Advanced Engineering Informatics 24 (2010) 510–524

Contents lists available at ScienceDirect

Advanced Engineering Informatics

journal homepage: www.elsevier.com/locate/aei

Ontological product modeling for collaborative design

Conrad Bock [a,*], XuanFang Zha [a], Hyo-won Suh [b], Jae-Hyun Lee [a]

[a] *U.S. National Institute of Standards and Technology, 100 Bureau Dr., MS 8263, Gaithersburg, MD 20899-8263, USA*
[b] *Department of Industrial System Engineering, Korea Advanced Institute of Science and Technology, 373-1, Guseong-dong, Yuseong-gu, Daejon 305-701, Republic of Korea*

ARTICLE INFO

Article history:
Received 15 November 2007
Received in revised form 20 June 2010
Accepted 27 June 2010
Available online 22 July 2010

Keywords:
Product modeling languages
Ontology
Collaborative design

ABSTRACT

This paper shows how to combine ontological and model-based techniques in languages that facilitate collaborative design exploration. The proposed approach uses ontology to capture alternative designs and incremental refinements that meet requirements and earlier design commitments. Model-based techniques are applied to develop more powerful, engineering-friendly languages for using ontology. It uses ontology's open world semantics to support design collaboration with flexible and accurate design combination, refinement, and consistency checking. It also leads to more reliable interpretation of models across the product lifecycle due to more rigorous language semantics. An example language is described using these techniques.

Published by Elsevier Ltd.

1. Introduction

Product design has become increasingly complicated due to many additional factors that must be considered, and a wider range of collaborators involved [1,2]. Designers are aware of many more stages of the product lifecycle. They interact with a wide range of other designers, especially in sophisticated products. Consideration of more lifecycle stages and more collaboration result in better products [3,4], but also place a significant burden on designers to examine a larger set of alternative designs at varying levels of detail. Designers do not know about all the lifecycle stages at once, and usually no single design will optimize for all criteria at all stages. They must develop many alternatives, from less to more detail, in consultation with many other engineers. Collaboration is often hampered by lack of uniform interpretation of product modeling languages and terminologies, leading to rework when discrepancies are discovered. Designers distributed geographically and organizationally in global economies worsen these problems [5]. Product information assets become fragile and return less on investment.

This paper addresses these challenges by applying ontology in an engineering-friendly way through model-based techniques. Ontologies bring a taxonomic approach to managing design complexity, and an "open world" semantics enabling independently developed product models to partially describe the same products. They also have 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. 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.

Section 2 gives requirements on product models and languages. Section 3 covers previous work on these requirements. Section 4 covers an example language addressing the requirements, with brief introductions to ontology and model-based architecture. Section 5 concludes the paper.

2. Requirements on product models and languages for collaboration

Product modeling languages are engineered just as products are, including requirements and designs. Language requirements are naturally intertwined with the kinds of models expected to be constructed by engineers or other stakeholders. This section takes the scenario of collaborative design exploration 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 and example language described 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

* Corresponding author. Tel.: +1 301 975 3818; fax: +1 301 975 8273.
E-mail addresses: conrad.bock@nist.gov (C. Bock), xuanfang.zha@nist.gov (X. Zha), hw_suh@kaist.ac.kr (H.-w. Suh), jaehyun.lee@nist.gov (J.-H. Lee).

1474-0346/$ - see front matter Published by Elsevier Ltd.
doi:10.1016/j.aei.2010.06.011

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, 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 little is specified. 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. 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.

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 might apply to some objects 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*) [6]. 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 8:05 am and 8:30 am 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.

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:

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

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 [7], 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.

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. For example, a general model of cars might describe them as weighing less than 2000 kg, but in a special model for small cars they might be less than 1000 kg. All objects fitting the description of small cars will fit the description of cars in general. Fig. 1 shows a notation for generalization, the hollow-headed arrow, adapted from the Unified Modeling Language (UML) [8], where it has the same meaning as this paper, and as subclassing in the Web Ontology Language (OWL) [9]. In the figure, vehicles generalize cars and boats (cars and boats are vehicles), or more precisely, all objects fitting the description of the car or boat models will fit the description of the vehicle model.

Generalization provides a simple and well-defined framework for collaborative design exploration. Engineers can independently refine models for the same product with alternative specializations

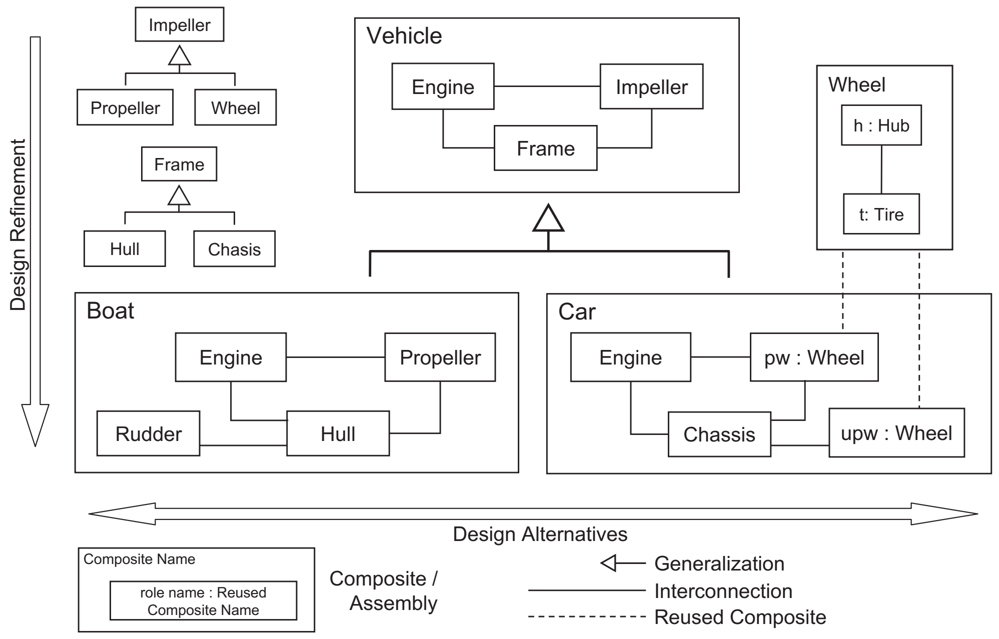

Fig. 1. Product generalization.

of a general platform, and combine them later with multiple generalizations of the final product model. Specialized models that do meet requirements can be abandoned, returning to more general models and alternative specializations. Abandoned models are still available for further adaptation if requirements change, or other solutions turn out to be worse overall. Generalization ensures specialized models are compatible with general ones, keeping the design space more organized than typical versioning techniques, where any kind of change can occur between versions. More discussion of this is in Section 4.4.

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.” Fig. 1 illustrates this in a simplified car model on the lower right using a wheel model twice, in different ways, shown with notation adapted from the UML composite structure diagram [8,10]. The colon notation for the wheels specifies a reused subassembly to the right of the colon, and the way it is used (“roles”) to the left of the colon (role names are omitted for brevity on the other parts). One wheel usage is powered by the engine and the other is not, as shown by the lines between the engine and one of the wheels. The resulting product will have two subassemblies specified by the wheel model, only one of which will be connected to the engine. The examples here are just for devices, but the environment can have interconnections also.

Generalization applies to refinement of interconnections and subassemblies, also illustrated in Fig. 1. Vehicles include an element that exerts force outside the vehicle (the impeller), which generalizes wheels and propellers, as shown in the upper right of Fig. 1. Similarly, frames in vehicles generalize chasses and hulls 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 (not shown for brevity). 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.

Interconnections within an assembly or composed product model can be specified as a composition of other interconnected elements, as illustrated in Fig. 2. The connection between engine and wheel in manual transmission cars on the upper right decomposes into a separately specified set of interconnected elements that includes a clutch and gearbox for manual transmissions. Things described by the manual 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, a chemical plant might have the same piping patterns between tanks used many times, see Fig. 3. This enables product designers to reuse complex interconnections defined by others.

Alternative product model refinements using generalization and interconnection composition are also illustrated in Fig. 2. 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 between engines and wheels, such as power delivery requirements, as explained next.

Behaviors can be taken as relations between the objects involved in them, enabling behaviors to interconnect elements in a composite or assembly. For example, a behavior for the interconnections between a table top and its leg might specify that the objects involved have small relative movement. Fig. 2 shows another example of a behavior on the lower left used by an interconnection in the assembly above it. The behavior describes power transmission between the engine and wheel, potentially establishing maximum and minimum limits. The model generalizes two others with decompositions satisfying 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.

Interconnections can be interconnected in a product model, as illustrated in Fig. 3. Two interconnections that decompose into piping elements are themselves interconnected. The individual physical devices described by the assembly will have two individual pipes with heat transfer between them. This can be combined with generalization similarly to Fig. 2, for example, to specify thermal conduction between the interconnections without specifying that it is achieved with piping. Then the thermal connection applies to specializations of the assembly that connect the units in different ways.

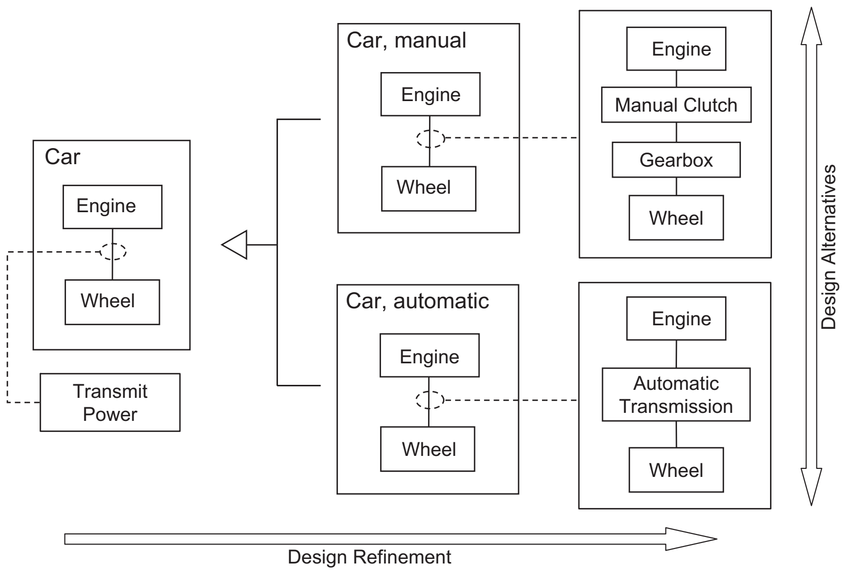

Fig. 2. Interconnection decomposition.

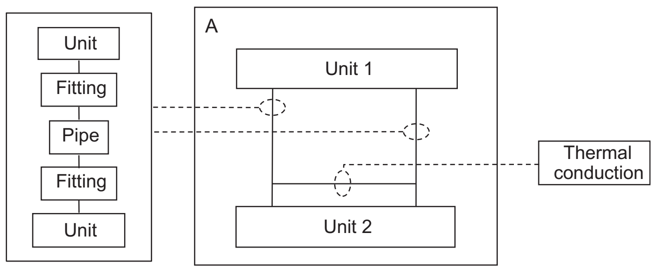

Fig. 3. Interconnections of interconnections.

3. Previous work

Previous work in product modeling takes either an ontology 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. 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 have been used much longer than ontological techniques. They provide engineering-oriented languages, but specify their meaning informally, and lack the benefits of ontology. These languages include ISO 10303 (STEP, STandard for the Exchange of Product model data) [11], the Unified Modeling Language (UML) [8], and its extensions in the Systems Modeling Language (SysML) [12], the U.S. National Institute of Standards and Technology (NIST) Core Product Model (CPM) [13] and its extension in the Open Assembly Model (OAM) [14], and Methodology and tools Oriented to Knowledge-based engineering Applications (MOKA) [15]. This section reviews these efforts.

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 and mechanical products (application protocols). STEP gives a standard terminology for product management system vendors [16]. STEP models have assembly support within some of the application protocols, but limited assembly representations across subdomains. For example, ISO 10303-44 specifies the assembly structure of a product and manages its configuration during its lifecycle [17]. 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 defines kinematic and geometric constraints for mechanical assemblies, including assembly feature relationships and constraint schemas [18]. ISO 10303-239 provides capabilities for product lifecycle support, including product lifecycle activity management, product description, operational feedback, and support solution and environment [19]. 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 [18], but it does not cover assembly generalization and relation decomposition for configuration management of assemblies and components, nor behaviors as relations and interconnections. Extensions of STEP-compatible models have been proposed to address some of the missing capabilities in STEP assembly models [20,21], for example, introducing entities such as assembly, subassembly, part, and connector for integrated assembly design and process planning. 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 [22]. 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 behaviors as relations or interconnections. SysML and UML do not 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 [23]. Heterogeneous Material Model [24], Mechatronic Device Model [25], the Product Family Evolution Model [26], and the Embedded System Model [27]. 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 [21] to completely capture the detailed geometric information using XML Schema [28]. 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 [29,25].

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 from combining ontology and modeling languages 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) in its definition of function, or requirements modeling generally. This makes it difficult to apply MOKA during conceptual design. MOKA supports interconnection of elements in a composition, but not behaviors 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 [30]. 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 develops ontologies for the static aspects of individual physical objects, such as relations between parts and wholes, congruence, convexity, and perpendicularity [31], or for dynamic aspects using qualitative physics and packaged in reusable model libraries [32]. Other work extends these general physical concepts to include dynamics, then relates dynamics to mathematical equations for use in simulation [33]. 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 [34]. Some research explores how these product taxonomies can be used in knowledge management and configuration systems [35,36], while another effort gives a methodology for defining product family ontologies [37]. 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 defining existing or new product modeling languages, rather than classifying physical objects or occurrences of behaviors. Some research gives taxonomies of functions, such as providing material, preventing a side-effect, or driving a process [38]. 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 [39]. Other research uses ontology to formalize an assembly language that was defined in UML, and integrate it into design tools [40]. Some research formalizes the syntax of product modeling languages using ontology languages [41,42], or uses ontology to evaluate the syntax [43]. 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 product taxonomies or 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 [44–46]. These efforts do not draw out the implications of ontology for collaborative design exploration (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.

4. An ontological product modeling language

This section describes an example product modeling language that provides the benefits of ontology in an engineer-friendly way through 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 described in this paper, such as relations, while Section 4.6 covers extensions for engineering concepts. Together these last two sections describe an example language using the techniques 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.[1] 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.

Ontology is usually formalized with set theory, where members of sets are actual or intended things, and rules for membership (called *classes*) capture expert knowledge and specifications. In product modeling, members of 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).[2] The members of the set are said to *conform* to the class, be *classified* by the class, or informally, be “described” by the class. Classes are sometimes informally called “categories” or “types.”

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

- 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, for example, sets 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.[3] For example, Fig. 4 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.

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.

[1] See Footnote 5 in Section 4.3 regarding ontologies of linguistic entities.

[2] 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.

[3] This distinguishes ontological reasoning from rule systems, which require instances to perform inference, due to closed world semantics. 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 physical products usually do not exist before the models.

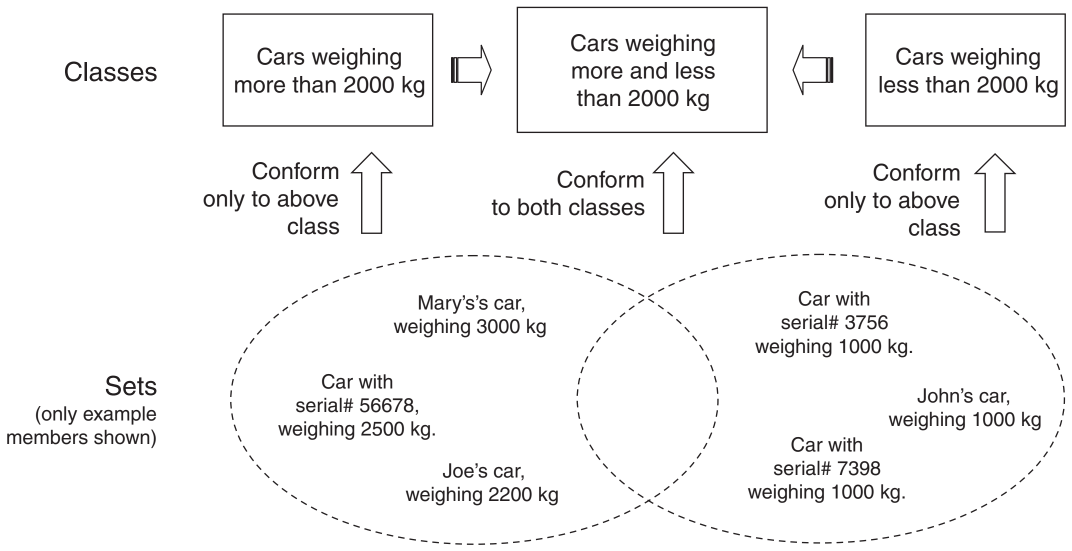
Classes
Cars weighing more than 2000 kg
Cars weighing more and less than 2000 kg
Cars weighing less than 2000 kg
Conform only to above class
Conform to both classes
Conform only to above class
Sets (only example members shown)
Mary's's car, weighing 3000 kg
Car with serial# 56678, weighing 2500 kg.
Joe's car, weighing 2200 kg
Car with serial# 3756 weighing 1000 kg.
John's car, weighing 1000 kg
Car with serial# 7398 weighing 1000 kg.

Fig. 4. Classes and sets.

- 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 Fig. 5. 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 Fig. 5. This model limits the weight of the car involved in the occurrence to less than 2000 kg. This is just an 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 Fig. 5. 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.

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 [9]. This is illustrated under the left box in Fig. 6, with one ontology language element, Class, shown at the at the *M2 level*, which is where languages are defined.[4,5] 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.[6]

[4] The levels in Fig. 6 are numbered by OMG's Model-Driven Architecture [50,22], inherited from the Electronic Industries Alliance's Computer-Aided Software Engineering Data Interchange Format [51].

[5] 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. Ontology languages can be used as metalanguages, classifying linguistic entities rather than real things (syntax rather than semantics). This ensures precise specification of modeling languages, as in this paper and some previous work described in Section 3.2, but does not provide the benefits of ontology to engineers working at M1.

[6] 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.

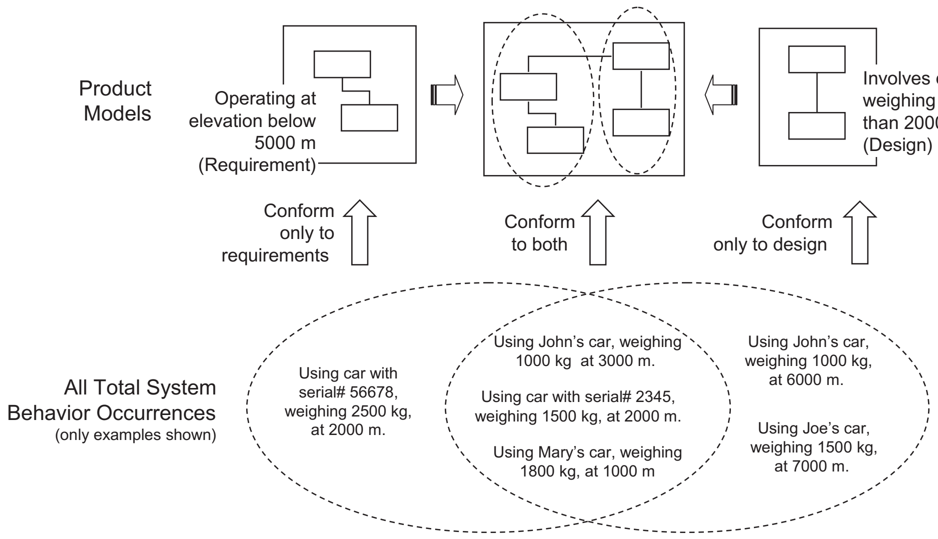
Product Models
Operating at elevation below 5000 m (Requirement)
Involves device weighing less than 2000 kg (Design)
Conform only to requirements
Conform to both
Conform only to design
All Total System Behavior Occurrences (only examples shown)
Using car with serial# 56678, weighing 2500 kg, at 2000 m.
Using John's car, weighing 1000 kg at 3000 m.
Using car with serial# 2345, weighing 1500 kg, at 2000 m.
Using Mary's car, weighing 1800 kg, at 1000 m
Using John's car, weighing 1000 kg, at 6000 m.
Using Joe's car, weighing 1500 kg, at 7000 m.

Fig. 5. Product models describing behavior occurrences.

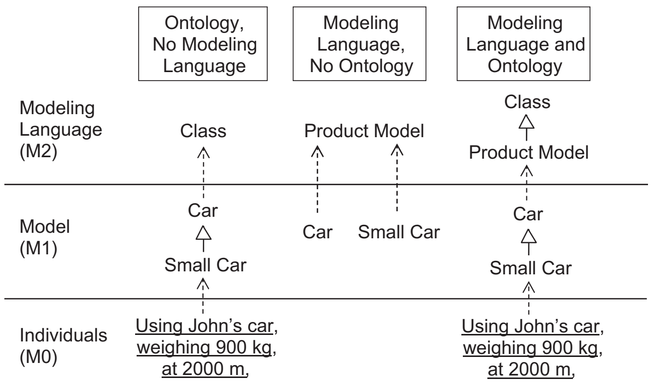
Ontology, No Modeling Language
Modeling Language, No Ontology
Modeling Language and Ontology
Modeling Language (M2)
Model (M1)
Individuals (M0)
Class
Product Model
Class
Product Model
Car
Small Car
Car
Small Car
Car
Small Car
Using John's car, weighing 900 kg, at 2000 m,
Using John's car, weighing 900 kg, at 2000 m,

Fig. 6. Ontology and product modeling languages.

Using only ontology languages at M2, as under the left box of Fig. 6, 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 (see Section 2.2 about generalization notation), and to classify actual cars at M0 (notated with a dashed arrow, the conformance link from Car and Small Car to Class is omitted for brevity), 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 will not 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 Fig. 6, 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. 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 Fig. 6. It includes Class as a generalization of Product Model in the modeling language at M2.[5] 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.

The modeling technique illustrated on the right of Fig. 6 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 Fig. 6 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 Fig. 6, and the implication of M1 models for M0 individuals (*semantics*).

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 Fig. 7. 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

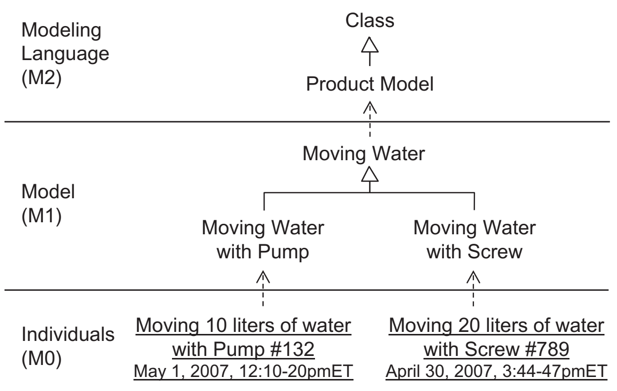
Modeling Language (M2)
Class
Product Model
Model (M1)
Moving Water
Moving Water with Pump
Moving Water with Screw
Individuals (M0)
Moving 10 liters of water with Pump #132 May 1, 2007, 12:10-20pmET
Moving 20 liters of water with Screw #789 April 30, 2007, 3:44-47pmET

Fig. 7. Alternative designs for the same requirement.

specialized models introduce devices, a pump and Archimedes screw, respectively (conformance links from the specialized M1 models to Product Model are omitted for brevity). 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.

Requirements and designs are kinds of product models, as shown at the M2 level in Fig. 8. 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 10,000 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 Fig. 5 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 increase the accuracy and completeness of testing built products against requirements. Fig. 8 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 [47].[7] 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.

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). 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, a mechanical relation has conforming links at M0 that might be categorized into fixed and moving (see Fig. 10 for another example). Relations are always between other things, modeled in Fig. 9 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.

Relations by themselves are insufficient for capturing the composition of interconnected elements, for example, the interconnections between engines and wheels in cars in Fig. 1 in Section 2.2 [10]. 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 spare wheel in the same car, or a propeller in a boat. 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.

A solution to the above problems is illustrated in Fig. 9, notated with an adapted form of UML class diagrams at the user model level (M1),[8,9] and some of the conformance links shown. 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. These “whole–part” relations are notated with UML composition notation, with the black diamond on the composite end [8,10].
(2) Treat “part–part” interconnections like any other element of the composite, with the additional capability of connecting the whole–part 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 from the composite, just like the relations for engines and wheels in the first item above. In Fig. 9, 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 Fig. 1 in Section 2.2, as well as connected by other connectors, as in Fig. 3.

[7] 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. Requirements can be divided into those concerning the operation of the device and those concerning its effect.

[8] Rectangles as in UML are used at M1 in the rest of the paper to distinguish it from the other levels.

[9] UML composite structure diagrams are a compact and scalable notation for this, see Fig. 1 in Section 2.2 [8,10]. 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 can be shown to the left of a colon in the nested rectangle labels. The interconnections are shown as lines between the nested rectangles.

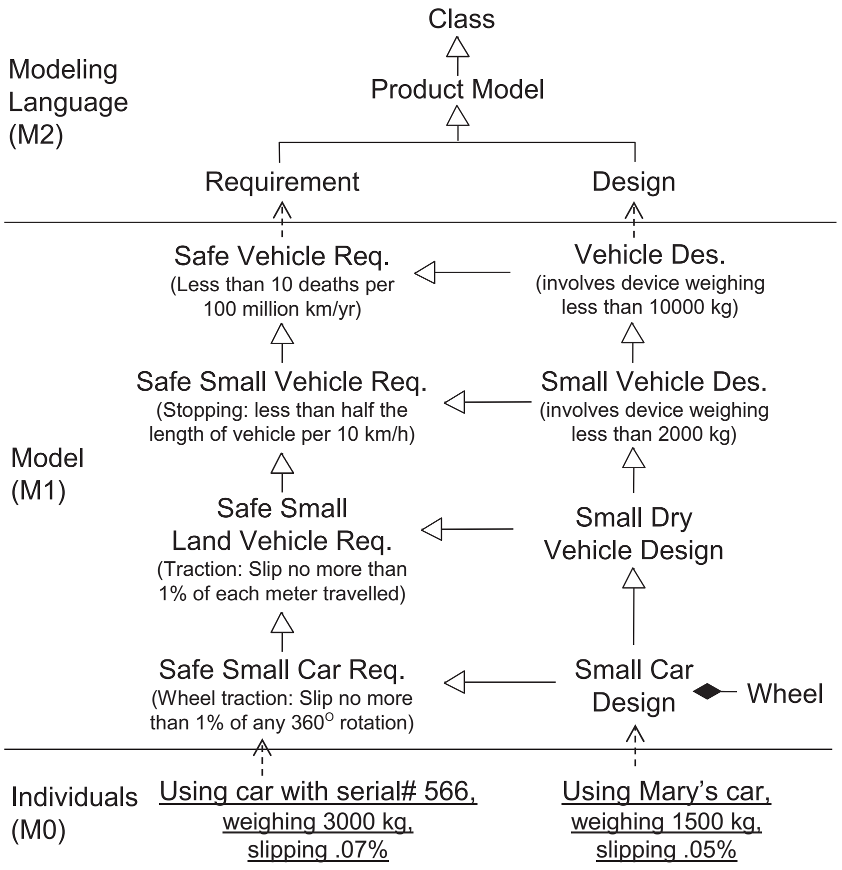
Class
Product Model
Modeling Language (M2)
Requirement
Design
Safe Vehicle Req. (Less than 10 deaths per 100 million km/yr)
Vehicle Des. (involves device weighing less than 10000 kg)
Safe Small Vehicle Req. (Stopping: less than half the length of vehicle per 10 km/h)
Small Vehicle Des. (involves device weighing less than 2000 kg)
Model (M1)
Safe Small Land Vehicle Req. (Traction: Slip no more than 1% of each meter travelled)
Small Dry Vehicle Design
Safe Small Car Req. (Wheel traction: Slip no more than 1% of any 360° rotation)
Small Car Design
Wheel
Individuals (M0)
Using car with serial# 566, weighing 3000 kg, slipping .07%
Using Mary's car, weighing 1500 kg, slipping .05%

Fig. 8. Requirements and designs.

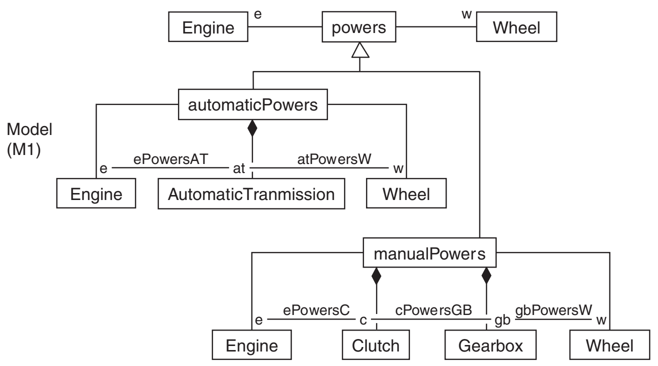

Fig. 10. Relation generalization with alternative decompositions.

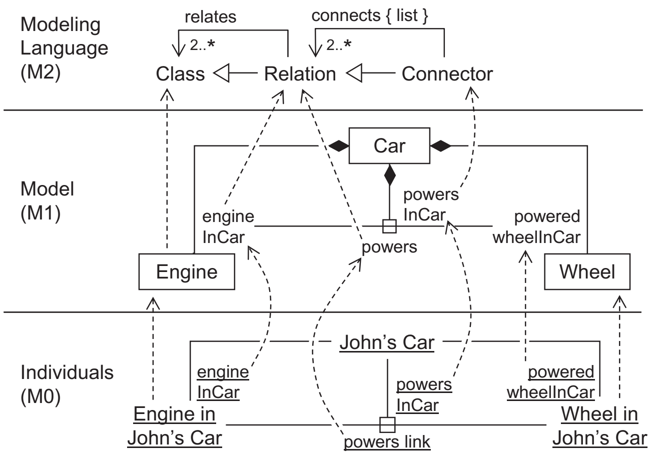

Fig. 9. Relations and interconnections.

Interconnections can be specified between subassemblies of subassemblies, for example, between the hubs of the wheels in Fig. 2 in Section 2.2 and the crankshaft of the engine (not shown for brevity). A connector between the hubs and the crankshaft is insufficient, because not all hubs are in powered wheels. The modeling language (M2) specifies the ends of connectors with a series of relations from the whole, as shown in Fig. 9 by the "{list}" annotation on CONNECTS. In this example, one end of the connector would refer to a list of POWEREDWHEELINCAR and HUBINWHEEL (a relation between wheels and hubs), to power only the hubs in powered wheels, while the other end refers to a list of ENGINEINCAR and CRANKSHAFTINENGINE (a relation between engines and crankshafts), to take power from 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 the interconnections in it are between 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 approach is similar to Fig. 9, except the composite element is a relation, as illustrated in Fig. 10. The manualPowers relation at the lower right 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 Fig. 1 in Section 2.2). The relations identifying the objects linked by manualPowers, e and w. do not have diamonds because they are not contained in manualPowers. The four elements are interconnected by other relations, such as ePowersC and cPowersGB, as specified by the connectors (relations to the connectors are omitted for brevity). Individual links at M0 conform to manualPowers by having interlinked elements, such as an individual engine and clutch in an individual car linked in conformance to ePowersC. Individual M0 links for composite relations can be established by M1 connectors, such as a powers link established by the connector at M1 in Fig. 9.

Relations can generalize others that have alternative decompositions, like classes in general. For example, the powers relation of Fig. 9 might generalize relations for manual and automatic transmissions, as illustrated in Fig. 2 in Section 2.2. The powers relation does not introduce subassemblies between the engine and wheels, leaving these to its specializations, as shown in Fig. 10 (M2 and M0 omitted for brevity). The relations between powers and its elements, e and w, are available in automaticPowers and manualPowers, because M0 links conforming to automaticPowers or manualPowers also conform to powers, by the definition of generalization. The specialized power relations each have their own way of interconnecting the engine and wheels, using different subassemblies.

Specialized relations can be used to specialize interconnections, for example, the connector between engines and wheels in Fig. 2 in Section 2.2. The model for this is shown in Fig. 11. Cars in general use the powers relation, while specialized cars use the manualPowers and automaticPowers 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 Fig. 11. This means M0 links conforming to automaticPowersInCar and manualPowersInCar also conform to powersInCar. For example, if John's car is manual, then the link between the engine and wheels in it will conform to manualPowers and powers, and the link will be identified by both manualPowersInCar and powersInCar.

Interconnections can be interconnected, for example, the connector between the pipes in Fig. 3 in Section 2.2. The model for this is shown in Fig. 12. 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 the thermal relation. The thermal connection applies to specialized assemblies also, like the rest of the elements of A, in particular to specialized kinds of plumbing between the units.

Behaviors are introduced into the modeling language (M2) as a specialization of Class, as shown in Fig. 13 (through Relation, described next). This enables M1 behavior models to be generalized at M1, and to have conforming behavior occurrences at M0. For example, a rotation behavior has conforming occurrences at M0 that might be categorized into fast and slow rotations. Behaviors always include things that are behaving, modeled at M2 with involves, to identify the kinds of things involved in the behavior, as shown in Fig. 13.[10] The M1 TransmitPower behavior involves Engine and Impeller, as identified by two relations, e and i, without diamonds, because the engine and impeller are not contained by the behavior (conformance links from Impeller and i are not shown for brevity). Occurrences of TransmitPower at M0 will involve individual engines and impellers conforming to Engine and Impeller, respectively.

Behaviors can generalize others involving additional and specialized elements, like classes in general. For example, a behavior that transmits power generalizes other behaviors doing this in different ways, as shown Fig. 13. One specialized behavior involves a wheel and the other a propeller, both special kinds of impellers. The specialized behaviors transmit power, because their M0 occurrences are also occurrences of TransmitPower, by the definition of generalization.

Behaviors can be treated 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. The behaviors in Fig. 13 relate engines to impellers of various kinds. This enables behaviors and relations to form generalizations, as in Fig. 13, where a behavior for transmitting power generalizes the power relations from Fig. 10. Behavior relations also enable behaviors to be used as connectors in assemblies, for capturing functional requirements, as in Fig. 11 where specialized power transmission behaviors connect elements of cars and boats. Behaviors as relations is introduced into the modeling language (M2) with Relation generalizing Behavior, as shown in Fig. 13, meaning all M1 behaviors are relations, by the definition of generalization. And relates generalizes involves, meaning things involved in behaviors are also in relation to each other. Generalizing relations means links conforming to the special relation also conform to the general one, for example, people's sisters are also their siblings, or in Fig. 13, anything involved in behaviors are also related by behaviors.[11]

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 Fig. 14. 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

[10] 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.

[11] This is property subsetting in UML [8], where it is notated textually rather than graphically as in Fig. 13, and has the same semantics as in this paper, and as subproperties in OWL [9].

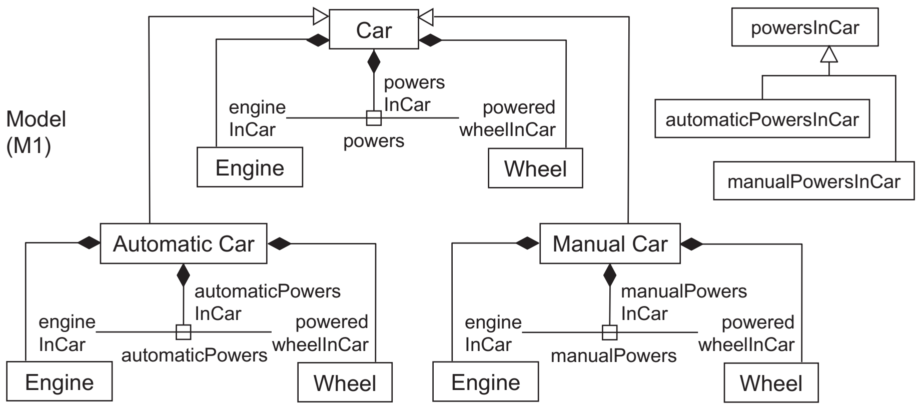

Fig. 11. Generalization of interconnected elements using relation generalization.

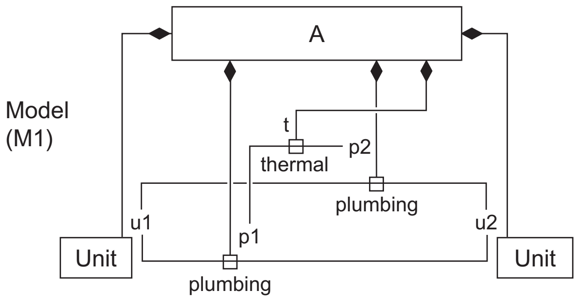

Fig. 12. Interconnection of interconnections.

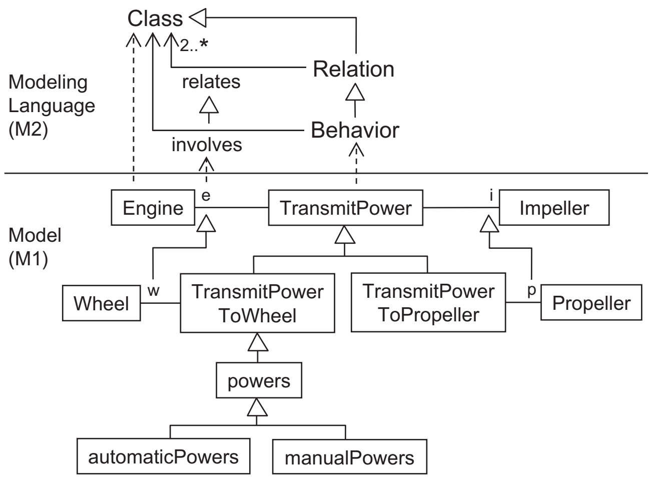

Fig. 13. Behavior generalization and behavior relations.

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 Fig. 8. Other specializations of product models can be defined, for example, for those only about dynamics, versus those that are only about structure. 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”).

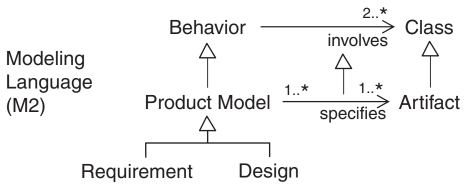

Fig. 14. Product models.

Artifacts can contain other artifacts, as modeled in Fig. 15. 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 Fig. 15, but this paper does not propose a more detailed categorization [48,49].

Assemblies depend on assembly relations between their contained artifacts, also modeled in Fig. 15. 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. These definitions and others can be modeled as specializations of Fig. 15, but this paper does not propose a more detailed categorization [14]. 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 Fig. 15 (omitting the disjoint union notations for brevity).

Forms include materials and geometries [13]. 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 Fig. 16. Individuals conforming to an M1 material are objects made only of that material, while individuals conforming to

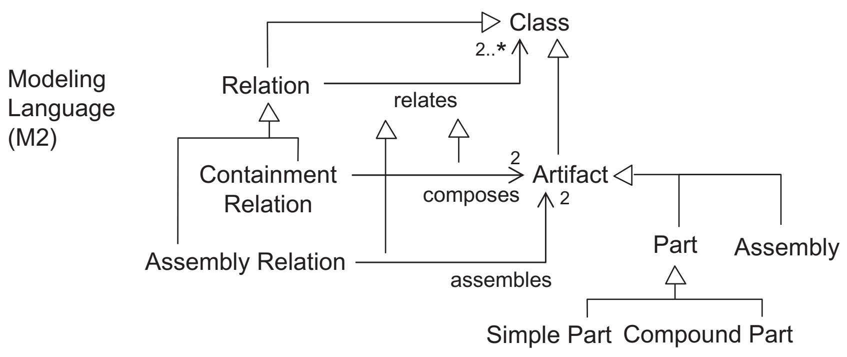

Fig. 15. Containment and assembly relations, artifact taxonomy.

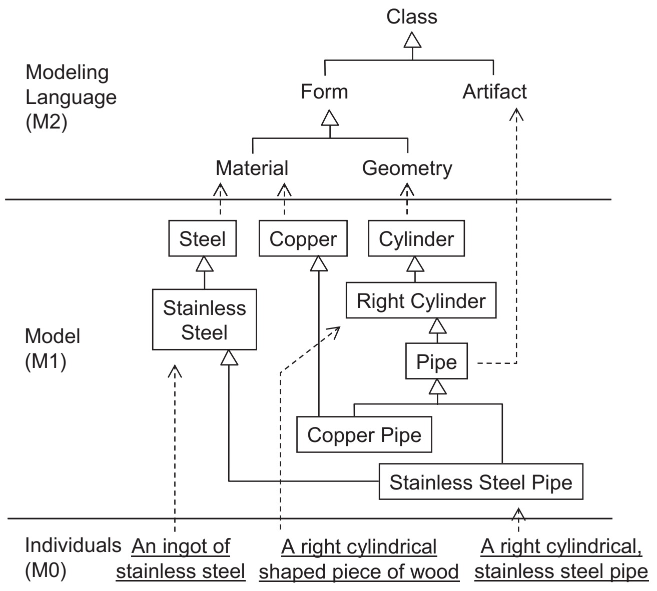

Fig. 16. Forms and artifacts.

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, Fig. 16 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 Fig. 16. 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.

Another way for modeling languages to make ontology more accessible to engineers is derived relations, for example, in assembly and form, as shown in Fig. 17. The relations are based on (derived from) other information in the M1 model, and notated with a slash at the beginning of the name, borrowed from derived relations in UML [8]:

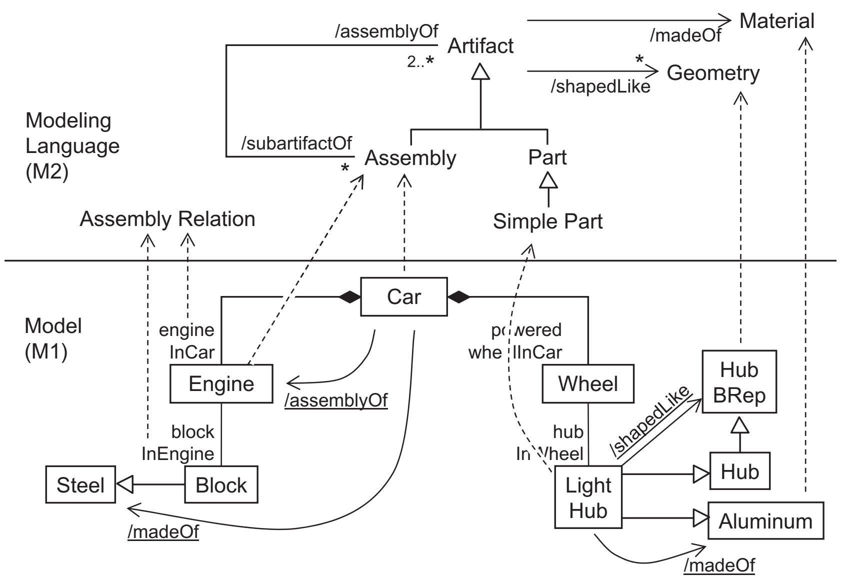

Fig. 17. Derived relations.

- /assemblyOf, /subartifactOf: Subartifacts in an assembly are derived from containment relations. For example, in Fig. 17 Car is an assembly of Engine and Wheel (only one derived 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 Fig. 17 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 Fig. 17 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.

5. Conclusion

This paper describes an example product modeling language combining the benefits of ontology with expanded capabilities in conventional product modeling languages to improve support for collaborative design exploration 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.

Disclaimer

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.

Acknowledgements

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

References

[1] N. Wognum, A. Trappey, PLM challenges, Advanced Engineering Informatics 22 (4) (2008) 419–420.
[2] R. Sriram, S. Szykman, D. Durham, Special issue on collaborative engineering, Guest Editorial, Journal of Computing and Information Science in Engineering 6 (2) (2006) 93–95.
[3] R. Sriram, Distributed and Integrated Collaborative Engineering Design, Sraven Publishers, 2002.
[4] S. Szykman, R. Sriram, W. Regli, 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) (2001) 3–11.
[5] W. Shen, J. Barthès, Special Issue on collaborative design and manufacturing, Advanced Engineering Informatics 22 (3) (2008) 281.
[6] International Organization for Standardization, Process specification language, ISO 18629, June 2006.
[7] P. Zave, M. Jackson, Four dark corners of requirements engineering, Association for Computing Machinery Transactions on Software Engineering and Methodology 6 (1) (1997) 1–30.
[8] Object Management Group, Unified Modeling Language: Superstructure. Available from: <http://doc.omg.org/formal/2010-05-05/>, May 2010.
[9] World Wide Web Consortium, OWL 2 Web Ontology Language Document Overview. Available from: <http://www.w3.org/TR/2009/REC-owl2-overview-20091027/>, October 2009.
[10] C. Bock, UML 2 composition model, Journal of Object Technology 3 (10) (2004) 47–73.
[11] J. Owen, STEP: An Introduction, Information Geometers, 1993.
[12] Object Management Group, Systems Modeling Language. Available from: <http://doc.omg.org/formal/2010-06-01/>, June 2010.
[13] S. Fenves, S. Foufou, C. Bock, R. Sriram, 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) (2008). 014501-1–014501-6.
[14] S. Rachuri, Y. Han, S. Foufou, S. Feng, U. Roy, W. Fujun, R. Sriram, K. Lyons, A model for capturing product assembly information, Journal of Computing and Information Science in Engineering 6 (1) (2006) 11–21.
[15] M. Stokes (Ed.), Managing Engineering Knowledge: MOKA Methodology for Knowledge Based Engineering Applications, Professional Engineering Publishing, 2001.
[16] P. Gu, K. Chan, Product modeling using STEP, Computer-Aided Design 27 (3) (1995) 163–179.
[17] International Organization for Standardization, Integrated generic resource: product structure configuration, ISO 10303-44, 2000.
[18] International Organization for Standardization, Integrated application resource: kinematic and geometric constraints for assembly models, ISO 10303-109, 2004.
[19] International Organization for Standardization, Application protocol: product life cycle support, ISO 10303-239, 2003.
[20] T. Liu, An Object-Oriented Assembly Applications Methodology for PDES/STEP based Mechanical Systems, Ph.D. thesis, The University of Iowa, 1992.
[21] X. Zha, H. Du, A PDES/STEP-based model and system for concurrent integrated design and assembly planning, Computer-Aided Design 34 (14) (2002) 1087–1110.
[22] Object Management Group, Model-Driven Architecture. Available from: <http://www.omg.org/mda/>, 2010.
[23] S. Fenves, Y. Choi, B. Gurumoorthy, G. Mocko, R. Sriram, Master Product Model for the Support of Tighter Design-Analysis Integration, U.S. National Institute of Standards and Technology Interagency Report 7004, May 2003.
[24] A. Biswas, S. Fenves, V. Shapiro, R. Sriram, Representation of heterogeneous material properties in the core product model, Engineering with Computers 24 (1) (2007) 43–58.
[25] C. Xu, S. Gupta, Z. Yao, M. Gruninger, R. Sriram, Towards computer-aided conceptual design of mechatronic devices with multiple interaction-states, in: Proceedings of the American Society of Mechanical Engineers Design Engineering Technical Conferences, 2005.
[26] F. Wang, S. Fenves, R. Sudarsan, R. Sriram, Towards modeling the evolution of product families, in: Proceedings of the American Society of Mechanical Engineers Design Engineering Technical Conferences, 2003.
[27] X. Zha, S. Fenves, R. Sriram, A feature-based approach to embedded system hardware and software co-design, in: Proceedings of the American Society of Mechanical Engineers Design Engineering Technical Conferences, 2005.
[28] X. Zha, S. Foufou, R. Sudarsan, R. Sriram, Analysis and evaluation for STEP-based electromechanical assemblies, Journal of Computing and Information Science in Engineering 6 (3) (2006) 276–287.
[29] X. Zha, R. Sriram, S. Gupta, Information and knowledge modeling for computer supported microelectromechanical systems design and development, in: Proceedings of the American Society of Mechanical Engineers Design Engineering Technical Conferences, 2005.
[30] F. Metzger, The challenge of capturing the semantics of STEP data models precisely, in: Proceedings of Workshop on Product Knowledge Sharing for Integrated Enterprises, Product Data Technology Advisory Group, ESPRIT Project 9049, 1996.
[31] N. Guarino, S. Borgo, C. Masolo, Logical modelling of product knowledge: towards a well-founded semantics for STEP, in: Proceedings of European Conference on Product Data Technology, 1997, pp. 183–190.
[32] M. Yoshioka, Y. Umeda, H. Takeda, Y. Shimomura, Y. Nomaguchi, T. Tomiyama, Physical concept ontology for the knowledge intensive engineering framework, Advanced Engineering Informatics 18 (2) (2004) 95–113.
[33] P. Borst, H. Akkermans, J. Top, Engineering ontologies, International Journal of Human–Computer Studies 46 (2–3) (1997) 365–406.
[34] D. Leal, ISO 15926 ‘Life Cycle Data for Process Plant’: an overview, Oil and Gas Science and Technology 60 (4) (2005) 629–638.
[35] E. Chan, K. Yu, A framework of ontology-enabled product knowledge management, International Journal of Product Development 4 (3–4) (2007) 241–254.
[36] D. Yang, M. Dong, R. Miao, Development of a product configuration system with an ontology-based approach, Computer-Aided Design 40 (8) (2008) 863–878.
[37] J. Nanda, T. Simpson, S. Kumara, S. Shooter, 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) (2006) 103–113.
[38] Y. Kitamura, A functional concept ontology and its application to automatic identification of functional structures, Advanced Engineering Informatics 16 (2) (2002) 45–163.

[39] J.X. Lin, M.S. Fox, T. Bilgic, A requirement ontology for engineering design, Concurrent Engineering Research and Applications 4 (3) (1996) 279–291.

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

[41] L. Patil, D. Dutta, R. Sriram, Ontology-based exchange of product data semantics, Institute of Electrical and Electronics Engineers Transactions on Automation Science and Engineering 2 (3) (2005) 213–255.

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

[43] A. Gehlert, E. Werner, Toward a formal research framework for ontological analyses, Advanced Engineering Informatics 21 (2) (2007) 119–131.

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

[45] J. Lee, H. Suh, Ontology-based multi-layered knowledge framework for product lifecycle management, Concurrent Engineering: Research and Applications 16 (4) (2008) 301–311.

[46] D. Brown, D. Leal, Chris. McMahon, R. Crossland, J. Devlukia, A Web-enabled virtual repository for supporting distributed automotive component development, Advanced Engineering Informatics 18 (3) (2004) 173–190.

[47] S. Shooter, W. Keirouz, S. Szykman, S. Fenves, A model for the flow of design information in product development, Engineering with Computers 16 (3–4) (2000) 178–194.

[48] M. Winston, R. Chaffin, D. Herrmann, A taxonomy of part–whole relations, Cognitive Science 11 (4) (1987) 417–444.

[49] J. Odell, Six different kinds of composition, Journal of Object-Oriented Programming 5 (8) (1994) 10–15.

[50] Object Management Grsoup, Unified Modeling Language: Infrastructure. Available from: <http://doc.omg.org/formal/2010-05-03/>, May 2010.

[51] R. Flatscher, Metamodeling in EIA/CDIF—meta-metamodel and metamodels, Association of Computing Machinery Transactions on Modeling and Computer Simulation 12 (4) (2002) 322–342.