Draft # DETC2009-87307

Formal Representation of Product Design Specifications for Validating Product Design

Alex Weissman and Satyandra K. Gupta
Department of Mechanical Engineering and Institute for Systems Research
University of Maryland
College Park

Xenia Fiorentini, Rachuri Sudarsan, and Ram Sriram
Manufacturing Engineering Laboratory
NIST

Abstract:
As collaborative efforts in electro-mechanical design have scaled to large, distributed groups working for years on a single product, an increasingly large gulf has developed between the original stated goals of the project and the final design solution. It has thus become necessary to validate the final design solution against a set of requirements to ensure that these goals have, in fact, been met. This process has become tedious for complex products with hundreds of design aspects and requirements. By formalizing the representation of requirements and the design solution, tools can be developed to a large extent automatically perform this validation. In this paper, we propose a formal approach for relating product requirements to the design solution. First, we present a formal model for representing product requirements. Then, we introduce the Core Product Model (CPM) and the Open Assembly Model (OAM) for representing the design solution. Finally, we link these models formally and provide an example with an actual consumer device.

1. Introduction

The process of designing an electro-mechanical device often begins with generating a product design specification (PDS) document. This document describes the intended function of the device being designed, and the environment in which it will be used. It also specifies certain high-level requirements related to global constraints such as safety, shipping, and manufacturing. In this paper, PDS document means requirements document. A properly written PDS is solution neutral and does not specify design details; i.e., it describes what the product will do and not how it will do it. This is crucial to ensure that the creative control of the designers is not stifled, and that changes to the design details will not necessarily require a change to the PDS. Furthermore, with regard to communication within large design teams, the PDS serves to ensure that every member of the team is working towards the same overall goals [1].

Once the PDS has been completed, the engineering design team can begin work on a solution which meets all of the requirements and specifications described in the PDS. To create a detailed model of the product, concepts are generated, refined, and elaborated until the final design is determined. This model includes information about the components and assembly and how they function together to realize the stipulations provided by the PDS.

Currently, most PDS are written in English, with little standardization. PDS written in natural language were sufficient when design teams were small and all members were in constant

contact with one another. However, as the complexity of products has increased, it has necessitated much larger teams composed of individuals from a wide variety of backgrounds, who may be both physically and temporally distributed well apart from one another. Information exchange among design team members can easily degrade under these circumstances and can easily lead to the following two problems. First, a statement written as a part of a PDS might lead to two different interpretations, and consequently to two conflicting design choices, by two different team members. Second, levels of detail in a PDS may vary significantly based on the experience and style of the individual writing it. The PDS may thus be incomplete in one extreme, or overly verbose in the other. Such PDS documents may lead to confusion during design, and may require significant revisions that might delay the design process.

Because most design processes are iterative in nature, both the PDS and product models may go through many revisions. The product model evolves over time as better alternatives are found and incorporated into the design. Although the PDS is intended to be as independent from the design solution as possible, there are instances in which substantial changes to the design process itself may necessitate a change in the PDS. For example, initially the PDS may be too demanding and the design process will reveal that they need to be altered to successfully realize a product. In this case, we say that the specifications are over-constrained. It is also possible for the specifications to be under-constrained, in which case the PDS is not sufficiently rigorous to provide the desired information. For this reason, it is necessary that the PDS record a history of the changes made along with detailed explanations of these changes. This history is important for keeping every design team member well apprised of the changes to ensure that they are working with the most up-to-date version of the PDS.

Currently, design teams consult the PDS during the design process. However, as the design evolves no formal approach is used to ensure that various design elements support the requirements in the PDS document. Hence discrepancies between the PDS and product model are discovered during the design process. In some situations, design features have been added that do not support any stated requirements (e.g., very high surface finish in the interior of the housing with no requirement calling for it). In some situations there are design requirements that are not satisfied by the design (e.g., enclosure needed to provide safety in an electric saw was missing). Even when the design meets all the requirements, after a few months no one can recall why a particular feature was added. A proactive approach to linking the PDS and product model can identify these problems much earlier in the design process.

Ideally, when validating a design solution against a set of requirements, one should be able to trace every design element to a requirement [2]. In order to establish this accountability, every decision made during the design process must have some relationship to the goals originally outlined in the PDS. Accountability can be formally represented as tracing statements (links) which map one or more elements of the design solution to one or more requirements. In our proposed approach, after an element has been created in the product model, it should be linked to the relevant requirements. A successful design process will generate solutions in which all elements in the design can be traced to the PDS. In other words, the product design does not contain any element that is not needed to meet the requirements in the PDS. On the other hand, all aspects of the PDS must be addressed by some design feature. These links between the PDS and the product model provide the design rationale and also ensure that the design is complete (i.e., does not have any missing features) and is not redundant (i.e., does not contain any unnecessary features). This ensures a low design cost and also ensures that designs are safe and desirable to customers.

In order to represent links between the PDS and product models formally, the PDS and product models must both have formal representations. Currently, there is no formal and convenient framework to represent a PDS. To address this need, we have described a framework for representing PDS in Section 3. As a byproduct, our formalism will also enable searching the requirements based on their semantics. Fortunately, significant progress has been made in formally representing product models. NIST's Core Product Model (CPM) and Open Assembly Model (OAM) are comprehensive models for representing the relevant aspects of the design solution, and are summarized in Section 4. This model will serve as our means for linking the PDS to the design solution. Section 5 provides a formal model for the connections between elements of PDS and elements of product models at an appropriate level of granularity. These links describe what roles are played by various elements of the design solution in meeting the requirements. Figure 1 represents these relationships between the PDS and the design solution.

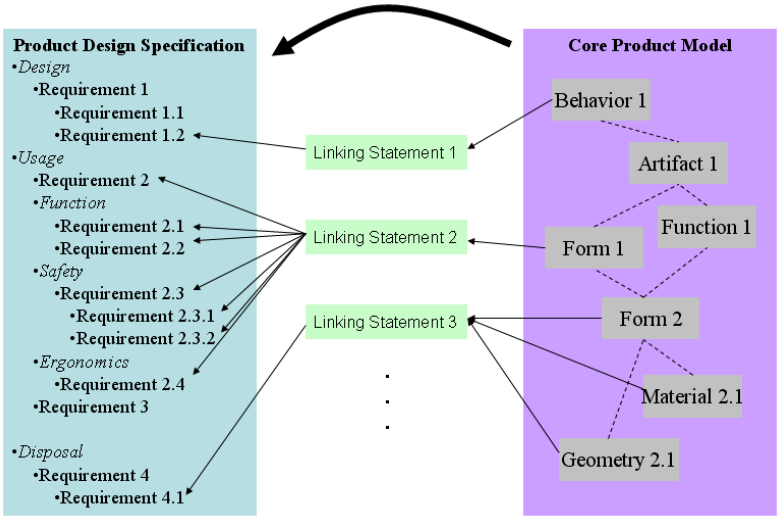


Figure 1: Overview of the project

A case study is also presented during the paper both to demonstrate an application and to test the viability of our model. The overall specification for our case study, the Automatic Pot Stirrer, is the design of a new system to automatically stir the food contained and cooked in a pot. The implementation of this case study includes: 1) using our framework to represent the PDS for the system, 2) using CPM and OAM to model a design solution for the Automatic Pot Stirrer, and 3) using the linking criteria to connect the product design to the PDS

2. Previous works

In comparison with software engineering, there have been limited efforts towards any formal modeling of requirement statements in the field of electro-mechanical products. Models for software requirements are well developed, but these models are extremely specialized and cannot easily be ported to the electro-mechanical domain [3] [4].

Currently, most work on requirements formalization for electro-mechanical products focuses on requirements management [5] [6]. Requirements management techniques seek to first analyze the processes of requirements generation and propagation and then provide constraints and rules for these processes with the goal of guaranteeing ownership, prioritization, agreement and communication of the requirements. However, they do not attempt to provide constraints and rules for the actual PDS document. To date, most efforts have largely been carried out within the field of system engineering: the requirements are generated at the level of product system and subsequently refined for product design. In this section we analyze these efforts.

The EIA-632 [7] standard describes the two fundamental processes for system engineering: requirements definition and solution definition. In the requirements definition process, system technical requirements are derived from stakeholders requirements. In the solution definition process, a logical and physical representation of the system solution is presented and system technical requirements are refined at the subsystems level. The EIA-632 defines requirements as "Something that governs what, how well, and under what conditions a product will achieve a given purpose" and classifies them into operational, performance and enabling requirements. Operational requirements focus on the goals, objectives, and general desired capabilities of the system without indicating how the system can be implemented. Performance requirements focus on how well the system is suited to perform a function, along with the conditions under which the function is performed. Enabling requirements include technology constraints, product design constraints and requirements associated with the processes of the product lifecycle. The EIA-632 also lists the desirable characteristics of each requirement statement: clarity, correctness, feasibility, focus, implementability, modifiability, certainty, singularity, testability and verifiability. This standard provides a definition for each of these characteristics but it does not suggest any means to achieve them.

Similarly, the IEEE 15288 [8] standard presents a framework for describing the system lifecycle processes. This standard recommends a process for requirements derivation that is similar to the one of IEA-632. First, stakeholders requirements are translated into system requirements. Second, the system is characterized with its functions, the performance of its functions and the conditions under which the functions will be performed. Third, the system architecture is developed and the subsystems requirements are specified. Once again, the IEEE 15288 lists the characteristics required for each requirement statement but it does not describe how to achieve them.

The landscape of the standards within the field of systems engineering includes not only process standards (such as IEA-632 and IEEE 15288) but also modeling standards. SysML [9] is a standard language that provides a means to capture the system modeling information, which includes system requirements. The requirements categories proposed in SysML closely follow the other standards: requirements are divided into functional, interface, performance, physical and design constraint categories. In SysML, the requirements are expressed in plain English and connected to the system model in a requirements diagram. In this diagram, requirements, design elements and test cases are connected to each other through pre-defined properties (derive, satisfy, verify, refine, trace, copy, and contain.) SysML provides only a text-based definition for each of these properties; thus, it does not propose a computer-interpretable semantic for them. Although SysML allows tracing of relationships between requirements and design elements, it does not enable an efficient keyword-based search of them; the usage of plain English to express requirements remains one of the major drawbacks of SysML.

The AeroSpace and Defence Industries Association of Europe proposes in the ASD-STE100 [10] standard to adopt a controlled form of English to write technical documentation: Simplified Technical English. ASD-STE100 was originally developed to avoid lexical ambiguity and sentence complexity in aircraft maintenance documents. Today, it is used in many other industries and for all kinds of technical documentation including PDS. The Simplified Technical English is composed of a dictionary of controlled vocabulary and a set of writing rules.

In the STE dictionary, all synonyms words are grouped together and only one word can be used to express them. Moreover, for each word a unique and precise definition is given, together with the part of the speech it can represent. As an example, according to the

specification: i) the words “dull” and “faint” should be avoided and replaced with the word “dim”; ii) the word “dim” can be used only as an adjective and its meaning is “not bright”, rather than “hopeless”; iii) using the word “dim” as a verb is prohibited: the verb “decrease” should be used instead. The usage of this controlled vocabulary is regulated by writing rules. The goal of these rules is to help readers to understand a technical document. The writing rules impose, for example, usage of the active form of verbs when possible, prohibition of noun clusters of more than three nouns, a limitation of at most 25 words in each sentence, and expression of only one topic in each sentence. The writing rules also provide some indications on how to integrate proprietary taxonomies (called Technical Names and Technical Verbs) into the controlled vocabulary. For this purpose, they list all the technical categories to which the proprietary words should belong.

Some tools and software have been developed to assist authors to write technical documents in simplified English conforming to ASD-STE100. Unfortunately, these tools and software can only check simple rules, such as sentence lengths and noun clusters, but they can not check other rules, such as number of topics in each sentence and content meaning. As declared by the ASD association, even with the help of these tools, “training is the first essential step for a technical author to be able to apply ASD-STE100 correctly”[11]. To adopt this standard, companies are thus required to devote a considerable amount of time and resources to training.

The ASD-STE100 standard imposes a standard list of words with predefined meanings for using in requirement statements while leaving the user free to choose the statement structure. In our approach, instead, we give the user flexibility in word choice while allowing him/her to tag those words using elements from a standard taxonomy. We then suggest templates for structuring the sentences. In this way, we develop a formal model which can be adopted quickly and with a minimum of additional author training.

3. Formalizing the Product Design Specification

A PDS is a document made up of a series of discrete pieces of information, known as requirement statements, which outline the design goals. A **requirement statement** is a complete English sentence that expresses a rule with which the final design solution must comply. For example, “The product must be able to function underwater” is a requirement statement. In addition to intended functionality, a PDS will also include statements which describe information on how the device will be marketed, produced, and distributed, how the device should be maintained and disposed of, and the standards and regulations that apply to each of these areas. According to Magrab [12], “the PDS contains all the facts relating to the product’s outcome. It is a statement of what the product has to do and is the fundamental control mechanism and basic reference source for the entire product development activity”. All decisions made during the design process must be carefully made with respect to the PDS.

The traditional PDS is written in natural English as a series of paragraphs or sections logically organized by the various facets of the product’s performance and lifecycle. Sometimes, tables are included as well to list design parameters and map them to values, ranges of values, or qualitative statements.

An abridged example of a traditional PDS is given in Figure 2.

Performance • Tape inside corners. • Tape joints in any orientation. • Tape joints without leaking joint compound. • Taped joints will require no additional smoothing. • Tape will not break prematurely. Shipping • Will be shock, vibration, and weather resistant for shipping by any means: vibration environment from 4-33Hz at 0.06 inch (1.5 mm) amplitude; shock environment simulated with an 8 foot (2.4 m) drop test. Insensitive to temperature and humidity. • Will be packaged in a rectangular cardboard box that can be stacked up to 8 ft (2.5 m). Aesthetics • Product will have a durable finish. • Colors will be green and black. • Product will convey ruggedness.	Maintenance • All fasteners will be standard. • All components subject to wear to be inexpensively and easily replaceable with standard tolls and no special skills. • Design will be modular for easy repair and cleaning. • Joint compound removed after each day's work. • No or very few lubrication points. Disposal and Recycling • Product will not contain any environmentally hazardous materials. • Easily disassembled for component recycling and reuse. Product Environment • Product must operate in the following environment: – Temperatures between 40 and 120°F (4 and 50°C) – Atmospheric pressure from sea level to 7,000 ft (2.1 km) – Relative humidity to 100% – Concentration of solid or liquid particulate smaller than 500μm.

Figure 2: Sample requirements statements from PDS for Drywall Taping System (Magrab, 1997).

This example sets down the requirements for an automated device that simultaneously applies tape and joint compound to drywall seams. It then gives a set of requirement statements, categorized under the author's own headings. Although this document is fairly well-organized, and uses a proper level of detail, it is still not sufficient for automated processing. For example, the terminology has not been strictly formalized, which could pose challenges in archiving requirement statements in a database and retrieving those statements via a keyword search. Even if the author were to apply a systematic method of organizing the statements, and possibly even a formalized technical vocabulary, there is no guarantee that the author's system will be consistently applied from PDS to PDS or even between versions edited by different authors.

Ideally, a formal model for requirements should minimize ambiguity and information loss while maximizing expressiveness. Furthermore, it is desirable that the model be flexible, customizable and extensible.

3.1 Overview

We have identified four main goals for requirements model:

- ***Flexibility***: Our model should not impose excessive constraints which would require the user to conform to a rigid and unnatural way of writing. It should simply propose a toolbox for categorizing traditionally written requirement statements, and attaching metadata to words or phrases in those statements (see Figure 3).
- ***Customizability***: Our model should be modular to allow users to discard some components of the model and replace them with their own.
- ***Extendibility***: Similarly, users should be able to add their own vocabulary to the model in a controlled fashion.
- ***Formality***: Statements generated using our formal model should be processed in a reliable and automated way. This is crucial for developing computerized tools such as search engines and design validation software.

Formalization of a product design specification begins with determining the way in which the name of the product will be identified. The name of the product should be consistent within the PDS, as well as in the PDS of the product's components and accessories. This is accomplished through the use of a device taxonomy, which creates a hierarchical association among similar devices. The names of devices are classified according to some criteria, such as their primary working principle. The taxonomy is represented as a tree, and devices which are more similar are placed closer together on the tree. This taxonomy must be specific to a very small subset of engineering design – it would be virtually impossible to construct a complete taxonomy of all possible engineering devices. As such, individual device taxonomies are to be constructed by each company or organization to meet the needs of their industry. However, different device taxonomies from different authors may be aggregated or harmonized as the model becomes more widely adopted, for example in the form of a public Internet database [13]. Once a taxonomy has been built for a class of devices, the elements of the taxonomy can be used to tag references to the product in every requirement statement [14].

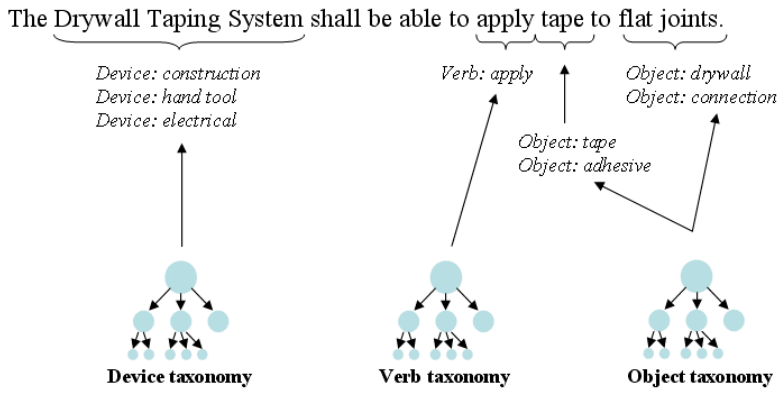
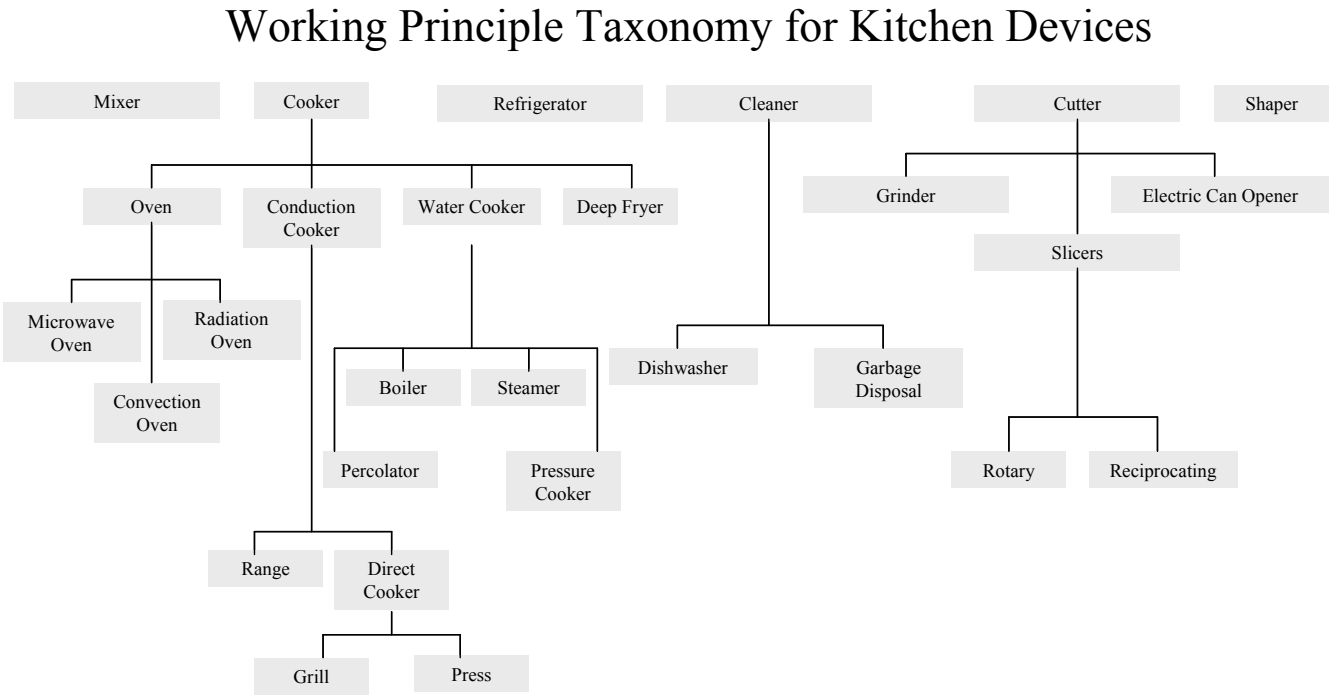


Figure 3: Tagging components of a requirement statement with meta-data drawn from standard taxonomies

Sample device taxonomy for kitchen appliances, based on their primary working principles, is shown in Figure 4. The working principle criterion was chosen because most appliances have only one or two primary working principles (e.g. cooking and mixing), which thus minimizes the number of tags necessary to categorize a given device name. As an example of how this taxonomy can be useful, consider the device "Automatic Pot Stirrer", an appliance which is meant to simultaneously stir and cook food. Virtually every requirement statement will contain a reference to this device, so it would be useful to associate some standard terms with the device in each statement. In this case, appropriate terms would be "Mixer" and "Cooker → Water Cooker" or "Cooker → Conduction Cooker". This connection is extremely useful for comparing the device with other products that might have a similar purpose but a different name. No matter how the specific instance of the product is labeled (e.g. "Automatic Pot Stirrer" vs. "Mix-n-Cook"), it will always be linked to one or more of these labels in the device taxonomy.

If the device taxonomy is built in such a way that each level of the tree represents a more specific type of device than its parent, then the notion of requirement inheritance can be introduced. Each type of device can inherit some or all of the requirements associated with its parent on the taxonomy tree. For example, in the kitchen appliance taxonomy, the device "Slicer" might have a safety requirement "must not allow operator to directly access the blade during use." This requirement clearly applies to every device which is a child of the device

"Slicer." An approach based on inheritance makes it convenient to reuse requirements defined for the product class.



Devices may map to:

- A single keyword
  - Coffee Maker (Cooker->Water Cooker->Percolator)
  - Electric Knife (Cutter->Slicer->Reciprocating)
- Multiple keywords
  - Blender (Mixer, Cutter->Grinder)
  - Bread Machine (Mixer, Shaper, Cooker->Oven->Radiation Oven)
  - Waffle Iron (Shaper, Cooker->Conduction Cooker->Direct Cooker->Press)

Figure 4: A sample taxonomy of kitchen appliances based on primary working principle.

In Section 3.2, we propose two different schemes which will be applied to each requirement statement within a PDS. The first scheme (Requirement Categories), places every statement in a particular phase of the product's design and deployment lifecycle. This mode of organization is useful for a development team member who is only concerned with a specific aspect of the product's life cycle. For example, a marketing manager may want to reference the aspects of the PDS that are relevant to marketing the product, without seeing the details on the manufacturing constraints. The second scheme (Requirement Definition Templates) attempts to capture the syntactic structure of each requirements statement, based upon their component parts of speech. In this scheme, phrases within a requirements statement are tagged according to their role in the statement, such as the subject phrase, verb phrase, or an object or modifying phrase. This allows each statement to be automatically parsed and broken down into logical units of data that can be indexed and compared. Once the parts of speech and their organization within a statement have been identified, tools can be developed which match statements based on structural patterns rather than on specific words. In the kitchen appliance example, we might want to determine which requirements refer the device performing some action on a piece of

food. Rather than simply querying requirements which refer the device and the piece of food, we might look only at requirements which contain the pattern "subject-phrase, verb-phrase, object-phrase", where the subject-phrase contains a reference to the device, and the object phrase contains a reference to the piece of food.

Our model also includes a specialized taxonomy of verbs, attributes, units, and use-environment objects, which can be applied as tags on requirements statements. The purpose is similar to that of the device taxonomy; it allows non-standard vocabulary to be used by the PDS author, and yet remain linked with a standard lexicon. These taxonomies are described in Section 3.3. Finally, in Section 3.4 we test our model with several PDSs taken from external sources, and demonstrate that our model has fulfilled the four primary desired characteristics as outlined above.

3.2 Requirement Categories and Definition Templates

The first scheme of our formal model for requirement statements allows the author to assign each statement to one or more categories based on its content. The hierarchy of categories has been constructed based on a state-transition diagram of the product lifecycle.

- Conduct Market Research
  - Patent Infringement
  - Public Relations
  - Competitive positioning
  - Cost
  - Timeline
  - Production volume
- Design
- Procure
- Manufacture/Assembly
- Test/Inspect
- Ship/Transport
- Warehouse
- Display
- Install
- Use/Operate
  - Function
    - Motion
    - Structure
    - Energy
    - Interface
  - Environmental Conditions
  - Safety
  - Ergonomics
    - Ease of use
    - Accessibility
  - Reliability
  - Aesthetics
- Store
- Service/Repair
- Upgrade
- Dismantle
- Recycle
- Dispose

Figure 5: Product lifecycle categories for requirement statements.

This decision was made for the sake of completeness: a product only exists within its lifecycle, and therefore any requirements related to the product can be mapped to at least one stage of its lifecycle. Many of the subcategory names, such as "Geometry" and "Kinematics" have been taken from the categories given in Pahl and Beitz [15]. Other categorization schemes have been suggested, but none of these cover the entire lifecycle [16].

Notice that requirement categories are not simply represented as a list, but rather a hierarchical tree structure. This is useful for organizing requirements based on how closely they are related to one another. For example, according to the hierarchy given in Figure 5, a statement tagged with the category "Structure" is more closely related to a statement tagged with "Interface" than one tagged with "Cost". Authors may also extend the requirement categories in Figure 5 with their own sub-categories when a finer distinction is required.

The second scheme deals with establishing a connection between the syntactic form of a statement, and its meaning. In the English language, it is evident that there is some disjunction between a piece of information, and the manner in which it is represented. Various controlled languages, such as Simplified Technical English, have been implemented, but none of these give a level of expressiveness suitable for daily use in the broader field of mechanical design.

In order to help establish a stricter correspondence between the syntactic form of a statement and its meaning, we introduce a method of tagging requirement statements. We have called these Requirement Definition Templates (RDT's). Each template specifies a syntactical part of a sentence such as the subject, verb, or object [17]. These parts of speech are in turn composed of lexical primitives: verbs, nouns, adjectives, prepositions, and adverbs, conjunctions, interjections, and articles. Many of these lexical primitives can be tagged with keywords from the taxonomies presented in Section 3.3.

Because RDTs are designed to capture the essential elements of the statement and their relationships, statements can be parsed in a consistent manner. Minor variations in the wording of each statement may exist, such as with the use of prepositions, but none of these variations are able to dramatically change the way in which the statement will be parsed. We now define the parts of speech which are defined using an RDT. It is assumed that the reader is already familiar with these primitives and so they will not be discussed here.

The **subject-phrase** is used to identify the subject of the statement, which is usually the first phrase of the sentence. The subject can contain entities, such as the device, components, or use-environment entities, attributes, and verbs (in nominal form). "The pot", "The gear mechanism", "The color of the surface", and "Use of a DC power source" are all valid subject phrases.

A **verb-phrase** consists of a core verb, plus any auxiliary verbs, negations, and infinitive or participle words. For example, "operate", "not be operated" "be able to operate", "be able to be operated", and "be able to continue operating" are all valid verb phrases. Since every requirement statement – every statement, for that matter – must contain a verb, it makes sense to classify statements based on the verb phrases that they use.

Object phrases include **what-phrases** and **who-phrases**. Like subject phrases, object phrases can contain entities, components, use-environment entities, attributes, quantities, and nominally formed verbs. The distinction between what-phrase and who-phrase RDTs is determined by whether the object is inanimate or animate. Object phrases are always the object of a transitive action in the verb phrase.

Modifying phrases are used to augment or specify more details for other phrases in a statement. A modifying phrase can be a **when-phrase**, **where-phrase**, **how-much-phrase**, **what-kind-phrase**, **what-state-phrase, how-phrase**, or **why-phrase**,. When-phrases can be used to specify a definite or indefinite period of time, or absolute date such as "2-3 years", "after January 14th", or "in approximately 18 seconds". When-phrases apply globally to the entire statement. Where-phrases specify a definite or indefinite location such as "next to the electrical outlet" or "10 cm from the inlet valve". Like when-phrases, they reference the entire statement.

How-much phrases are wrapper phrases for quantities; in other words, they specify an amount or range of quantities using exactly one quantity element such as "between 10 and 20 kilojoules" or "8 pieces". Unlike when-phrases and where-phrases, how-much phrases are not global modifiers. Instead, each quantity element within this phrase can optionally have a reference to an attribute or entity.

What-kind phrases are used to create an adjectival element that modifies an entity in the statement. In the statement "Units will not be sold that do not pass quality assurance," "that do not pass quality assurance" is a valid what-kind phrase that references the entity "units". Like how-much phrases, what-kind phrases can have a reference to an attribute or entity.

What-state phrases indicate the passive state or status of an entity in the requirement statement. "Under normal loads" and "without interference" are examples of what-state phrases. Once again, they can reference attributes or entities in the statement.

Why-phrases are used to characterize the purpose of an entity or action within a statement. For example "in order to provide power" and "to simulate five years service" are valid why-phrases. Unlike other modifying phrases, why-phrases can reference verbs and even quantities in addition to attributes and entities.

How-phrases, such as "such that safety regulations are upheld", indicate the manner in which an action is to be performed, or a means by which the action is to be performed. They are similar to why-phrases but emphasize method rather than purpose. Therefore, how-phrases can only reference verb elements in a statement since method only has meaning in the context of an action.

By assembling these grammatical units, a requirement statement can be formed. In other words, Requirement Definition Templates (RDTs) allow for requirement statements to be hierarchically encoded as a grammatical derivation tree. This derivation tree can then be stored in an XML format.

In accordance with the principles of good requirements generation, "run-on" requirement statements should be avoided – an ideal requirement should be limited to a single aspect of the design [18]. Any statement which violates this granularity condition should be decomposed into two separate statements. As a simple case, consider the statement "The motor shall supply 50Nm of torque and produce no more than 40dB of sound." This statement can be decomposed into "The motor shall supply 50Nm of torque," and "The motor shall produce no more than 40dB of sound" without any semantic loss. Granularity of statements is especially important when linking elements of the product model to requirements; it could otherwise be unclear as to which part of the requirement a design feature is related.

Once a requirements statement has been rewritten and various grammatical units have been tagged, additional metadata can be applied to each grammatical unit. For example, verb phrases can be tagged as transitive or intransitive, positive or negative, or potential (i.e. "able to"). Quantities can be marked as cardinal or ordinal (e.g. "2" vs "2^{nd}"), and entities can be marked as singular or plural. By default, verbs are positive and transitive, quantities are cardinal, and entities are singular. These tags help to provide more refined details about the role of each phrase in the statement.

Of course, it is impractical to expect the requirements author to explicitly identify and select grammatical tags for every statement. This would be counterproductive to a major goal of this exposition, which is to provide as "natural" a model as possible for writing requirement statements. To alleviate the tedium of tagging every single phrase and element of every statement, a tool could be developed that predicts the appropriate RDT and taxonomic tags, and prompts the author for feedback when necessary.

3.3 Taxonomies of Statement Words

Just as we tag each phrase in a requirement statement, individual words in a phrase can be tagged as well to further improve statement matching. Authors can use their own terminology, and then map it to one or more words in a standard lexicon in which each word has a precise, predetermined, and unique definition. Therefore, the meaning of each statement can be conveyed in the wording of the statement without fear of ambiguity in the form of synonyms and

homographs. This lexicon should be represented as a hierarchy or taxonomy, and it should be common to all requirements documents within a particular domain. In addition to the sample device taxonomy discussed in Section 3.1, sample taxonomies for use-environment objects, attributes, units, and verbs have been developed and are presented here. Words which have little value in keyword searches, such as prepositions and adverbs, do not have associated taxonomies.

For each type of device, such as kitchen appliances, there is an associated use-environment. Kitchen appliances are used primarily in the kitchen, automobiles are used on roadways, and battle tanks are used in combat zones. It therefore makes sense to form a standard taxonomy of objects within the use-environment to supplement each device taxonomy. Just as with device taxonomies, object taxonomies can be aggregated into larger hierarchies. An example of object taxonomy for use-environment objects is given in Figure 6.

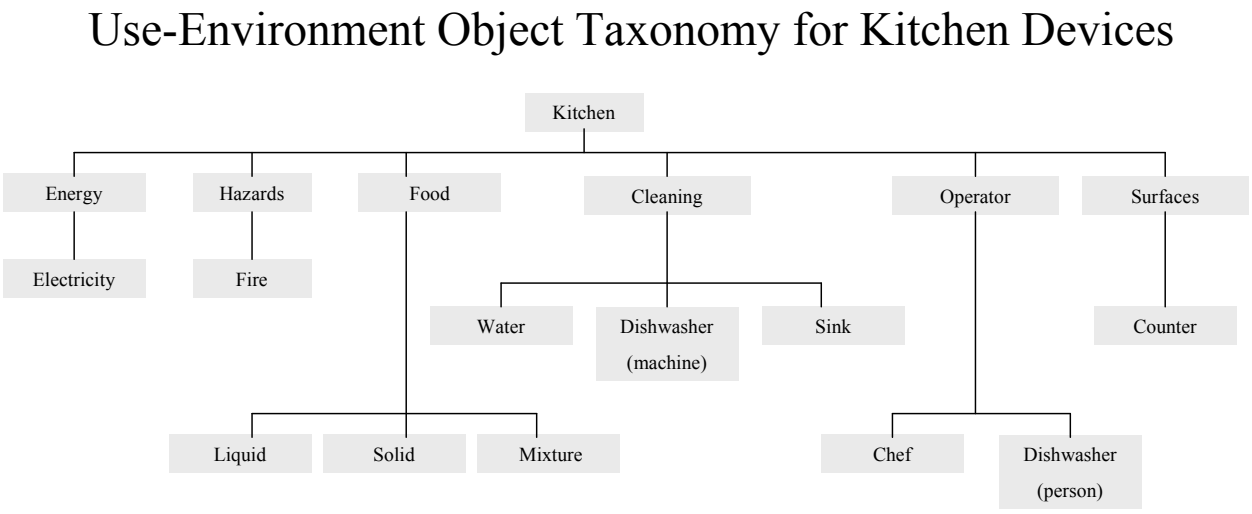


Figure 6: A sample taxonomy of objects typically encountered by kitchen appliances.

Attributes, also known as properties, are defined to be global to all devices and objects. Although many attributes (e.g. Reynold's number) are relevant to only a handful of devices, other attributes such as mass, length, and cost are applicable to virtually any devices. Therefore for the purpose of providing a simple, consistent lexicon of attributes, we declare every attribute global as applied to all devices and objects. Based on the 'property' taxonomy in Ira Golden's thesis [13], and the material library at the websites CustomPartNet [19] and MatWeb [20], we have constructed an exemplar hierarchy of device and object attributes. In the attribute taxonomy, attributes are intuitively categorized based on field of application. For example, the properties of "Reynold's number" and "Viscosity" are both relevant to devices that work with a fluid medium. Thus "Material->Fluidic" is the category of the attribute taxonomy that contains these two properties. This is shown in Appendix B.

In order to quantify attributes, for example in "how-much phrases", it is necessary to declare units such as millimeters, dollars, or pounds. Therefore, a taxonomy of units which parallels the structure of the attributes taxonomy is useful (see Figure 7).

Based on the hierarchy of the attributes taxonomy, a custom units taxonomy can be constructed in which the attributes at the leaf nodes of the taxonomy are replaced with the units used in the design. An optional expression can be associated with the author's units to convert to SI units. This allows automated tools to perform comparisons in different units for a given attribute [21].

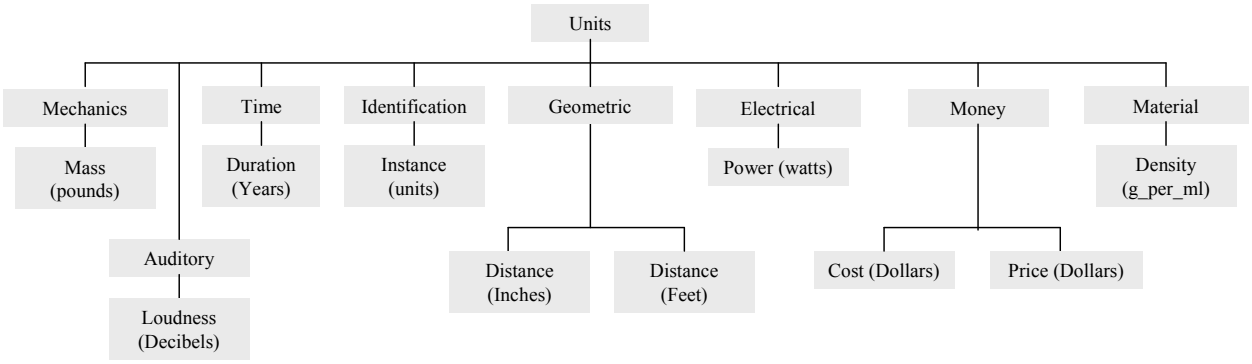


Figure 7: Taxonomy of units associated with subset of attributes

As we mentioned before, every requirement statement has a verb phrase, and every verb phrase has one core verb. Therefore, a taxonomy of core verbs is crucial to this model. Like attributes, core verbs are global to all requirements documents. There are very few verbs that are unique to a particular field of design, and those that are can generally be mapped to a more general verb in the global taxonomy. Our proposed core verb taxonomy is taken directly from Ira Golden's thesis [13], with the addition of the copula "to be." It contains a total of 827 verbs in a two-level tree, and was developed over the course of a year via extensive study of actual requirement documents. In this taxonomy, generalized hypernyms are chosen for the first level of verbs. Below each hypernym, several hyponyms, or more specific synonyms, are aggregated at the second level. This aggregation principle allows for advanced search queries to be performed on requirement statements; queries that contain a particular verb would find requirements that contain that verb's synonyms. For brevity's sake, it will not be shown here.

3.4 Validation

To accurately assess the efficacy of our proposed PDS model, we have taken several examples of real requirements documents and tagged each one using RDTs, statement categories, and lexical taxonomies. We then measured information loss by counting the number of discrete concepts in the original requirements document and subtracting the total number of discrete concepts in the formalized version. In this case, a discrete concept refers to a piece of information that can be represented by a single noun phrase, verb phrase, object phrase, or modifying phrase. Studied examples include the Drywall Taping System [12], a Sliding Attachment for Flatbed Trucks by a member of the Capstone Design Class at the University of Maryland, and the Carton Assembly Machine by Pahl and Beitz [15].

Almost (>95%) all statements in these documents were easily and meaningfully encoded using our proposed scheme. However, because our model is designed to capture the interactions between the device and its environment, data outside of this scope could not be easily encoded. This includes information that does not qualify as a requirement such as comments "Project conference minutes 16/70" [15], and information that lies outside of the use-environment such as

"Customer will be asked to complete a questionnaire" [12]. We do not believe that it constitutes any significant loss by excluding statements such as these from our model.

These results support our claims of flexibility, customizability, extendibility, and formality as outlined in Section 3.1. Flexibility has been demonstrated through the broad range of statements which can be expressed using RDTs and metadata taxonomies. Our model also provides for modular customizability, insofar as that a user can choose whether or not to use a particular element of the model at the expense of reduced formality and compatibility. It also allows for the user to scale the system by extending these elements, provided that the extension does not contradict any existing elements of our model. This is crucial towards ensuring backwards compatibility of a PDS with the model. Finally, our model is formal in the sense that every statement, every phrase, and most words with significance to the meaning of the statement, can be tagged with one or more pieces of metadata.

3.5 Formalized PDS for the Automatic Pot Stirrer

To show the applicability, benefits and limitations of our model, we take as a case study the design of an Automatic Pot Stirrer. This case study is loosely adapted from a students' project during the Capstone Design Class at the University of Maryland. In this project, using a set of requirements specified by the teacher, the students had to explore different design solutions with the help of engineering tools and techniques. The delivery of the project included a detailed description of the finally adopted design solution, along with an explanation of the motivation for choosing that solution.

We believe this case study to be appropriate for the purpose of this paper for its simplicity and completeness. Our case study represents on a small scale a real enterprise-wide design process; a set of requirements is given as input in the form of a document written in natural English, and a design solution is delivered as output in the form of CAD files and technical documents. Because this is a relatively simple device, it was possible for us to retroactively link the design solution to the requirements, and thus accomplish the goal that we set for this paper.

We have already provided taxonomies for the kitchen appliance device category (Figure 4), the corresponding use-environment objects (Figure 6), and the verbs, attributes (Appendix A), and units (Figure 7). The requirement statements are constructed so as to take advantage of the tagging capabilities of RDTs, and categorized according to their relevance within the product's lifecycle.

The full set of requirements is 14 pages long and will not be shown here. Instead, we present a representative set that exemplifies the diverse range of statements that can be meaningfully tagged using RDTs and taxonomies.

Sample Requirements from PDS for Automatic Pot Stirrer

R10.1.1.X: Use/Operate→ Function → Motion	R10.4.2.X: Use/Operate→ Ergonomics→ Accessibility
R10.1.1.1: The pot shall be able to automatically stir the food after activation. *Subject phrase: "The pot"* *Entity: "pot" (device::kitchen_appliance::mixer, device::kitchen_appliance::cooker::conduction_cooker)* *Verb phrase: "shall be able to automatically stir"* *Verb: "stir" (verb::mix::stir, potential)* *What-phrase: "the food"* *Entity: "the food" (use_environment::food::mixture)*	R10.4.2.1: The pot shall be able to be operated by one person. *Subject phrase: "The pot"* *Entity: "pot" (device::kitchen_appliance::mixer, device::kitchen_appliance::cooker::conduction_cooker)* *Verb phrase: "shall be able to be operated"* *Verb: "operate" (attribute::start::operate, passive)* *Who-phrase: "by one person"* *Entity: "person" (use_environment::operator)*

What-state-phrase: "after activation" *Reference: "pot"* R10.1.3.X: Use/Operate→ Function → Energy R10.1.3.1: The power for the pot shall be transmitted through an electrical cable. *Subject phrase: "The power for the pot"* *Attribute: "power" (attribute::material::electrical::power)* *Reference: "pot"* *Entity: "pot" (device::kitchen_appliance::mixer, device::kitchen_appliance::cooker::conduction_cooker)* *Verb phrase: "shall be transmitted"* *Verb: "transmitted" (verb::move::transmit, intransitive)* *How-phrase: "through an electrical cable"* *Entity: "electrical cable" (use_environment::cable)* *Reference: "transmitted"*	*Quantity: "one" (1, units::instances)* *Reference: "person"* R10.6.X: Use/Operate→ Aesthetics R10.6.1: The pot shall not absorb odors. *Subject phrase: "The pot"* *Entity: "pot" (device::kitchen_appliance::mixer, device::kitchen_appliance::cooker::conduction_cooker)* *Verb phrase: "absorb"* *Verb: "absorb" (verb::acquire::absorb, negative)* *What-phrase: "odors"* *Attribute: "odor" (attribute::material::chemical::odor)* R10.6.2: The acoustic noise of the pot shall be less than 110 dBA. *Subject phrase: "The acoustic noise of the pot"* *Attribute: "acoustic noise" (attribute::auditory::loudness)* *Reference: "pot"* *Entity: "pot" (device::kitchen_appliance::mixer, device::kitchen_appliance::cooker::conduction_cooker)* *Verb phrase: "shall be"* *Verb: "be" (verb::copula)* *How-much-phrase "less than 110 dBA"* *Quantity: "less than 110 dBA" (0-110, units::auditory::decibel)* *Reference: "acoustic noise"*

The lifecycle categories proposed in Figure 5 were sufficient to contain all of the requirement statements in the PDS. Furthermore, we found that although some statements were relevant to multiple categories, there was always a single category which was much more strongly associated with each statement than any other category. Some categories were left empty, either as "not applicable", such as Public Relations, or "to be determined", such as Installation. This reflects the dynamic nature of the PDS as development progresses.

Each statement was effectively decomposed using the RDTs proposed in Section 3.2. Some statements had to be rewritten to match our templates. For example, "Transmission of power shall be via an electrical cable" was rewritten as "The power for the pot shall be transmitted through an electrical cable." This was done to make "power" the subject of the statement rather than "transmission", which is better modeled in a verb phrase.

4. Core Product Model and Open Assembly Model

A design process begins with the identification of the product requirements and ends with the delivery of the design solution. During this process there is a two way interaction between the product requirements and the design solution. The requirements are updated, refined and decomposed during the generation of the design solution; the design solution is continuously checked against the requirements for validation.

Based on this interaction, we want to enable searching the products that correspond to some specific requirements or identifying the product characteristics affected by those requirements. To enable this search, both a model for requirements representation and a model for products representation are needed. We proposed in the previous section the Requirement Definition Templates (RDTs) as a model to formalize requirements; we propose in this section to use the Core Product Model (CPM) and the Open Assembly Model (OAM) as models to formalize the design solution. These models have been developed at the National Institute of

Standards and Technology (NIST); they were originally created in UML and subsequently implemented in XML and OWL. Some efforts of the ongoing research at NIST still focus on adapting these models to new areas, such as sustainability, and on enhancing them with semantics [22].

4.1 Description of CPM and OAM

We choose CPM and OAM for three reasons. First, we consider CPM and OAM as sufficiently complete and detailed models for the purpose of this paper. Second, we are very familiar with them. Third, we regard this work as an occasion to test CPM and OAM in the area of requirements engineering. The following is a brief description of the aspects of CPM and OAM relevant for our purpose; for a detailed description of them, please, refer to [23].

CPM is intended to form an open, non-proprietary, generic and extensible product model, capable of capturing the engineering context commonly shared throughout the product lifecycle, from the earliest ideation to manufacturing, operation and disposal. OAM is the CPM extension for assembly and tolerances representation.

In CPM, the model of a generic product contains requirements (the desired quality and performance of the product), function (what the product is supposed to do), form (the shape and material of the product) and behavior (how the product implements its function). In OAM, an assembly is modeled through assembly relationships (components and the relationships between them), kinematics (kinematic constraints and motion) and tolerances (the allowed variability of dimension and geometry). In the early design stage, the geometry and the parts of a product are not yet defined. In this stage, CPM and OAM allow for modeling the product with its functions and subassemblies. In the final design stage, the model of the product is completed with the information relative to its geometry, behavior, feature associations and tolerances.

The classes of CPM and OAM that are of major interest for the purpose of this paper are **Specification**, **Requirement, Artifact**, **Feature**, **Material**, **Geometry**, and **Form**, from CPM; **Assembly**, **Part**, **AssemblyFeature**, **ArtifactAssociation**, and **AssemblyFeatureAssociation** from OAM. Figure 8 shows, in UML format, the portion of the models that concern these classes.

The design of a product begins with the definition of its specification. **Specification** in CPM is the collection of information relevant to a product deriving from customer needs and/or engineering requirements; it is a container for the specific **Requirement**s that the product must satisfy. The requirements statements generated with the RDTs become instances of the class **Requirement**. The class **Requirement**, the function of which is to contain all the requirements that apply to the product to be designed, can then be specialized, depending on the application context. Each subclass could represent the requirements belonging to a particular lifecycle stage of the product, as previously explained in Section 3.2.

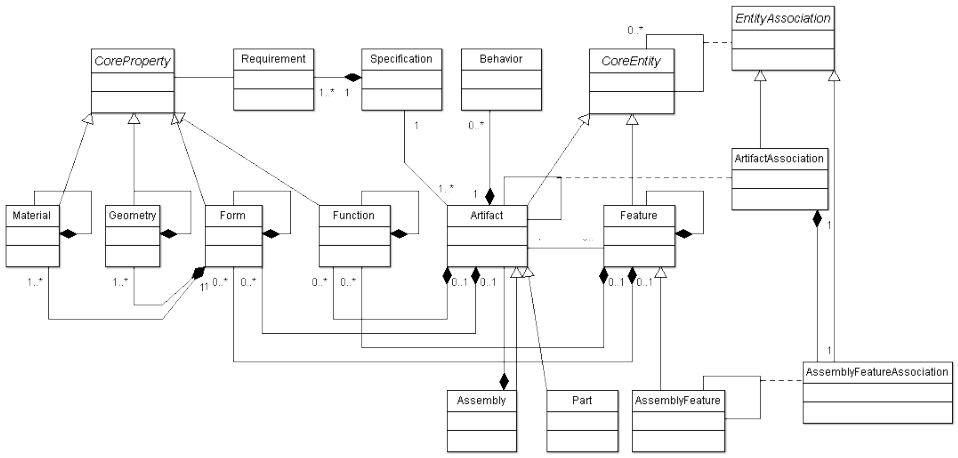


Figure 8: CPM and OAM classes relevant to PDS linking

After defining the requirements, the instances that represent the product itself are created. **Artifact** is the class in CPM that serves this purpose: it represents a distinct product, whether that product is a component, part, subassembly or assembly. Artifacts can be represented and interrelated through the subArtifactOf containment hierarchy. To each artifact one or more functions can be associated. **Function** in CPM represents what the artifact is supposed to do. The artifact satisfies customer needs and/or engineering requirements largely through its function. During the detailed design the artifact is characterized with its features; functions are then typically associated with the features. **Feature** is defined in CPM as a portion of the artifact that has some specific function assigned to it. Thus, an artifact may have design features, analysis features, manufacturing features, etc. **Feature** has its own containment hierarchy, so that compound features can be created out of other features.

Both artifacts and features can then be defined with their forms. The **Form** of the artifact or feature is the design solution adopted to realize the function. In CPM, the artifact's or feature's physical characteristics are represented by two distinct classes: **Geometry** and **Material**. **Geometry** is the spatial description of an artifact or feature while **Material** is the material composition of an artifact or feature. **Requirements** can apply not only to **Functions** but also to **Form**, **Material** and **Geometry**. **CoreProperty** is the parent class of all of them and all **CoreProperties** have their own containment hierarchy. For the purpose of this paper the relationship between CoreProperty and Requirements can be of the kind "satisfies" or "ensures", as will be explained in the next section.

In most cases, electro-mechanical products are assemblies, i.e., they are products constituted of other sub-assemblies and parts. Each component of an assembly (whether a sub-assembly or part) is represented in CPM with its requirements, functions, forms and behaviors, and the assembly relationships between components are represented in OAM.

In OAM the **Assembly** and **Part** classes are sub-classes of the CPM **Artifact** class. A Part is defined as the lowest level component of the assembly. When the features of the assembly components are in a mating relationship, e.g., a pin and a hole, they belong in OAM to the class **AssemblyFeature**, sub-class of the CPM **Feature** class. The class **AssemblyFeatureAssociation** represents the association between mating assembly features

through which relevant artifacts are associated. The class **ArtifactAssociation** is the aggregation of **AssemblyFeatureAssociation** and represents the assembly relationship between one or more artifacts. Both **ArtifactAssociation** and **AssemblyFeatureAssociation** are children of the CPM abstract class **EntityAssociation.** Since associated artifacts can have multiple feature-level associations when assembled, one artifact association may have several assembly features associations at the same time. Any assembly feature association relates in general to two or more assembly features.

For the purpose of this paper, we use CPM and OAM to model the final design solution. That is, we represent the designed product by instantiating the classes and relationships described in this section. We call these instances "design elements": for example, instances of **Material or Function** would be considered design elements.

4.2 CPM and OAM for representing the Automatic Pot Stirrer

The design solution proposed to meet the requirements consists of a normal pot containing a T-shape stirring shaft. The shaft is activated by a motor located in the top of the pot lid. The motor power is transmitted through an electrical cable and the motor speed can be regulated by the user.

Figure 9 represents in UML the object diagram of the Automatic Pot Stirrer. The diagram shows the instantiation of the CPM and OAM models of the pot assemblies, sub-assemblies and parts and their main functions. In this report, due to space limitations, we do not show the full instantiation of the Automatic Pot Stirrer. Instead, we present a few object diagrams, each dedicated to the representation of a particular aspect of the product design.

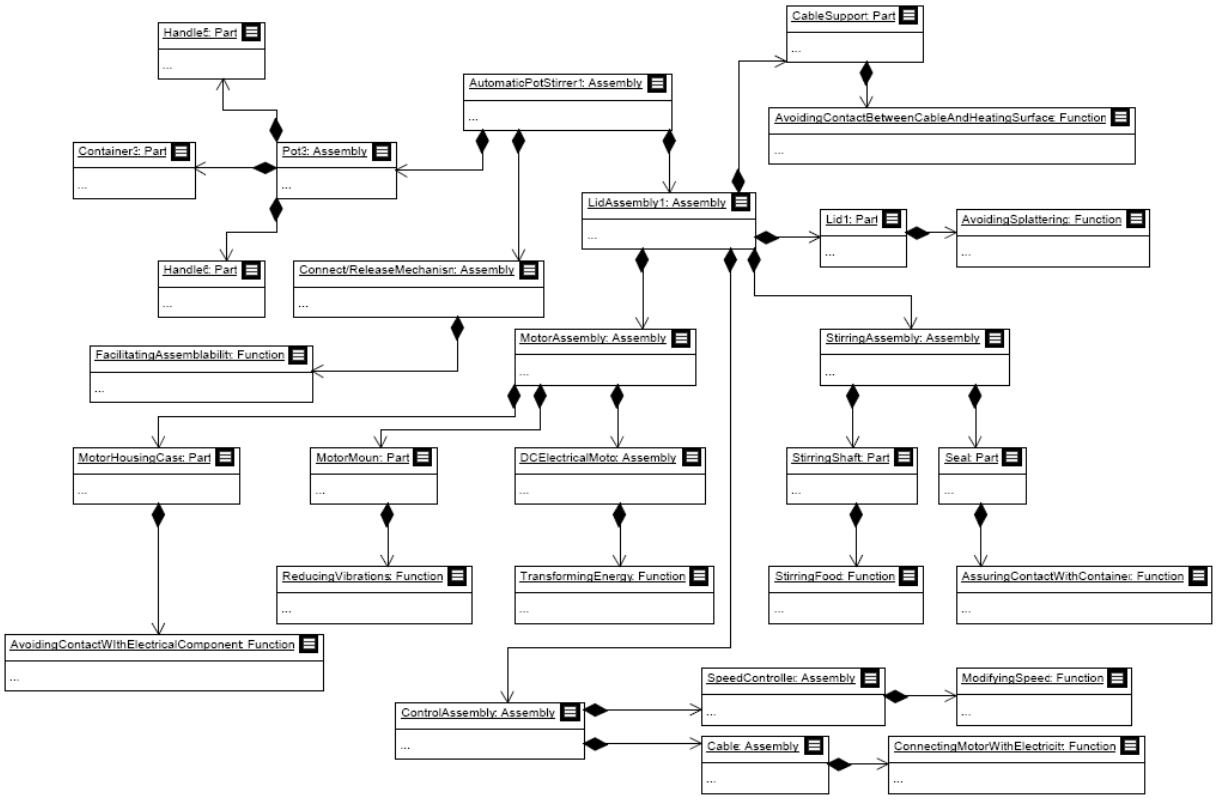


Figure 9: Assemblies and Functions of the Automatic Pot Stirrer

The Automatic Pot Stirrer is composed of three main assemblies: a *Pot*, a *LidAssembly* and a *Connect/ReleaseMechanism*. The *Pot* is composed of a *Container* and two *Handles* belonging to the class **Part** in OAM: its detailed representation is reported in Figure 10.

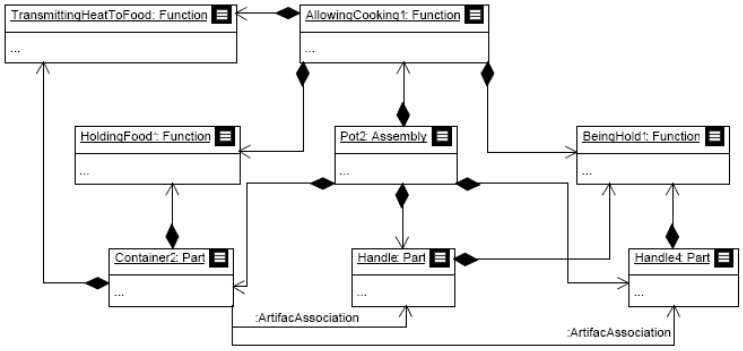


Figure 10: Assembly decomposition and function decomposition of the Pot

The generic function of the *Pot*, i.e., *AllowingCooking*, contains three sub-functions: *TransmittingHeatToFood* and *HoldingFood* are associated with the *Container* while *BeingHeld* is associated with the *Handles*. Two **ArtifactAssociation**s exist between the two handles and the pot. As an example, we provide in Figure 11 the CPM representation of the **Form**, **Geometry**, **Material** and **Features** of the *Container*. This example partially shows the instantiation of the CPM for the *Container* and is not intended as a full representation.

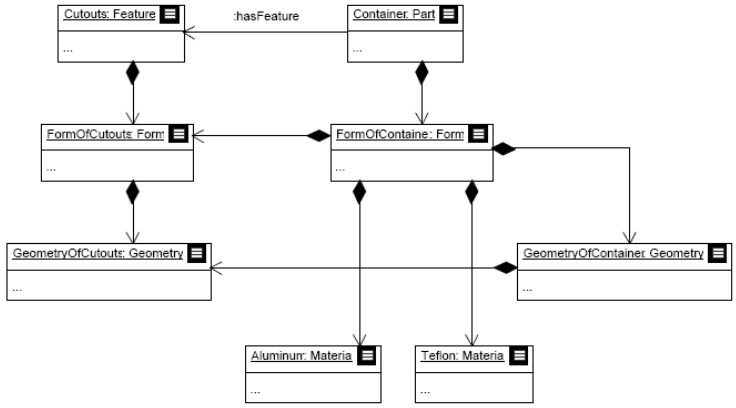


Figure 11: Form of the Container and form of its feature

At the top of the *Container* some *Cutouts* are designed: they are represented as **Feature** elements in CPM. Their form and, consequently, their geometry are part of respectively the form and geometry of the *Container*. *Aluminum* and *Teflon* are the **Material**s composing the *Container*.

The *LidAssembly* contains a *Lid*, a *CableSupport*, a *MotorAssembly*, a *StirringAssembly* and a *ControlAssembly*. The connections between the *Pot* and the *LidAssembly* are realized by the *Lid* and the *StirringAssembly*, both which are in contact with the *Container*. Figure 12 shows how the *Lid* and the *Container* are connected.

The *Lid* and the *Container* are linked through the **ArtifactAssociation** *Lid2Container*. This artifact association is realized by an **AssemblyFeatureAssociation** that involves the *Cutouts* of the *Container* and the *SecuringTabs* of the *Lid*, both which are **Feature**s of **Part**s involved in the two connected **Assemblie**s.

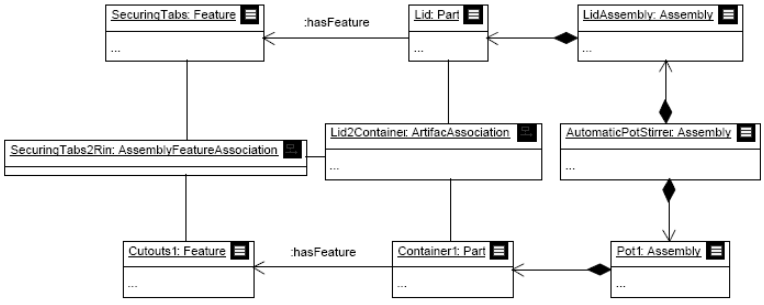


Figure 12: Assembly relationships between the Lid and the Container

The remaining product design representation follows the same principles adopted to draw the object diagrams illustrated here. The *MotorAssembly* contains a *MotorHousingCase* that accomplish the function of *AvoidingContactWithElectricalComponents*, a *MotorMount* with the function of *ReducingVibrations* and a *DCElectricalMotor*, with the function of *TransformingEnergy*. The *StirringAssembly* is composed of a *StirringShaft* and a *Seal*, which have the function, respectively, of *StirringFood* and *AssuringContactWithContainer*. The *ControlAssembly* is composed of a *SpeedController* for *ModifyingSpeed* and a *Cable* for *ConnectingMotorWithElectricity*. Finally, the *Cable* is supported by a *CableSupport*, itself part of the *LidAssembly*. Sub-functions, features, form, geometry, material and assembly relationships of all these design elements are also represented.

As the next step, we link the design elements of the Automatic Pot Stirrer to the requirements statements formalized through RDTs. The next section describes how we realize the links between design elements and the PDS statement.

5. Tracing Design Elements to Requirement Statements

During the first round of product development, when engineers make decisions regarding the design solution, the justification for each decision may seem fairly obvious, and thus no attempt is made to record it. However, when a product is upgraded or redesigned years later, even if the same people are working on the new design, the motivation for each decision may have been forgotten [18]. This could lead to difficulties when a change is made to either the PDS or the design solution. For example, if the requirements in the PDS have been changed in some way, the designer will need to know which elements of the design have been affected so that they can be reassessed. If, on the other hand, the designer makes a change to the design solution, some requirements may no longer be met. Although the designer could manually compare every design element to every requirement, this becomes highly time-consuming for complex designs with hundreds of requirements and thousands of design elements. To help track which design elements are relevant to which requirements, we introduce the concept of a tracing statement, or "link", which maps from the solution domain (CPM/OAM) to the requirements domain (PDS). A link consists of references to some design elements from the solution domain, references to some statements from the requirements domain, and a description of how the referenced design elements support the references requirements.

5.1 Linking and Labeling Criteria

Just as Section 3 presented a formal model for requirement statements, and Section 4 presented a formal model for the product design solution, this section outlines a formal model for the links made between these two domains. By formalizing the way in which links are represented, automated tools can be developed to help the designer to ensure that design solution elements are consistent with the requirements. Tools that depend on a formal linking model can be used, for example, to perform coverage analysis; every requirement and every design element must be connected to at least one tracing statement [18]. Coverage analysis is similar to, but distinct from the concepts of impact and derivation analysis. Coverage analysis looks at the relationship between the requirements and design solution, while impact and derivation analysis looks at the relationships among the requirements. Once coverage has been established, design element dependency can be summarized. A list of which design elements are dependent on which

requirements can be dynamically generated as design elements/requirements are added or removed. Finally, bottleneck analysis can be performed on these links. Bottleneck analysis measures how sensitive the requirements are to changes in the design, or vice-versa. If a particular design element is linked to many requirements, then it can be deduced that changing or removing this design element could have disastrous results in the requirements domain. As such, changes to this element must be made with special care to avoid violating requirements. A formal model for tracing statements must be designed to support the quantitative measures of coverage, dependency, and bottleneck analysis.

Additionally, tracing statements are valuable for the purpose of capturing design intent. Decisions with regard to the design solution may have arisen from many iterations of trial-and-error; as such, it is important that other designers who wish to upgrade the product should be able to access this information. By understanding that a particular possibility was chosen over others for a reason, the new designer can avoid repeating these mistakes. As another example, consider procurement personnel who might be looking to cut costs. It might make sense to him or her to replace a particular type of plastic with a cheaper plastic in the design. However, if the more expensive plastic was chosen because it won't melt when exposed to the temperatures of a microwave oven, then a proper tracing statement can ensure that the procurement personnel understands the problems associated with choosing the cheaper plastic.

5.2 A formal approach for linking requirements to design elements

Tracing statements can map one or more design solution elements to one or more requirements. The simplest tracing statements are one-to-one; they map one design solution element to one requirement. However, it may be necessary to link many related design elements or many related requirements (i.e. many-to-one, one-to-many, and many-to-many). We call these related design elements or requirements "groups". Grouping is useful, for example, when we wish to aggregate many one-to-one links into a single many-to-one or one-to-many link. This is done purely as a matter of convenience, to avoid repeating links that all refer to elements of the same group. In other words, this type of requirement grouping is necessary only from the practical standpoint of quickly creating large numbers of similar links. On other occasions, we truly wish to link a group of design elements or requirements as a whole, when no individual design element directly supports a requirement, or when no individual requirement is directly supported by a design element. This is necessary from a theoretical standpoint; if this type of grouping mechanism did not exist, then it might not be possible for certain requirements or design elements to be meaningfully linked.

Like requirement statements, tracing statements are constructed under a subset of English according to predefined templates. The semantics of tracing statements is further improved by mapping their components to a standard taxonomy, as with the taxonomies of Section 3.3. The templates developed for tracing statements are considerably more restrictive than requirement definition templates. Tracing statements typically do not contain a lot of information that is external to the requirements and design solution elements to which they refer. Therefore, it is unlikely that an author might encounter a situation where he would be unable to encode a trace statement using these templates. Furthermore, a restricted set of trace templates enables tools for the automated checking of design elements against requirement statements to be more easily constructed.

In these templates, a **<design_element_group>** is a group of units from CPM/OAM, and a **<requirement_group>** is in index to a requirement statement or group of statements in the requirement document. A **<trace_verb_clause>** is a verb clause constructed from one of the core trace verbs specified in Figure 13. It can be prefixed with "jointly" or "independently" to specify whether the design elements (in the case of a group) individually trace to the requirements or only realize the requirements as a group.

Similarly, the **<justification_mode_clause>** specifies how the requirements are justified by the design element(s), and whether or not they are satisfied individually or jointly. A **<justification>** is a grammatically and lexically unconstrained term that is intended to simply explain or justify the connection made in the statement to the end-user. It is not intended to be used as a search parameter, but rather to simply contribute information regarding design intent.

Satisfies (incidentally fulfills requirement)	Ensures (purposely fulfills requirement "by design")
• Allows	• Is imposed by
• Fulfills	• Assures
• Supports	• Certifies
• Facilitates	• Determines
• Provides for	• Is established by
• Suffices for	• Resolves
	• Guarantees

Figure 13: Complete taxonomy of trace statement linking verbs

The verbs in the trace taxonomy are grouped such that a distinction is made between design elements that "naturally" fulfill a requirement, and those that fulfill a requirement "by design." This type of classification is largely up to the discretion of the designer. For example, the design element *DCElectricalMotor* might "naturally" fulfill the requirement "components shall be easily and inexpensively replaceable." That particular motor might have been chosen for a reason other than how easily it can be replaced, which would simply be an incidental effect that happened to fulfill the requirement. Or, the designer might have chosen that motor specifically because it fulfills that requirement, and so would choose a descriptive verb from the "by design" group of verbs for the trace statement. Either way, future designers will be aware of the decision to use the motor, and the reasoning behind it.

A simple trace statement involving a one-to-one relationship between a design element and a requirement statement is the following: *FunctionOfLid* guarantees R10.3.1.1 by disallowing the food to escape the pot. Where "R10.3.1.1: The pot shall prevent splattering of the food on the operator". For the next examples we will omit the details of the justification for the trace statement.

In order to implement grouping in the requirements domain, it is necessary to construct a hierarchical tree of requirements and sub-requirements. In Section 3.5, the PDS of the Automatic Pot Stirrer has already been represented in a hierarchical tree. The predefined lifecycle categories of Figure 5, as well as user-defined subcategories, are ideally suited to defining groups of requirements. However, this imposes the limitation that a statement can only be a subrequirement of other requirements in the same category. Requirements can be indexed within these groups according to the needs of the author. For example, "R1.1.1: The pot shall not infringe on US Patent 5863121" is placed in the "R1.1.X: Patent Infringement" category, which is a subcategory of "R1.X: Conduct Market Research". "R10.4.1.2.1: The weight of the pot shall be less than 12 pounds," is a subrequirement of "R10.4.1.2: The pot shall be able to be lifted easily." To accommodate the occasional requirement that is classified under more than one category, copies of the requirement are generated, and a copy is placed into each category. Thus, parallelism between the way that requirements are categorized, and the way that they are

grouped, is maintained. This makes linking easier to use for tracking design decisions and modifications at a given stage of the product's lifecycle.

Once a hierarchical grouping tree has been constructed, the requirements and categories must be indexed. This is done by assigning each statement at the top level a number (e.g. R1, R2, etc.), and then assigning sub-requirements using an additional subindex for each level below the top level (e.g. R1.2, R1.2.1, R1.3.2., etc.). Categories are indexed using the placeholder "X". For example, R1.2.X represents the category "Conduct Market Research→ Public Relations".

The simplest way to group requirements is as an explicit conjunction using the "&" operator. Consider, as an example, the following tracing statement, taken from the Automatic Pot Stirrer use case:

Teflon assures R10.1.2.1 & R10.1.2.2 jointly

In this example, the material of *Container*, i.e., *Teflon*, assures the satisfaction of both "R10.1.2.1: The pot shall not permit sticking of the food to the wall of the pot" and "R10.1.2.2: The pot shall not permit sticking of the food to the bottom of the pot".
However, using the "&" operator can become cumbersome for groups consisting of more than a few requirements. Thus, a mechanism for implicitly grouping requirements based on their arrangement in the hierarchy is desirable.

Groups of requirements can be implicitly referenced in one of several ways: "inheritance", where a requirement as well as all of its parent requirements back to the top level are included, "descendance", where a requirement as well as all of its sub-requirements, and their sub-requirements down to the lowest level, are included, and "cross-section", where all requirements or sub-requirements at a particular level are included in the reference.

To implement the "inheritance" type of group selection in a tracing statement, we reference a requirement, and follow it with the "^" operator. For example, to select R1.1.2, its parent R1.1, and its parent R1, we use the term "R1.1.2^". The following tracing statement includes the "^" operator:

FormOfPot satisfies R10.4.1.2.1^

This trace statement states that the form of the pot satisfies the requirement: "R10.4.1.2.1: The weight of the pot shall be less than 12 pounds," as well as its parent: "R10.4.1.2: The pot shall be able to be lifted easily."

To implement "descendance" selection, we reference a requirement and follow it with the "*" symbol. In a hierarchy of requirements, R1, R1.1, R1.2, R1.3, R1.1.1, R1.1.2, and R1.2.1, we use the term "R1*" to encompass R1 and all of its sub-requirements down to the lowest level. For the Automatic Pot Stirrer use case, we write the following tracing statement:

FormOfPot Is imposed by R10.4.1.3* independently

to express that the form of the Pot was established to fulfill the requirement "R10.4.1.3: The pot shall be able to be held easily" and all its decendants, i.e., "R10.4.1.3.1: The diameter of the pot shall be 10 inches"and "R10.4.1.3.2: The depth of the pot shall be 6 inches".

Finally, to implement "cross-section" selection, we use the category operator "X" as a placeholder in the requirement index. In the same example as before, to select "R1.1, R1.2, and R1.3" for our tracing statement, we use the term "R1.X". This syntax can be compounded as appropriate. For example, "R1.X*" represents R1.1, R1.2, R1.3, and all of their sub-requirements down to the lowest level of the tree. In our use case we use R1.1.X to represent all

the requirements belonging to the Conduct Market Research → Patent Infringement category. We write:

FormOfAutomaticPotStirrer satisfies R.1.1.X

to assert that the form designed for the Automatic Pot Stirrer does not infringe any patent listed in the PDS.

Grouping is allowed not only for requirements but also, with the same purpose, for design elements. While for requirements the user can liberally select the type of group and the requirements that belong to it, for design elements the choice is more constrained. Groups of design elements belonging to the classes Artifact, Feature, Form, Geometry and Material are created out of the compositions relationships contained in the CPM and OAM models. The composition relationships are used as a mechanism for grouping as they specifically represent part-whole relationships. Every time the user chooses a design element to be linked to the PDS, a group is created containing that design element and the elements that compose it, directly and indirectly.

Consider, as an example, the class Feature that contains a composition relationship with itself and with the classes Function and Form, where Form is composed of Material and Geometry. If a specific feature is selected to realize the link, then all its sub-features, functions, forms, geometries and materials will be automatically grouped together and the group will be considered as a whole. This principle forces the user to select the most specific design element to realize the link. If the user, for example, intends to link only the material of a feature to the PDS, he will need to select the instance of the class Material, since selecting the instance of Feature would imply selecting also all its sub-elements. Obviously, if the selected element represents a leaf in the composition structure, it will not be necessary to create a group.

Consider again the example provided above:

FormOfPot satisfies R10.4.1.2.1^

FormOfPot in the trace statement represents the group of design elements consisting of the material and geometry of the pot and of its components (see Figure 10).

When the selected design element belongs to the class Function, the grouping cannot be automatically executed. Consider as an example, the design of a device that needs to provide purified water to the second floor of a building (first requirement). A function satisfying this requirement is “pumping water from the basement to the second floor”. Because the water needs to be purified at the first floor (second requirement), this function is decomposed into “pumping water from the basement to the first floor” and “pumping water from the first floor to the second one”. In this case, the first sub-function does not satisfy the first requirement so it would be meaningless to group it with its super-function and to state that the group satisfies the first requirement. Since grouping together a function and its sub-functions can be incorrect, an automatic grouping mechanism for Function cannot be established.

To allow the user to group design elements at his own discretion, we provide the possibility of using the “&” operator. When two or more design elements are joined together through the “&” operator, a group is created containing: first, the listed design elements and, second, their composing elements in case they belong to the classes Artifact, Feature, Form, Geometry or Material.

In our use case, the "&" operator between design elements is used, for example, in the following trace statement:

MaterialOfContainer & *MaterialOfSeal* & *MaterialOfStirringShaft* assures R10.6.1

where "R10.6.1: The pot shall not absorb odors."

Once a group of design elements is either automatically created or established by the user, it can be linked to a requirement or a group of requirements, so as to realize a many-to-one or a many-to-many mapping respectively.

5.3 Application of Tracing Statements

As mentioned earlier, tracing statements are useful for purposes such as coverage and bottleneck analyses. To demonstrate the efficacy of our tracing scheme, we present several queries which might be made by a design engineer or procurement personnel. We then solve these queries using our tracing statements.

1) What are the design elements related to the safety of the pot?

Such a query might be asked if an injury lawsuit is brought up against the company, and a lawyer must defend the safety of the current design. The results would include, for example: **Artifact:** *MotorHousingCase*, **AssemblyFeatureAssociation:** *SecuringTabsToRim*, and **ArtifactAssociation:** *SpeedController2Cable*.

2) Why did the designer choose silicone rubber for the seal?

This query may arise if the cost of silicone suddenly becomes significantly more expensive and the company must find a cheaper alternative. The results would contain requirements such as: "R10.6.1: The pot shall not absorb odors," "R10.2.4: The pot shall be able to endure temperatures up to 500F," "R10.6.2: The acoustic noise of the pot shall be less than 110 dbA," and "R1.4.1: The cost of the pot shall be less than $70." This would thus reveal that the chosen material no longer satisfies the cost requirement.

3) If I eliminate the requirement "R10.6.2: The acoustic noise of the pot shall be less than 110 dbA," what design elements will be affected?

Suppose the desired market for this product shifts from household usage, to industrial food production. Noise will no longer be as crucial of an issue. In this case, results of the query might yield: **Artifact:** *MotorMount*, **Assembly:** *DCElectricalMotor*, **Material:** *MaterialOfBlade*, **Material:** *MaterialOfSeal*, and **Material:** *MaterialOfContainer*.

These are just a few representative examples of a wide range of queries which can be automatically answered by using formalized tracing statements and a suitable software tool.

6. Conclusions and Future Work

To address the growing need for a means of consistently validating a product design solution against its PDS, we have developed a robust framework. Our main contributions were as following. First, we devised a system of RDTs and taxonomies to semantically augment the

requirements. After proposing a means for tagging individual words in each statement using device, use-environment, verb, and attribute taxonomies, we outlined a means for tagging entire phrases in each statement using an appropriate RDT. Requirements categories are then used to classify the requirements statements based on the product lifecycle.

Next, we showed how CPM and OAM can be used to model the final design solution. This consists of identifying the design elements, recognizing the relationships between them, and finally instantiating the CPM and OAM models.

Finally, a formal way of writing tracing statements to connect the design elements to the requirements statements is given. This consists of grouping design elements, grouping requirement statements, realizing each link using a taxonomy of trace verbs, and finally providing an optional justification for each trace.

To demonstrate how our model can be used effectively, the design of an Automatic Pot Stirrer was chosen as a use case. Appropriate device and use-environment taxonomies were developed, and all requirements for this device were successfully categorized and tagged. The design solution was represented in CPM and OAM, and design elements were linked to the requirements using formalized tracing statements. Finally, some realistic queries were generated and successfully solved using these tracing statements.

For this framework to be of practical use in the product development community, software tools must be developed. Future work will consist of developing tools for efficiently writing and tagging requirement statements, constructing the CPM and OAM models of the design solution, and composing tracing statements. We also plan to develop search and validation tools to allow personnel to search within and across multiple PDS, check that a product design has met all of its requirements, and identify critical design elements. These tools will assist in generating other use cases, which will allow us to more accurately assess the viability of our model. Finally, we will explore the relationships between our framework and several available standards in the fields of systems engineering and requirements engineering.

Acknowledgements: This research is supported in part by the National Institute of Standards and Technology's (NIST) Manufacturing System Integration Division.

Disclaimer: Any commercial product or company name in this paper is given for informational purposes only. Their use does not imply recommendation or endorsement by NIST.

7. References

1. Safayeni, F., Duimering, P. R., Zheng, K., Derbentseva, N., Poile, C., and Ran, B., "Requirements engineering in new product development," *Communications of the ACM*, Vol. 51, No. 3, 2008, pp. 77-82.
2. Grady, J., *System verification:proving the design solution satisfies the requirements*, Elsevier/Academic Press2007.
3. Aurum, A., and Wohlin, C., *Engineering and managing software requirements*, Springer2005.
4. Guide to the Software Engineering Body of Knowledge. http://www.swebok.org/ . 2009.
5. Telelogic DOORS. http://www.telelogic.com/corp/products/doors/doors/index.cfm . 2006.
6. McKay, A., Pennington, A. D., and Baxter, J., "Requirements management: a representation scheme for product specifications," *Computer-Aided Design*, Vol. 33, No. 7, 2001, pp. 511-520.

7. EIA-632, Processes for Engineering a System, GEIA standard. 1999.
8. IEEE-15288, Systems Engineering - System Life Cycle Processes, IEEE standard. 2004.
9. OMG SysML v1.1 Specification. http://www.omgsysml.org/ . 2008.
10. ASD-STE100, ASD Simplified Technical English, ASD standard. 2007.
11. ASD-STE100 training. http://www.asd-ste100.org/raining.htm . 2009.
12. Magrab, E., *Integrated Product and Process Design and Development: The Product Realization Process*, New York, CRC Press1997.
13. Golden, I., "Function Based Archival And Retrieval: Developing A Repository Of Biologically Inspired Product Concepts," MS Thesis, University of Maryland, 2005.
14. ISO. ISO 13584-1:2001, Industrial automation systems and integration -- Parts library -- Part 1: Overview and fundamental principles. 2001. Geneva, Switzerland, ISO.
15. Pahl, G., and Beitz, W., *Engineering Design*, Design Council, London,1984.
16. HThanh, Roberts, V., Tobias, C., Williams, A. M., Stirling, J., and Madelin, K., "Decision support at the wheel–rail interface: the development of system functional requirements," Vol. 222, Professional Engineering Publishing, Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit,2008, pp. 195-206.
17. Austin, J. L., *How to do Things with Words*, Oxford University Press1965.
18. Hull, E., Jackson, K., and Dick, J., *Requirements Engineering*, Springer2002.
19. Custompart.net: Free Online Manufacturing Cost Estimation and Education Resource. http://www.custompartnet.com . 2008.
20. Matweb. http://www.matweb.com . 2008.
21. Units Markup Language project. http://unitsml.nist.gov/ . 2009.
22. Fiorentini, X., Gambino, I., Liang, V., Foufou, S., Rachuri, S., Bock, C., and Mahesh, M., "Towards an ontology for open assembly model," International Conference on Product Lifecycle Management,2007, pp. 445-456.
23. Fenves, S., Foufou, S., Bock, C., Bouillon, N., and Sriram, R. D., "CPM2: A Revised Core Product Model for Representing Design Information ," National Institute of Standards and Technology, NISTIR 7185, Gaithersburg, MD 20899, USA, 2004.

Appendix A: Taxonomy of Attributes.

■ Auditory — Loudness — Pitch — Timbre ■ Environmental — Barometric pressure — Dirt/Dust — Humidity — Particulate Concentration — Precipitation — Temperature — Visibility — Wind speed — Wind direction ■ Financial — Cost — Price ■ Geometric — Area — Configuration — Dimension	■ Material — Acoustic • Acoustic Absorption • Refractive Index — Chemical • Activation Energy • Biodegradation • Chemical Energy • Concentration • Electro-negativity • Half-life • Hydrophobicity • Hygroscopy • Ionization Potential • Odor • pH • Radioactivity • Reactivity • Solubility • Surface Energy • Surface Tension	— Mechanical • Compressibility • Elongation › Elongation at Yield › Elongation at Break • Friction coefficient • Machineability Rating • Modulus › Compressive Modulus of Elasticity › Modulus of Rigidity › Modulus of Toughness › Secant Modulus › Shear Modulus › Specific Modulus › Tensile Modulus • Moment of Inertia • Piezoelectric Constant • Poissons Ratio • Strength › Charpy Impact	— Thermal • Coefficient of Thermal Expansion • Emissivity • Enthalpy • Entropy • Flammability • Heat of Combustion • Material Phase • Oxygen Index • Phase Transition › Autoignition Temperature › Boiling Point › Critical Temperature › Curie Point › Deflection Temperature › Eutetic Point › Fire Point › Flash Point

- Direction
- Location
- Perimeter
- Shape
- Volume

■ Identification
- Appearance
- Classification (e.g. species)
- Instantiation (e.g. serial number)
- Logo
- Motto
- Name

■ Quality Rating

■ Mechanics
- Angular momentum
- Force
- Kinetic Energy
- Intensity
- Mass
- Momentum
- Potential Energy
- Power
- Pressure
- Torque
- Work

■ Motion
- Acceleration
- Angular Acceleration
- Angular Velocity
- Deformation
- Displacement
- Motion
- Rotation
- Translation
- Velocity
- Vibration

■ Quantity
- Taste
- Toxicity

- Electrical
  - Arc Resistance
  - Capacitance
  - Charge
  - Current
  - Comparative Tracking Index
  - Conductivity
  - Dielectric Constant
  - Dielectric Strength
  - Dissipation Factor
  - Electromagnetic energy
  - Inductance
  - Magnetic flux density
  - Permittivity
  - Potential difference
  - Power
  - Resistance
  - Surface Resistance
- Failure
  - Buckling
  - Corrosion
  - Creep
  - Fatigue
  - Fracture
  - Impact
  - Mechanical Overload
  - Melting
  - Thermal Shock
  - Wear
  - Yielding
- Fluidic
  - Reynolds Number
  - Viscosity
    - Compressive Yield Strength
    - Ductility
    - Fatigue Strength
    - Fracture Toughness
    - Hardness
    - Izod Impact
    - Malleability
    - Rupture Strength
    - Tensile Strength at Yield
    - Tensile Strength at Break
- Optical
  - Absorptivity
  - Emissivity
  - Color
  - Gloss
  - Haze
  - Luminosity
  - Photosensitivity
  - Reflection Coefficient
  - Refractive Index
  - Transmittance
  - Visible Transmission
- Physical
  - Material Type
  - Composition
  - Density
  - Linear Mold Shrinkage
  - Melt Flow Index
  - Moisture Vapor Transmission
  - Oxygen Transmission
  - Porosity
  - Surface Roughness
  - Water Absorption
    - Moisture Absorption at Equilibrium
    - Water Absorption
    - Water Absorption at Saturation
    - Freezing Point
    - Glass Transition Temperature
    - Heat Distortion Temperature
    - Heat of Fusion
    - Heat of Sublimation
    - Heat of Vaporization
    - Liquidus Temperature
    - Melting Point
    - Softening Point
    - Solidus Temperature
    - Triple Point
  - Pyrophoricity
  - Seebeck Coefficient
  - Service Temperature
    - Hot Ball Pressure Test
    - Maximum Service Temperature
    - UL RTI
  - Specific Heat Capacity
  - Temperature
  - Thermal Conductivity
  - Thermal Diffusivity
  - Thermal Energy
  - Vapor Pressure

■ Signal
- Amplitude
- Attenuation
- Frequency
- Phase
- Wavelength

■ Time
- Duration
- Frequency