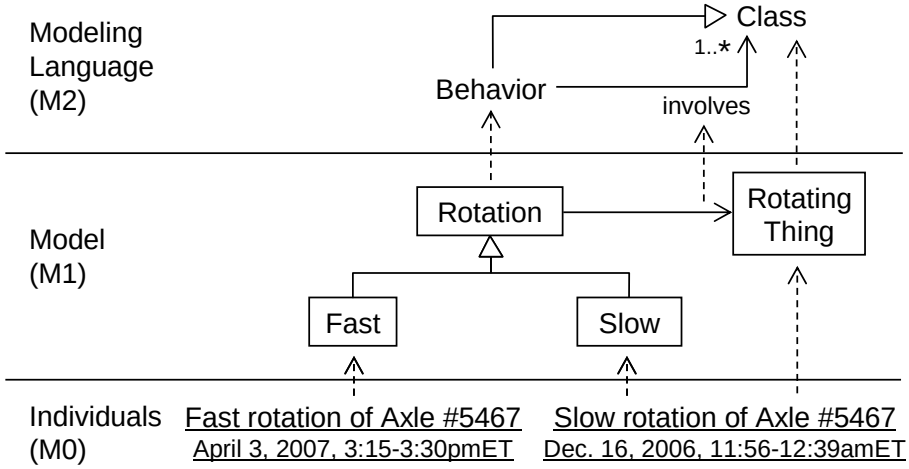


**Figure 21: Behaviors**

Behaviors can generalize others involving additional interconnected elements in the behavior, like classes in general. For example, a behavior specifying that two things do not move in relation to each other generalizes other behaviors that achieve this in different ways, as shown Figure 22. The NO RELATIVE MOVEMENT behavior involves two things, identified by two relations, F1 and F2 (without diamonds, because the fixed things are not contained by the behavior). The behavior generalizes two others, one involving a rivet and the other a bolt and nut. The specialized behaviors have interconnections between the fixed things and the other elements involved in achieving that. The specialized behaviors must prevent relative movement, because their M0 occurrences are also occurrences NO RELATIVE MOVEMENT, by the definition of generalization.

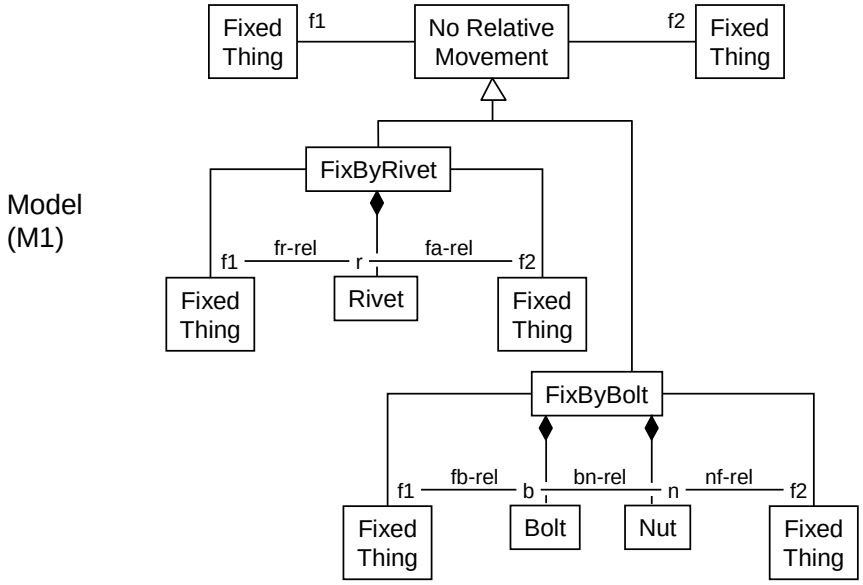


**Figure 22: Behavior Generalization**

Behaviors can be modeled as relating the things involved in them. For example, a relative rotation behavior can be modeled as a relation between the things rotating relative to each other. This enables behaviors to be used as connectors in assemblies to capture functional requirements (shown next). This is introduced into the modeling language (M2) with RELATION generalizing BEHAVIOR, and RELATES generalizing INVOLVES, as shown in Figure 23. [28,29] This means things involved in behaviors are also in relations with each other due to the behaviors. For example, the two axles at M0 in Figure 23 are linked in relative rotation, in conformance with RELATIVE ROTATION.

[28] In UML this is called "property subsetting," notated with a textual annotation rather than graphically as in Figure 23.

[29] Behaviors have at least one thing involved, whereas relations have at least two related things. This is resolved by introducing another class for behavior relations that is generalized by both behaviors and relations, see Section 5.1.

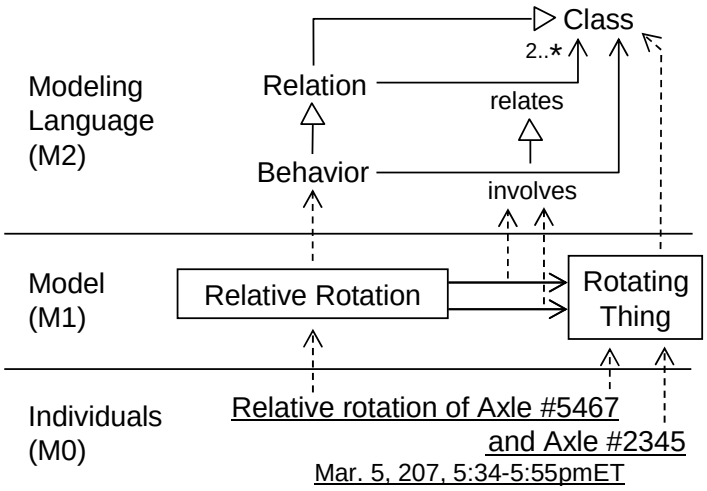


**Figure 23: Behavior Relations**

Behaviors as relations can be used to interconnect elements, as illustrated on the left of Figure 8 in Section 2.2. Connectors of this kind can be specialized in taxonomies, as illustrated on the right of that figure. The model for this is shown in Figure 24. The assembly A specifies that a plate and bracket do not move relative to each other, using the behavior defined in Figure 22. This generalizes two other assemblies, A1.1 and A1.2, which achieve the behavior in the different ways. The connectors are also generalized, as shown in the upper right.

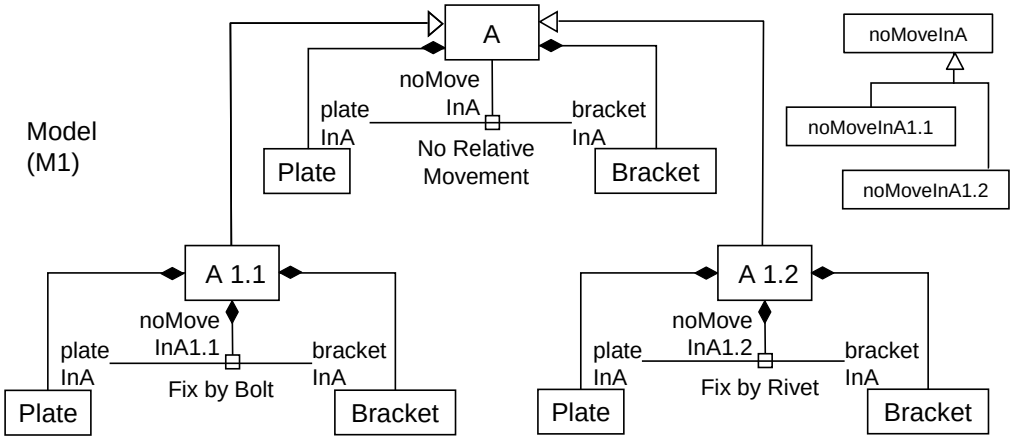


**Figure 24: Generalization of Interconnected Elements Using Behavior Generalization**

## *4.6 Engineering Modeling Language*

This section builds on the models of Section 4.5 to capture common engineering concepts, such as product models, requirements, artifacts, and form. Most of these are special cases of the modeling elements of Section 4.5, while some are top-level categories. They provide a framework for development of more specialized engineering concepts.

Product models as defined in Section 2.1 describe behaviors and objects involved in them, where the:

- involved objects are the devices being specified and objects in their intended environment.
- behaviors are those of the above objects, including their interaction.

Product models might place many constraints on the above or very few, or might place many constraints on some aspects and few on others. For example, a model might only describe the structure of devices and environmental objects, but not the behavior. The model still describes total systems, but only the structural aspects of them. Other models might focus only on the dynamic aspects, rather than structure.

The modeling language of Section 4.5 is extended for product models as shown in Figure 25. Models identify the device among the involved objects with the SPECIFIES relation. Other things involved in the model are either inside the device or in the environment of its use. Artifacts are devices specified by product models, as indicated by the minimum multiplicity of one on the product model end of the relation. Things not specified by product models are not artifacts. For example, the moon is not an artifact, because its orbiting behavior is not specified by a product model. Product models generalize requirements and designs. Designs describe the artifact, requirements describe the environment in which they are used, see Section 2.2. M1 product models must fall into one of these categories, and fall into both if they describe both environment and device. The same M0 total system can conform to multiple M1 models, as in Figure 13. Other specializations of product models can be defined, for example, for those only about dynamics, versus those that are only about structure.[30] These specializations can be combined with requirements and designs, for example, to classify some models as only describing the dynamics of the environment (sometimes called "function").

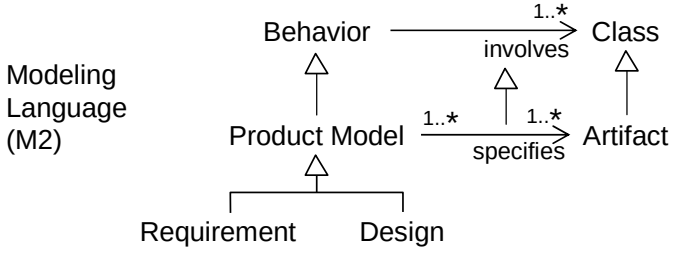


**Figure 25: Product Model**

[30] Requirements can be specialized into those concerning the operation of the device and those concerning its effect, see footnote 24 in Section 4.4.

Artifacts can contain other artifacts, as modeled in Figure 26. Containment is a kind of relation between exactly two artifacts. The notion of containment has various definitions, such as the boundary of the contained artifacts are within the boundaries of the container (topological), or that the contained artifacts contribute to the function of the container (integral), or that some operations on the container apply to the contents, such as movement or destruction (operation propagation). These definitions and others can be modeled as specializations of CONTAINMENT RELATION in Figure 26, but this paper does not propose a more detailed categorization [Winston 1987] [Odell 1994].

Assemblies depend on assembly relations between their contained artifacts, also modeled in Figure 26. Assembly relations are a kind of relation between artifacts. The notion of assembly relation has various definitions, such as contents of the related artifacts are not significantly modified in shape when they are brought together. For example, a table is assembled from its top, legs, and fasteners, but a silicon wafer is not assembled from the materials contained in it.[31] These definitions and others can be modeled as specializations of Figure 26, but this paper does not propose a more detailed categorization [Rachuri 2006]. Assemblies are artifacts composed of interconnected elements (see Section 4.5) where at least one connector uses an assembly relation. Parts are artifacts where no connectors use assembly relations. Individual M0 artifacts are either assemblies or parts, but not both (artifacts are a disjoint union of assemblies and parts). Compound parts have containment relations, such as layers on a silicon wafer, while simple parts do not have containment relations, such as a piece of steel. Individual M0 parts are either simple or compound, but not both (parts are a disjoint union of simple and compound parts). The model for these categories is shown on the lower right of Figure 26 (omitting the disjoint union notations for brevity).

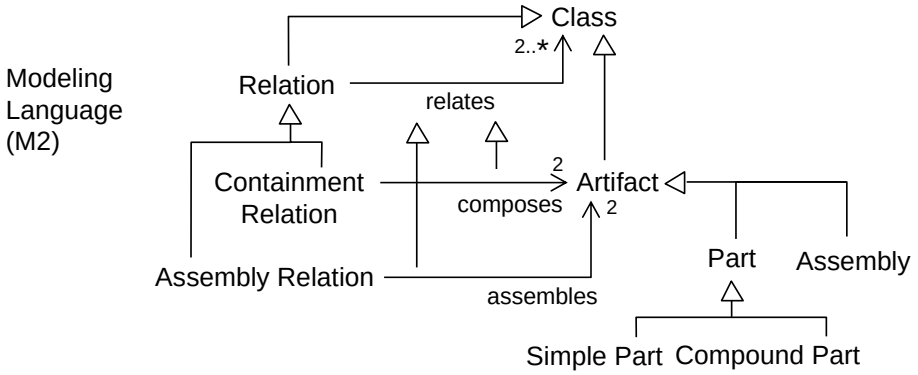


**Figure 26: Containment and Assembly Relations, Artifact Taxonomy**

[31] Assembly relations are a point of integration with production models, for example to capture manufacturing operations [Zha 2006] and tolerancing [Rachuri 2006].

Forms include materials and geometries [Fenves 2008]. They are introduced into the modeling language as a specialization of CLASS, to enable generalization at M1, and categorization of M0 individuals, as shown in Figure 27. Individuals conforming to an M1 material are objects made only of that material, while individuals conforming to an M1 geometry are objects that have a specified shape. For example, an ingot of steel conforms to the M1 STEEL material, while a cylindrically shaped piece of wood conforms to the M1 CYLINDER geometry. The same M0 individual can conform to both a material and geometry (the classes are not disjoint), for example, a cylindrically shaped piece of steel conforms to both STEEL and CYLINDER. Taxonomies of materials and geometries can be defined, for example, Figure 27 has STAINLESS STEEL and RIGHT CYLINDER generalized by STEEL and CYLINDER, respectively. Other taxonomies might include materials that are alloys of other materials, or geometries that apply to both hollow and solid objects, but these are not proposed in this paper.

Forms of artifacts are specified by generalization, as shown at the M1 level in Figure 27. All pipes are cylindrical, with two kinds, one made of plastic, another of copper. The M0 individuals conforming to these also conform to the generalizations, for example, an individual stainless steel pipe conforms to both CYLINDER and STAINLESS STEEL. Materials at M1 can generalize any artifact that is made completely of a single material, including assemblies of parts all made of the same material. Geometries at M1 can generalize any kind of artifact, including assemblies, because geometry is concerned only with surfaces, regardless of the composition of the artifact.

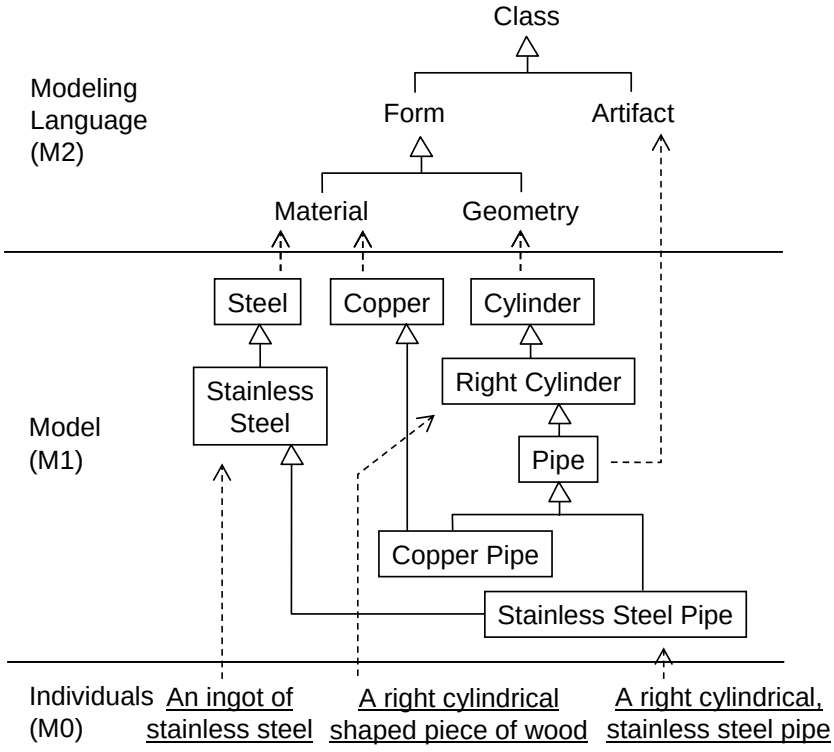


**Figure 27: Form and Artifact**

Another way for modeling languages to make ontology more familiar to engineers is derived relations, for example in assembly and form, as shown in Figure 28. The relations are based on (derived from) other information in the M1 model, and notated with a slash at the beginning of the name:[32]

- /ASSEMBLYOF, /SUBARTIFACTOF: Subartifacts in an assembly are derived from containment relations. For example, in Figure 28 CAR is an assembly of ENGINE and WHEEL (only one derive link shown at M1 for brevity).

- /MADEOF: The materials in an assembly are the material generalizations of the assembly or its components. For example, in Figure 28 CAR is made of STEEL and ALUMINUM (only one derived link shown for brevity).

- /SHAPEDLIKE: The geometries of an artifact are the geometry generalizations. For example, in Figure 28 the geometry of LIGHT HUB is HUB BREP.

Derived relations such as the ones above provide engineers with a familiar view of the M1 model while preserving capabilities depending on M0 individuals (M1 generalization and categorization of M0 individuals). They can be added as necessary to tie engineering-friendly modeling languages at M2 to product ontologies at M1.

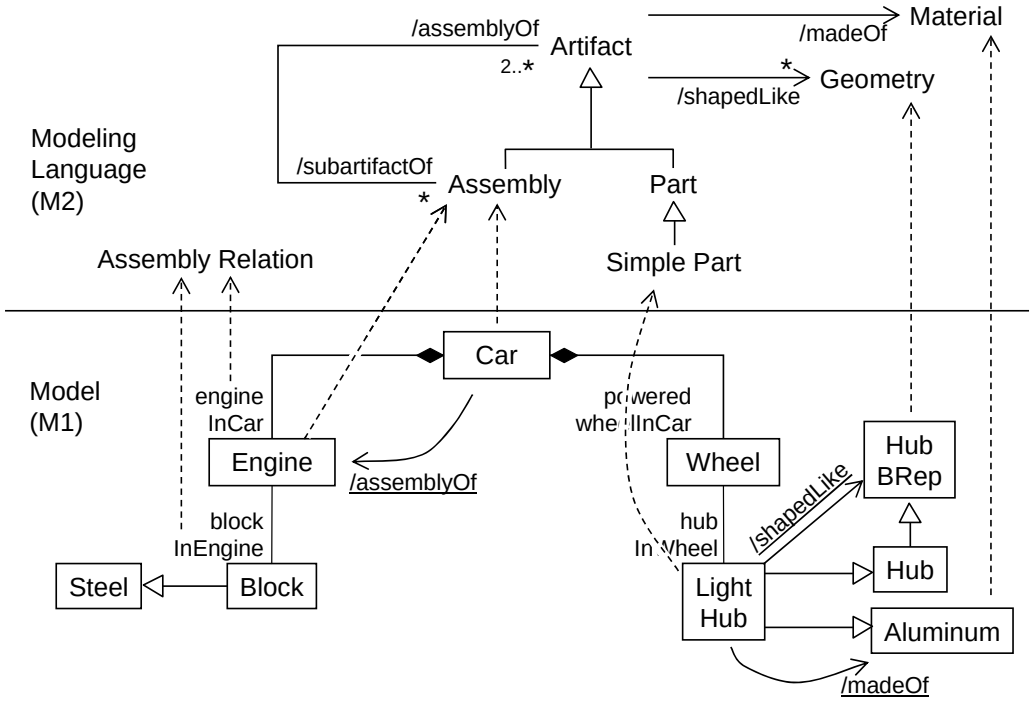


**Figure 28: Derived Relations**

[32] The slash notation is borrowed from derived relations in UML [OMG 2009b].

# 5 Proof of Concept

This section describes a proof-of-concept implementation of the models in Sections 4.5 and 4.6, to show they translate reliably to software implementations. Section 5.1 discusses the architecture and issues in implementation, and Section 5.2 gives an example.

## *5.1 Implementation*

The proof-of-concept implementation uses the Web Ontology Language (OWL) as a modeling language, because of OWL focuses on ontology modeling and has a rigorous mapping from M1 to M0 to support reasoning [W3C 2004a].[33] OWL enables reliable refinement and combination of product models, including consistency checking and validation. OWL does not have composition and behavior models, or explicit metalevels, which are added as extensions in the implementation. UML has these capabilities, but its M1 to M0 mapping is not defined well enough to support automated reasoning. EXPRESS was not chosen because of its reduced expressiveness and lack of rigorous M1 to M0 mapping.

The proof-of-concept implementation was developed in Eclipse [Eclipse 2009], on the JENA API [SourceForge 2009]. Eclipse and JENA are widely used open source platforms for Java and OWL respectively. The JENA API supports specializations of OWL (M2 generalizations), as needed by OPML. The implementation includes a modeling platform API to support modeling levels as described in Section 4.3 and composition as described in Sections 2.2 and 4.5 [Liang 2008].[34] The modeling level support enables access to both the class and individual aspects of M1 elements. Composition support includes automatic maintenance of consistency based on modeling language rules. The rest of the models in Sections 4.5 and 4.6 are layered on the modeling platform API.

The modeling platform as used in this implementation contains three model spaces, each of which holds modeling elements and constraints for a particular modeling level (M2, M1, and M0).[35] Access to the three modeling spaces is through a programming API or OWL files. Each modeling space in the platform provides operations appropriate to it, for example, to define elements in the spaces, and conformance relations between elements in different spaces. Invocations of these operations activate policies registered for the elements in the model spaces to maintain consistency. The policies in the implementation are written in JENA rules. The modeling platform can save the model spaces in separate files to facilitate reasoning limited to less than all three spaces.

[33] OWL is also used a language for specifying the modeling language (metalanguage, see footnote 21 in Section 4.3).
[34] The platform supports the composition model of [Bock 2007]. Only the Full level is used in this paper.
[35] The modeling platform supports any number of modeling levels. The implementation of this paper only used three.

The steps in developing the implementation were:

1. Construct the base and engineering models and their policies (Jena rules) on the modeling platform at the M2 modeling space.
2. Construct the example product models at in the M1 modeling space.
3. Construct example individuals conforming to the product models in the M0 modeling space.
4. Generate three OWL files for the three model spaces, M2, M1, and M0.

Some aspects of the modeling language were not mentioned for brevity in Sections 4.5 and 4.6:

- The classes of a relation are specified in sequence. This gives the proper roles to the participants in links conforming to the relation. For example, in the containment relation the container class is first in the sequence.
- The relations specified by a connector are assumed to have the container as the first related class (DOMAIN in OWL), even if they are not containment relations. Connectors specify relations in the same sequence as the classes of the relation that is the contained class of the connector (RANGE in OWL).
- An M2 element is introduced to classify M1 relations for behavior involvement, INVOLVEMENTRELATION. M1 involvement relations classify M0 links between occurrences of behaviors and the individuals involved in them. The first related class is the behavior (DOMAIN in OWL, see previous bullet), while the second is the involved class (RANGE in OWL). The range classes must be the same as the classes identified by the M2 INVOLVES relation for the behavior. A corresponding M2 element is introduced for SPECIFIES on product models.
- Behavior relations are introduced in the modeling language generalized by behaviors and relations. Behavior relations involve at least two things, which is more restrictive than behaviors in general. Everything involved in behavior relations are also related by them. The elements of the INVOLVES list for behavior relations always include the M1 RELATES classes at the beginning of the list. The range classes of the involvement relations must be same as this list, including duplicates.

Classes and generalization in the model of Section 4.5 translate directly to OWL classes, and relations in the model translate to OWL properties, with some accommodations:

- OWL properties are binary, whereas OPML relations are n-ary. This is addressed by subclassing RELATION from OWL PROPERTY, with the convention that the first two elements in the list of classes related by an M1 relation are the same as the domain and range as an OWL property.
- Some of the relations in OPML can link to the same element twice, for example, RELATES, see Figure 14 in Section 4.5, which OWL properties do not support. This is addressed by using lists as the ranges of OWL properties in these cases, with the convention that the elements of the list are of the type specified in OPML

(for RELATES this is OWL CLASS). Multiplicities translate to limitations on the length of the list. Lists are also needed to specify sequences of classes, as described in the previous bullet.

- The M0 instances of relations in OPML are "tuples" of the related objects, whereas OWL only supports statements (see Resource Description Framework, RDF [W3C 2004b]), which combine relation and tuple. The implementation uses statements as M0 instances of M1 relations (M1 relations are subclasses of OWL STATEMENT), with the convention that all the statements conforming to an M1 relation (OWL property) have the relation or one of its specializations as predicate. For behavior relations, two of the involvement relations must be subproperties of OWL SUBJECT and OBJECT.
- Subclassing of relations in Section 4.5 translates to subproperties in OWL, except when the range is a list, whereupon the elements in the list of a subrelation are in the list of the superrelation.

## *5.2 Example*

This section gives an example using the implementation outlined in Section 5.1. It is a model of fasteners for specific kinds of plates, as shown in Figure 29 (some links between M1 and M2 omitted for brevity). The specified artifact is a fastener, within an environment involving two plates. The requirement is that the two plates must be held in fixed a relative position (the requirement and design for the plates is assumed to be specified elsewhere). Two alternative designs satisfy the requirement, one using a bolt and the other a rivet. The M2 level shows some of the modeling language from Sections 4.5 and 4.6. The inset on the right sketches the requirement portion of the M1 model in notation from Section 4.5, and the corresponding engineering drawings.

The M1 model centers on PLATE FASTENING REQUIREMENT. It involves two plates, by being the domain of two relations INVOLVESPLATE1 and INVOLVESPLATE2 (behavior involvement relations, not shown for brevity). These are connected with a behavior relation, FIXES, ensuring the plates do not move in relation to each other (also shown in an M1 notation in the upper right inset). A fastener is also involved as the specified artifact, PLATE FASTENER, but requirements do not constrain artifacts, so it is not generalized or elaborated.

The engineer introduces designs to meet the requirement in PLATES FASTENING WITH RIVET and PLATES FASTENING WITH BOLT ONLY. These are generalized by PLATE FASTENING REQUIREMENT, to ensure they meet the requirements. They add design constraints on the artifact by:

- specifying special kinds of rivets and bolts for this application, PLATE RIVET and PLATE BOLT.
- establishing a connection between one of the plates and the fastener (not shown on the rivet design for brevity). These connections use behavior relations INTERFERENCEFIT and SCREWFIT, which are generalized by FIXES (generalization not shown for brevity).

- restricting INVOLVESPLATE1 to RIVETED PLATE and BOLTED PLATE, for the domains of INTERFERENCEFIT and SCREWFIT.

At M0, the total systems FIXED PLATES WITH SMALL RIVET #1 and FIXED PLATES WITH SMALL BOLT ONLY #1 conform to the designs for fixing plates by rivet and bolt-only, respectively. Together these involve four individual plates, a bolt and a rivet, conforming to the M1 classes of the requirements and designs.

Ontological product modeling enables requirements and designs about the same product to be independently specified, and checked for consistency when combined (open world). In this example, generalization combines a requirement with a design. The design can be checked for consistency with the requirement, because every M0 individual satisfying the subclass (design) also satisfies the superclass (requirement), by the definition of generalization, see Section 2.2. Generalization also enables other designs to be specified independently and checked against requirements, because a separate generalization is used for each design.

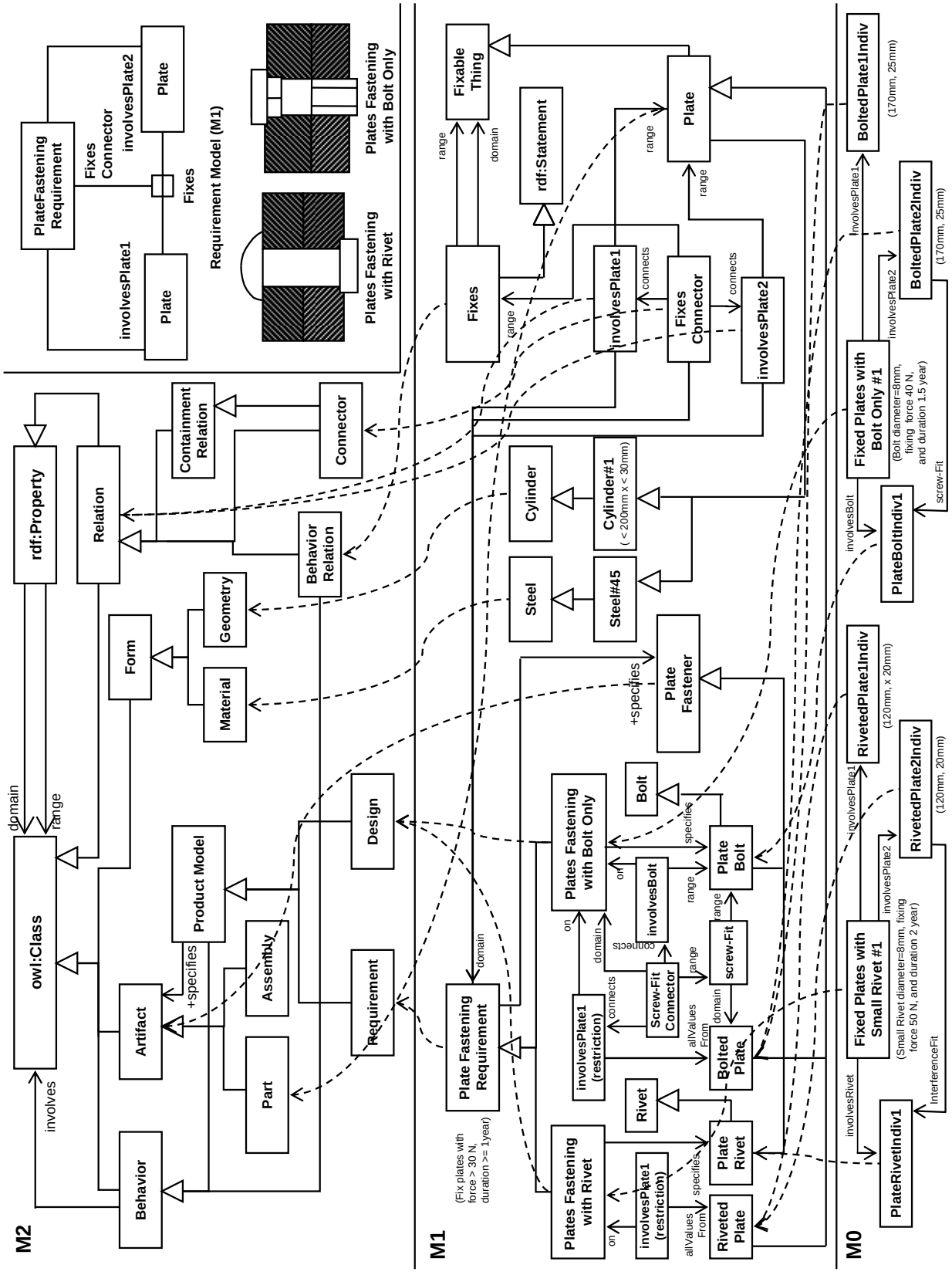


**Figure 29: Example Implementation of Fastener Requirement and Designs**

With the above models, three model spaces are constructed at M2, M1, and M0. OWL files are generated using the proof-of-concept implementation. Figure 30 shows the portions of generated M2, M1, and M0 OWL files.[36]

**M2**

```
<owl:Class rdf:about="http://www.nist.gov/comp/opmbase#Connector">
   <rdfs:subClassOf>
     <owl:Class rdf:about="http://www.nist.gov/comp/opmbase#Relation"/>
     <owl:Class rdf:about="http://www.nist.gov/comp/opmeng#ContainmentRelation"/>
   </rdfs:subClassOf>
 </owl:Class>
<owl:Class rdf:about="http://www.nist.gov/comp/opmeng#Artifact">
   <rdfs:label>Artifact</rdfs:label>
   <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Class"/>
   <rdfs:equivalentClass>
     <owl:Class>
       <owl:unionOf rdf:parseType="Collection">
         <owl:Class rdf:about="http://www.nist.gov/comp/opmeng#Assembly"/>
         <owl:Class rdf:about="http://www.nist.gov/comp/opmeng#Part"/>
       </owl:unionOf>
     </owl:Class>
   </rdfs:equivalentClass>
</owl:Class>
```

**M1**

```
<opmbase:Relation rdf:about="http://www.nist.gov/comp/opmeg1#involvesPlate1">
   <rdfs:domain>
     <opmeng:Requirement rdf:about=
                         "http://www.nist.gov/comp/opmeg1#PlateFasteningRequirement">
       <opmeng:specifies rdf:resource="http://www.nist.gov/comp/opmeg1#PlateFastener"/>
     </opmeng:Requirement>
   </rdfs:domain>
   <rdfs:range rdf:resource="http://www.nist.gov/comp/opmeg1#Plate"/>
</opmbase:Relation>
<opmbase:BehaviorRelation rdf:about="http://www.nist.gov/comp/opmeg1#Fixes">
  <rdfs:range>
    <opmeng:Part rdf:about="http://www.nist.gov/comp/opmeg1#FixableThing"/>
  </rdfs:range>
  <rdfs:domain rdf:resource="http://www.nist.gov/comp/opmeg1#FixableThing"/>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement"/>
</opmbase:BehaviorRelation>
```

**M0**

```
<opmeg1:PlateBolt rdf:about="http://www.nist.gov/comp/opmeg0#PlateBoltIndiv1"/>
<opmeg1:PlateRivet rdf:about="http://www.nist.gov/comp/opmeg0#PlateRivetIndiv1"/>
<opmeg1:PlatesFasteningwithBoltOnly
                   rdf:about="http://www.nist.gov/comp/opmeg0#FixedPlateswithBoltOnly1">
   <opmeg1:involvesBolt rdf:resource="http://www.nist.gov/comp/opmeg0#PlateBoltIndiv1"/>
   <opmeg1:involvesPlate2 rdf:resource="http://www.nist.gov/comp/opmeg0#BoltedPlate2Indiv">
   <opmeg1:involvesPlate1 rdf:resource="http://www.nist.gov/comp/opmeg0#BoltedPlate1Indiv"/>
</opmeg1:PlatesFasteningwithBoltOnly>
<opmeg1:BoltedPlate rdf:about="http://www.nist.gov/comp/opmeg0#BoltedPlate2Indiv">
  <opmbase:screwFit rdf:resource="http://www.nist.gov/comp/opmeg0#PlateBoltIndiv1"/>
</opmeng:BoltedPlate>
<opmeg1:PlatesFasteningwithRivet
                   rdf:about="http://www.nist.gov/comp/opmeg0#FixedPlateswithSmallRivet1">
  <opmeg1:involvesRivet rdf:resource="http://www.nist.gov/comp/opmeg0#PlateRivetIndiv1"/>
  <opmeg1:involvesPlate2 rdf:resource="http://www.nist.gov/comp/opmeg0#RivetedPlate2Indiv"/>
  <opmeg1:involvesPlate1 rdf:resource="http://www.nist.gov/comp/opmeg0#RivetedPlate1Indiv"/>
</opmeg1:PlatesFasteningwithRivet>
<opmeg1:RivetedPlate rdf:about="http://www.nist.gov/comp/opmeg0#RivetedPlate1Indiv">
  <opmeg1:interferenceFit rdf:resource="http://www.nist.gov/comp/opmeg0#PlateRivetIndiv1"/>
</opmeg1:RivetedPlate>
```

**Figure 30: Example OWL in Modeling Spaces**

[36] The M0 level assumes physical behavior occurrences are observed or simulated, then recorded in OWL.

# 6 Future Work

The modeling language described in this paper has a number of limitations to be addressed in future work, including:

- Kinematics and tolerances [Rachuri 2006].
- Product line modeling, such series, families, versions, and platforms [Wang 2003] [Callahan 2006].
- More detailed behavior modeling [ISO 2006] [OMG 2008b].
- Additional product modeling concepts typically used in engineering.
- Implementation in other metalanguages, as needed, such as EXPRESS, for integration with STEP standards.

The language can be applied in a number of ways:

- Design process specifications require classifications of product models, for example, those that are ready for manufacturing. Classification of product models is captured at the M2 level, as are design processes generally.

- Reasoning applications can leverage the relation of M1 and M0 given in this paper. These can provide automated deduction of new information about products.

- Other stages of the product lifecycle besides design can benefit form the approach of this paper, for example, fabrication, maintenance, and disposal. These can be based on models of the device and its environment for the different areas, such as fabrication-of-devices, maintenance-of-devices, and disposal-of-devices.

Benefits of the language can be demonstrated in additional proofs-of-concept:

- Federation of disparate product data sources.
- Improved management of complex products.
- Expanded collaboration from engineering to the broader business.
- Automatic validation of the product knowledge base.
- Product knowledge capture with increased expressiveness.
- Enhanced interoperability and maintainability and more effective reuse of assembly knowledge.
- Facilitating function-based conceptual design.

# 7 Conclusion

This paper describes a product modeling language combining ontology with expanded capabilities in conventional product modeling languages to improve support for collaborative design over earlier approaches. The language treats product models as

ontological classifications of total systems, including behavior of the environment of engineered devices, within a model-based architecture that provides engineer-friendly terminology. It is flexible and accurate in refining, combining, and checking consistency of models of the same product from multiple, disparate sources. The language captures partial and high-level product descriptions, and supports reliable interpretation before and after model interchange. Examples are given in a proof-of-concept implementation.

## Acknowledgements

The authors thank Steven Fenves, Vei-Chung Liang, Fabian Neuhaus, Sudarsan Rachuri, and Eswaran Subrahamian for their input to this paper.

Commercial equipment and materials might be identified to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the U.S. National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

## References

[Ackermann 2004] Ackermann, P., Eichelberg, D., "Product Knowledge Management," International Conference on Economic, Technical and Organizational Aspects on Product Configuration Systems, Technical University of Denmark, Copenhagen, June 2004.

[Biswas 2007] Biswas, A., Fenves, S., Shapiro, V., Sriram, R., "Representation of Heterogeneous Material Properties in the Core Product Model," Engineering with Computers, 2007.

[Bock 2004] Bock, C., "UML 2 Composition Model," Journal of Object Technology, 3:10, pp. 47-73, November-December 2004.

[Bock 2009] Bock, C., "Part-part Relations in an RDF/S and OWL Extension," U.S. National Institute of Standards and Technology, Interagency Report 7507, 2009.

[Borst 1997] Borst, P., Akkermans, H., Top, J., "Engineering ontologies," International Journal of Human-Computer Studies, 46:2-3, pp. 365-406, 1997.

[Bracewell 1994] Bracewell, R. Sharp, J., "A Computer Aided Methodology for the Development of Conceptual Schemes for Mixed Energy-transforming and Real-time Information Systems," in Sharp, J., Ho, V. (eds.), Computer Aided Conceptual Design, LooseLeaf Co., pp.79-94, 1994.

[Callahan 2006] Callahan, S., "Extended Generic Product Structure: An Information Model for Representing Product Families," Journal of Computing and Information Science in Engineering, 6:3, pp. 263-275, September 2006.

[Chan 2007] Chan, E. Yu, K., "A framework of ontology-enabled product knowledge management," International Journal of Product Development, 4:3-4, pp. 241-254, 2007.

[Eclipse 2009] The Eclipse Foundation, “Eclipse, an open development platform,” http://www.eclipse.org, 2009.

[Fenves 2003] Fenves, S., Choi, Y., Gurumoorthy, B., Mocko, G., Sriram, R., “Master Product Model for the Support of Tighter Design-Analysis Integration,” U.S. National Institute of Standards and Technology Interagency Report 7004, May 2003.

[Fenves 2008] Fenves, S., Foufou S., Bock, C., Sriram, R., “CPM 2: A Core Product Model For Product Data,” Journal of Computing and Information Science in Engineering, Special Issue on Engineering Informatics, Transactions of the American Society of Mechanical Engineers, 8:1, pp. 014501-1-014501-6, March 2008.

[Fiorentini 2008] Fiorentini, X., Rachuri, S., Mani, M., Fenves, S., Sriram, R., “An Evaluation of Description Logic for the Development of Product Models,” U.S. National Institute of Standards and Technology Interagency Report 7481, April 2008.

[Flatscher 2002] Flatscher, R., “Metamodeling in EIA/CDIF—Meta-Metamodel and Metamodels,” Association of Computing Machinery Transactions on Modeling and Computer Simulation, 12:4, pp. 322-342, October 2002.

[Gero 1990a] Gero, J., Knowledge-Based Design Systems, Addison-Wesley, 1990.

[Gu 1995] Gu P., Chan, K., “Product Modeling using STEP,” Computer-Aided Design, 27:3 pp. 163-179, March 1995.

[Guarino 1997] Guarino, N., Borgo, S., Masolo, C., “Logical Modelling of Product Knowledge: Towards a Well-Founded Semantics for STEP,” in Proceedings of European Conference on Product Data Technology, pp. 183-190, 1997.

[ISO 2000] International Organization for Standardization, “Integrated generic resource: Product structure configuration,” ISO 10303-44, 2000.

[ISO 2003a] Organization for Standardization, “Implementation methods: XML representations of EXPRESS schemas and data,” ISO 10303-28, 2003.

[ISO 2003b] International Organization for Standardization, “Application protocol: Product life cycle support,” ISO 10303-239, 2003.

[ISO 2004] International Organization for Standardization, “Integrated application resource: Kinematic and geometric constraints for assembly models,” ISO 10303-109, 2004.

[ISO 2005] International Organization for Standardization, “The EXPRESS Language Reference Manual,” ISO 10303-11, 2005.

[ISO 2006] International Organization for Standardization, “Process Specification Language,” ISO 18629, June 2006.

[Kim 2006] Kim, K. Y., Manley, D. G., Yang, H., "Ontology-based assembly design and information sharing for collaborative product development," Computer-Aided Design, 38: 12, pp. 1233-1250, 2006.

[Kitamura 2002] Kitamura, Y., "A functional concept ontology and its application to automatic identification of functional structures," Advanced Engineering Informatics, 16:2, pp. 145-163, 2002.

[Leal 2005] Leal, D., "ISO 15926 'Life Cycle Data for Process Plant': an Overview," Oil and Gas Science and Technology, 60:4, pp. 629-638, 2005.

[Lee 2008] Lee, J., Suh, H., "Ontology-based Multi-layered Knowledge Framework for Product Lifecycle Management," Concurrent Engineering: Research and Applications, 16:4, pp. 301-311, 2008.

[Liang 2008] Liang, V., Zha, X., Bock, C., "An Ontological Modeling Platform," U.S. National Institute of Standards and Technology Interagency Report 7509, June 2008.

[Lin 1996] Lin, J. X., Fox, M. S., Bilgic, T., "A requirement ontology for engineering design," Concurrent Engineering Research and Applications, 4:3, pp. 279-291, 1996.

[Liu 1992] Liu, T., "An object-oriented assembly applications methodology for PDES/STEP based mechanical systems," Ph.D. thesis, The University of Iowa, 1992.

[Metzger 1996] Metzger, F., "The Challenge of Capturing the Semantics of STEP Data Models Precisely," Proceedings of Workshop on Product Knowledge Sharing for Integrated Enterprises, Product Data Technology Advisory Group, ESPRIT Project 9049, 1996.

[Nanda 2006] Nanda, J., Simpson, T., Kumara, S., Shooter, S., "A Methodology for Product Family Ontology Development using Formal Concept Analysis and Web Ontology Language," Journal of Computing and Information Science in Engineering, 6:2, pp.103-113, June 2006.

[Odell 1994] Odell, J., "Six Different Kinds of Composition," Journal of Object-Oriented Programming, 5:8, pp 10-15, January 1994.

[OMG 2008a] Object Management Group, Systems Modeling Language, http://doc.omg.org/formal/08-11-02, 2008.

[OMG 2008b] Object Management Group, Business Process Definition Metamodel, http://doc.omg.org/formal/2008-11-05, 2008.

[OMG 2009a] Object Management Group, Unified Modeling Language: Infrastructure, http://doc.omg.org/formal/09-02-04, February 2009.

[OMG 2009b] Object Management Group, Unified Modeling Language: Superstructure, http://doc.omg.org/formal/09-02-02, February 2009.

[OMG 2009c] Object Management Group, Model-Driven Architecture, http://www.omg.org/mda, 2009.

[Owen 1993] Owen J., STEP: An Introduction, Information Geometers, 1993.

[Pahl 1998] Pahl, G., Beitz, W., Engineering Design: A Systematic Approach, Springer-Verlag, 1998.

[Patil 2005] Patil, L., Dutta, D., Sriram, R., "Ontology-based Exchange of Product Data Semantics," Institute of Electrical and Electronics Engineers Transactions on Automation Science and Engineering, 2:3, pp. 213-255, 2005.

[Rachuri 2006] Rachuri, S., Han, Y., Foufou, S., Feng, S., Roy, U., Fujun W., Sriram, R., Lyons, K., "A Model for Capturing Product Assembly Information," Journal of Computing and Information Science in Engineering, 6:1, pp. 11-21, March 2006.

[Shooter 2000] Shooter, S., Keirouz, W., Szykman, S., Fenves, S., "A Model for the Flow of Design Information in Product Development," Engineering with Computers, 16:3-4, pp. 178-194, 2000.

[SourceForge 2009] SourceForge, "Jena – A Semantic Web Framework for Java," http://jena.sourceforge.net, 2007.

[Sriram 2002] Sriram, R., Distributed and Integrated Collaborative Engineering Design, Sraven Publishers, 2002.

[Sriram 2006] Sriram, R., Szykman, S., Durham, D., Special Issue on Collaborative Engineering, Guest Editorial, Journal of Computing and Information Science in Engineering, 6:2, pp. 93-95, June 2006.

[Stokes 2001] Stokes, M., (ed.), Managing Engineering Knowledge: MOKA Methodology for Knowledge Based Engineering Applications, Professional Engineering Publishing, March 2001.

[Szykman 1998] Szykman, S., Sriram, R., Bochenek, C., Racz, J., "The NIST Design Repository Project," Advances in Soft Computing - Engineering Design and Manufacturing, Roy, R., Furuhashi, T., Chawdhry, P. (eds.), Springer-Verlag, pp.5-19, 1999.

[Szykman 2001a] Szykman, S., Fenves, S., Keirouz, W., Shooter, S., "A foundation for interoperability in next-generation product development systems," Computer-Aided Design, 33:7, pp. 545-559, 2001.

[Szykman 2001b] Szykman, S. Sriram, R., Regli, W., "The Role of Knowledge in Next-Generation Product Development Systems," American Society of Mechanical Engineers Journal of Computing and Information Sciences in Engineering, 1:1, pp.3-11, 2001.

[W3C 2004a] World Wide Web Consortium, "OWL Web Ontology Language Guide," Smith, M., Welty, C., McGuinness, D. (eds.) http://www.w3.org/TR/2004/REC-owlguide-20040210, February 2004.

[W3C 2004b] World Wide Web Consortium, "Resource Description Framework (RDF): Concepts and Abstract Syntax," Klyne, G., Carroll, J. (eds.), http://www.w3.org/TR/2004/REC-rdf-concepts-20040210, February 2004.

[W3C 2006] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0," http://www.w3.org/TR/REC-xml, September 2006.

[Wang 2003] Wang, F., Fenves, S., Sudarsan, R., Sriram, R., "Towards Modeling the Evolution of Product Families," Proceedings of the American Society Of Mechanical Engineers Design Engineering Technical Conferences, 2003.

[Winston 1987] Winston, M., Chaffin, R., Herrmann, D., "A Taxonomy of Part-Whole Relations," Cognitive Science, 11:4, pp. 417-444, 1987.

[Xu 2005] Xu, C., Gupta, S., Yao, Z., Gruninger, M., Sriram, R., "Towards Computer-Aided Conceptual Design of Mechatronic Devices with Multiple Interaction-States," Proceedings of the American Society of Mechanical Engineers Design Engineering Technical Conferences, 2005.

[Yang 2008] Yang, D., Dong, M., Miao, R., "Development of a product configuration system with an ontology-based approach," Computer-Aided Design, 40:8, pp. 863-878, 2008.

[Zave 1997] Zave, P., Jackson, M., "Four Dark Corners of Requirements Engineering," Association for Computing Machinery Transactions on Software Engineering and Methodology, 6:1, pp. 1-30, January 1997.

[Zha 2002] Zha, X., Du, H., "A PDES/STEP-based model and system for concurrent integrated design and assembly planning," Computer-Aided Design, 34:14 pp. 1087-1110, December 2002.

[Zha 2005a] Zha, X, Fenves, S., Sriram, R., "A Feature-based Approach to Embedded System Hardware and Software Co-design," Proceedings of the American Society of Mechanical Engineers Design Engineering Technical Conferences, 2005.

[Zha 2005b] Zha, X., Sriram, R., Gupta, S., “Information and Knowledge Modeling for Computer Supported Microelectromechanical Systems Design and Development,” Proceedings of the American Society Of Mechanical Engineers Design Engineering Technical Conferences, 2005.

[Zha 2006] Zha, X., Foufou, S., Sudarsan, R., Sriram, R., “Analysis and Evaluation for STEP-based Electromechanical Assemblies,” Journal of Computing and Information Science in Engineering, 6:3 pp. 276-287, 2006.