Artificial Intelligence for Engineering Design, Analysis and Manufacturing (2016), *30*, 337–352.
© Cambridge University Press 2016 0890-0604/16
doi:10.1017/S0890060416000330

An ontological approach to engineering requirement representation and analysis

ALOLIKA MUKHOPADHYAY AND FARHAD AMERI
Engineering Informatics Research Group, Texas State University, San Marcos, Texas, USA

(RECEIVED October 12, 2015; ACCEPTED May 31, 2016)

Abstract

Requirement planning is one of the most critical tasks in the product development process. Despite its significant impact on the outcomes of the design process, engineering requirement planning is often conducted in an ad hoc manner without much structure. In particular, the requirement planning phase suffers from a lack of quantifiable measures for evaluating the quality of the generated requirements and also a lack of structure and formality in representing engineering requirements. The main objective of this research is to develop a formal Web Ontology Language ontology for standard representation of engineering requirements. The proposed ontology uses explicit semantics that makes the ontology amenable to automated reasoning. To demonstrate how the proposed ontology can support requirement analysis and evaluation in engineering design, three possible services enabled by the ontology are introduced in this paper. These services are information content measurement, specificity and completeness analysis, and requirement classification. The proposed ontology and its associated algorithms and tools are validated experimentally in this work.

Keywords: Engineering Requirements; Information Content, Ontological Reasoning; Ontology; Requirement Management

1. INTRODUCTION

Requirement planning is one of the most critical activities in the product development process. Engineering requirements describe the attributes, behaviors, and functionalities that a product has to possess in order to satisfy the needs of product stakeholders (Pahl et al., 1984). Despite its significant impact on the outcome of the design process, requirement planning, particularly in the early design stages, suffers from a lack of quantifiable measures for evaluating the quality of the generated requirements and also a lack of structure and formality in representing engineering requirements. The quality of the generated requirements in terms of clarity, specificity, feasibility, traceability, and completeness is often appraised on a consensual basis. In addition, requirements are usually represented informally in natural language without following any standard protocol or terminology. Even in the same company, different design teams may follow different methods and conventions for representing engineering requirements. Engineering requirements, represented in natural language, can be imprecise and ambiguous and contain contradictory information (Tseng & Jianxin, 1998). Resolving ambiguities and inconstancies may lead to costly engineering changes in later stages of product development. In the absence of formal methods and models for engineering requirement representation, management of requirements becomes inefficient and tedious especially when designing complex products with multiple subsystems.

There are multiple formal requirements models in the domain of software engineering developed with the objective of improving the process of requirement elicitation, analysis, communication, validation, control, and reuse (Jureta et al., 2009). However, there is no universally accepted standard or formalism for requirement modeling and representation in engineering design (Jianxin & Chun-Hsien, 2006). A formal standard for requirement representation assists designers in retrieving, reusing, updating, and sharing engineering requirements within and across design projects. Furthermore, it can be used for creating searchable requirement repositories that are directly linked to design solutions. Linking design solutions to engineering requirements enables traceability of design attributes to stakeholder needs. Finally, formal requirement models enable integration of requirement planning tools with computer-aided engineering and design tools to support objective evaluation of design alternatives early in the conceptual design phase.

Reprint requests to: Farhad Ameri, Engineering Informatics Research Group, Texas State University, San Marcos, TX 78666, USA. E-mail: ameri@txstate.edu

The main objective of this research is to introduce a formal ontology for standard representation of engineering requirements. The particular focus is on the requirements, or "expectations," that are generated in the early stages of the design process, and they are not translated into engineering attributes and product specifications yet. The proposed ontology uses explicit semantics that makes the ontology amenable to automated reasoning. The most obvious benefit of a shared ontology is streamlined information exchange across the product value chain at a semantic level. Through using well-defined syntax and semantics, onotologies help avoid ambiguity and divergent semantic interpretation. However, this paper explores other utilities of a formal ontology beyond information sharing and semantic interoperability. More specifically, ontology-enabled requirement analysis is the main focus of this paper. To demonstrate how the proposed ontology can support requirement analysis, three possible services enabled by the proposed ontology are introduced in this paper. These services are information content measurement, specificity and completeness analysis, and requirement classification. Information-content measurement service can be used for quantifying the rate of uncertainty reduction, or information gain, throughout a design project. Because design is essentially an information transformation and generation process, it can be argued that an information-based metric can better reflect the progress of the design process compared to other indirect measures such as cost or time. Specificity and completeness analysis service can help designers evaluate the developed requirement in terms of the level of details incorporated in formulating the requirement. Too much specificity in requirement planning reduces designers' freedom during the ideation process, thus hindering innovation. Conversely, generic requirements cannot effectively define the feasible design space. Finally, requirement classification service can enable more effective design search and retrieval because design solutions are oftentimes attributed to a set of recurring requirements. Through automated classification of requirements, a designer can more readily search for design solutions that are associated with certain classes of requirements such as safety or durability.

The paper is divided into the following sections. Section 2 provides a discussion of the related works. The requirement ontology is described later with detailed examples and protocols related to converting requirement statements into ontological representation. In the latter sections of the paper, ontological analysis of requirements is introduced in three areas, namely, information content measurement, specificity analysis, and requirement classification. The paper ends with a conclusion and the outline for further work.

2. RELATED WORKS

Requirement modeling and representation is more rigorously studied in the software engineering domain, and several models and methods for structured and formal representation of requirements have been proposed and implemented (Schatz et al., 2005; Kossmann et al., 2008; Jureta et al., 2009; Mir et al., 2011; Qureshi et al., 2011). The main objective of requirement engineering in the software development domain is to materialize the vision for self-adaptive systems, the systems that can continuously adapt their behavior at runtime in response to changing user's requirements, operating contexts, and resource availability. Mir et al. (2011) proposed an ontology based on SysML for seamless integration of requirement model and system model in software development. Zhang and Zhang (2007) developed an ontology based on description logic for capturing and maintaining requirement-related knowledge in software engineering.

The body of research work in requirements representation and modeling in engineering design has been rather sparse. Yannou (2012) introduced a model-based approach using UML and SysML for requirement representation. In this work, property-based requirement theory (Micouin, 2008) was used to provide the underlying formalism of the requirement model. This work addresses "well-formed" requirements that are distinguished from "expectations." A well-formed requirement is defined as "a constraint applied to a property of an object (or system) when a condition is occurred (event) or is achieved (state)." Well-formed requirements are typically generated in the latter stages of design. In early stages, however, when requirements are still in the form of expectations, the proposed approach is not applicable.

Lamar and Mocko (2010) studied engineering requirements from a linguistic perspective and proposed a formalized syntax for requirement representation based on parts of speech, grammatical functions, and sentence structure. This study decomposes a requirement statement into four syntactical elements, namely, artifact, necessity, function, and condition. Using the proposed syntax and its associated analysis methods, one can assess the quality of requirement statements with respect to completeness, unambiguity, and traceability.

Morkos et al. (2012) developed a computational reasoning tool to help designers predict change propagation in the engineering domain. This tool uses the syntactical elements of requirements to build relationships between requirements. The syntactical elements used in Morkos' model include subject, modifier, verb (modal and transitive), object, and condition.

Lin et al. (1996) proposed an ontology for representing requirements that supports a generic requirements management process in engineering design. First-order logic is used as the knowledge representation formalism. In this ontology, engineering requirements are classified into four main categories: physical, structural, functional, and cost. The proposed ontology can be used for checking completeness, consistency, and satisfiability of engineering requirements.

Darlington and Culley (2008) proposed an ontology for organizing the terms used for capturing design requirements. The objective of this ontology is to eliminate the ambiguity about various concepts related to engineering requirements such as target market, requirement resource, stakeholder, and the like. The envisioned applications for this ontology include streamlining communication among design engineers, supporting software application development (in particular, developing case-based reasoning systems for engineering requirements), and improving the performance of search engines. This ontology, however, is high level and cannot be used for breaking down requirement statements into their elemental components. Other related works in requirement representation include the requirement taxonomy (Hauge & Stauffer, 1993), the customer attribute hierarchy (Yan et al., 2001), and the functional requirement topology (Tseng & Jianxin, 1998).

The literature study reveals that, despite some efforts for structured representation of engineering requirement, there is still a wide gap to be filled before requirement modeling can be fully formalized and integrated with other phases of the engineering design process. This work attempts to partially fill the gap through providing a formal ontology for representing top-level requirements, or expectations, that are directly elicited from customer need statements. The developed ontology provides a smooth transition between textual representation and model-based representation of engineering requirements.

3. REQUIREMENT ONTOLOGY (ReqOn)

ReqOn is a formal ontology aimed at representation of engineering requirements generated during the requirement planning phase when design concepts are not conceived yet. Due to its formal semantics, ReqOn is amenable to automated reasoning and machine processing. Therefore, it could be used for automation of requirement management. A linguistic and grammatical approach is adopted for ontology conceptualization (Morkos et al., 2012). Accordingly, parts of speech (verbs and nouns) and grammatical functions (subject, object, complement, and adjuncts) define the core classes of the ontology. The scope of ReqOn is currently limited to consumer products with medium complexity. However, the core classes of ReqOn are designed such that it can be evolved into a comprehensive requirements ontology that covers a wider range of artifacts. Web Ontology Language (OWL) is used as the ontology language of ReqOn.

3.1. ReqOn classes and properties

In ReqOn, each requirement statement is represented by the `Requirement` class, which has two disjoint subclasses: `FunctionalRequirement` and `NonFunctional Requirement`. A functional requirement is related to the use of a product and describes the necessary task, activity, or action that should be accomplished. For example, "*The electric kettle boils water quickly*" is a functional requirement because it pertains to the action of boiling water. Nonfunctional requirements describe the attributes of the product such as size, color, recyclability, or ease of repair. The requirement "*The electric kettle is light*" is an example of a nonfunctional requirement. The concept diagrams in Figure 1 and Figure 2 show the properties of the `FunctionalRequirement` and `Non FunctionalRequirement` classes, respectively.

Both types of requirements inherit `hasProduct` and `is Functional` properties from their common superclass (i.e.,

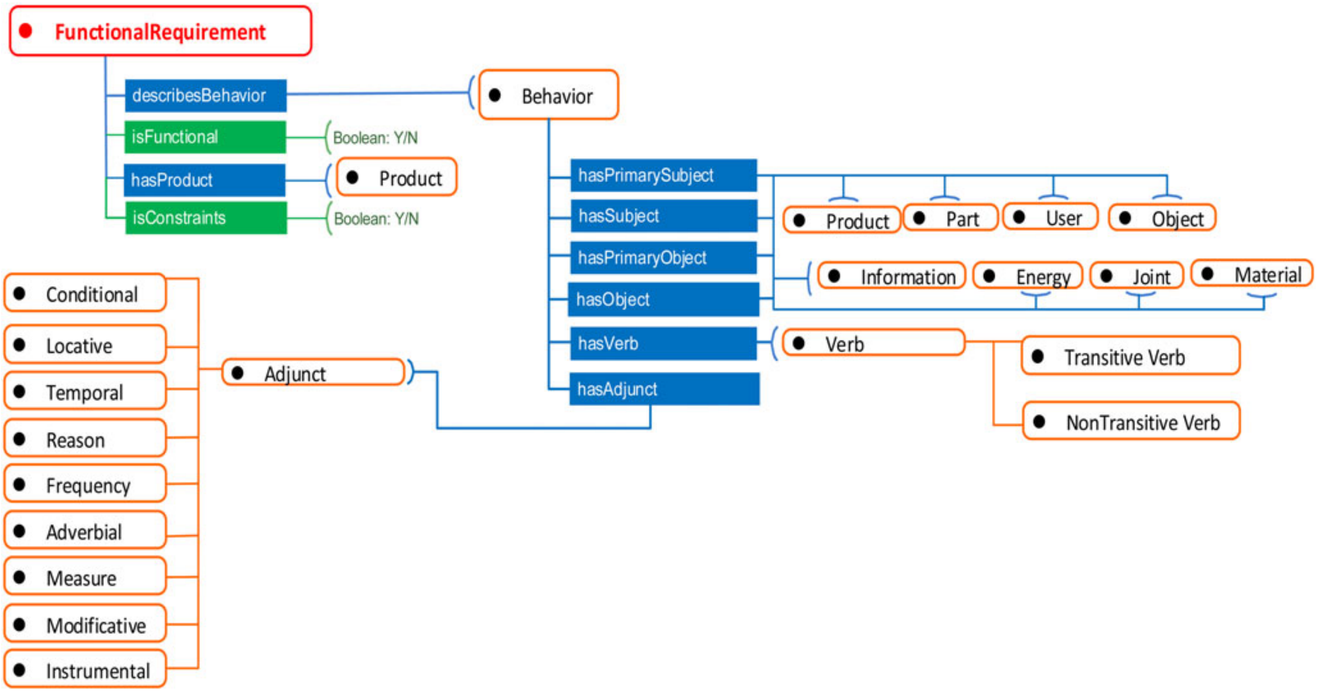


Fig. 1. Concept diagram for the FunctionalRequirement class.

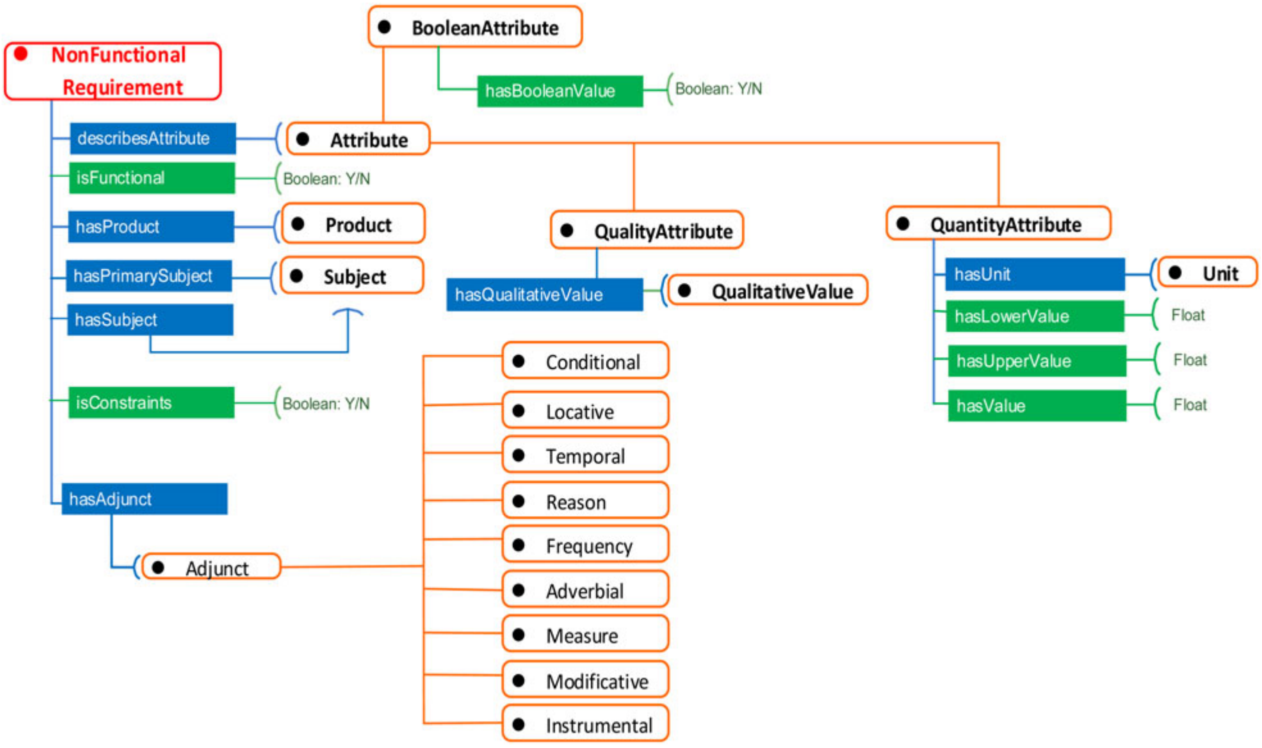


Fig. 2. Concept diagram for the NonFunctionalRequirement class.

Requirement). The property hasProduct refers to the product to which the requirement statement applies, and its range is limited to instances of the Product class. Each requirement statement has exactly one instance of the Product class associated with it. The property isFunctional is a Boolean property used for indicating whether or not the requirement is functional. describesBehavior is a property specific to the FunctionalRequirement class, and its range is limited to instances of the class Behavior. The behavior of a product describes the casual process through which the function is achieved (Sen et al., 2010). The Behavior class in the ontology is designed such that a series of subjects and objects that are involved in delivering the function can be linked to the behavior. Each behavior has exactly one action verb, either transitive or nontransitive. Because the functional basis (FB; Hirtz et al., 2002) provides a widely accepted functional schema in the engineering design community, this schema is adopted in ReqOn for breaking down the Verb class into more specific subclasses such as Connect, Convert, Support, and Branch. There are three possible types of functional requirements that can be modeled in ReqOn. Type 1 functional requirement describes the behavior of a product as a whole, Type 2 functional requirement describes the behavior of a part of a product, and Type 3 functional requirement describes the expected behavior of a product when it is the object of an action performed by the user.

Table 1 provides examples of different types of requirements that can be modeled in ReqOn. A functional requirement with a transitive verb has a primary subject and may have one or more secondary subjects. For example, if a functional requirement describes the behavior of *the wheel of a bicycle*, then *the wheel* is the primary subject and *the bicycle* is the secondary subject, because the requirement directly applies to the wheel and not to the bicycle itself. The requirement statements that have a transitive verb must have a primary object and may have one or more secondary objects. For example, in the requirement "*The electric wok has a lid that can be flipped easily*" the verb *flip* is Transitive Verb, the noun *lid* is PrimaryObject, and the noun *electric wok* is Object. Both object and subject can have different types such as user, product, part, material, energy, or signal. For instance, in the requirement "*A left-hand user easily handles the electric kettle*," *the user* is the primary subject. The taxonomy of the material, energy, and signal classes are directly imported from the FB. It is important to incorporate various class hierarchies in ReqOn because it allows for more accurate evaluation of the information content based on the depth of classes in the hierarchy.

The specificity of functional requirements can be improved by using adjuncts. An adjunct usually modifies a verb and indicates the time, manner, place, frequency, reason, degree, or condition pertaining to the requirement. For example, in the requirement statement "*the hand truck holds boxes securely on steep slopes*," *securely* is an Adverbial Adjunct while *on steep slopes* is a Locative Adjunct. As the requirements evolve, designers add more details to the requirement through introducing various types of adjuncts.

Table 1. *Different types of functional requirements*

Type 1: The product has a behavior as a whole. Example: The electric kettle boils water quickly.	
Product	Electric kettle
Subject	Electric kettle
Verb	Boils
Object	Water
Adverbial adjunct	Quickly
Type 2: The product has a part that has a behavior. Example: The electric kettle has a handle that insulates electricity.	
Product	Electric kettle
Primary subject	Handle
Subject	Electric kettle
Verb	Insulates
Object	Electricity
Type 3: The product (or its part) is the object in an action done by the user. Example: A left-hand user easily handles the electric kettle.	
Product	Electric kettle
Primary subject	Left-handed user
Verb	Handles
Object	Electric kettle
Adverbial adjunct	Easily

Table 2. *Different types of nonfunctional requirements*

Type 1: The product has a qualitative attribute. Example: The electric kettle is light.	
Product	Electric kettle
Qualitative attribute	Weight (implied)
Primary subject	Electric kettle
Value	Low
OR	
Product	Electric kettle
Boolean attribute	isLight
Value	True
Type 2: The product has a quantitative attribute. Example: The electric kettle's capacity is 1 liter.	
Product	Electric kettle
Primary subject	Electric kettle
Qualitative attribute	Volume
Value	1
Unit	Liter
Type 3: The product has a part that has an attribute (qualitative, quantitative, or Boolean). Example: The electric kettle has a cord that is long.	
Product	Electric kettle
Primary subject	Cord
Subject	Electric kettle
hasPart	Cord
hasQualityAttribute	Length
hasValue	High
Type 4: The product has a physical component. Example: The electric kettle has a dual water window.	
Product	Electric kettle
Primary subject	Electric kettle
hasPart	Dual water window
Type 5: The product (or one of its components) has a particular material. Example: The electric kettle has a plastic handle.	
Product	Electric kettle
Primary subject	Handle
Subject	Electric kettle
hasPart	Handle

A `NonFunctionalRequirement` uses `describes Attribute` property to describe the attributes of the product. There are three subclasses for the `Attribute` class in the requirements ontology, namely, `BooleanAttribute`, `QualityAttribute`, and `QuantityAttribute`. For example, in the requirement "*the phone is small in size*," *size* is the quality attribute, with *small* as its value. Grammatically, *small* is the *subject complement* in this example. However, if the actual dimensions, or ranges, are given for the size of the phone, then size can be regarded as a quantity attribute. A Boolean attribute is used for indicating if a product possesses a certain property. For example, in the requirement *the printer is easy to repair*, ease of repair can be treated as a Boolean attribute with a *true* value. Table 2 describes five possible types of nonfunctional requirement that can be represented in ReqOn. It should be noted that requirements that are developed at the early stages of the conceptual design phase are expected to be *form-free* to avoid premature anchoring of suboptimal solutions. Some of the example requirements provided in Table 2 contain solution-specific nouns such as cord, handle, and window. Those requirements are representative of the later stages of design when some solution fragments are identified.

Table 3 describes the protocol for identifying different components of natural language requirement statements and converting them into ontological representation.

3.2. ReqOn editor

ReqOn instances can be created and edited through Protégé ontology editor. However, working with Protégé requires OWL modeling knowledge that is a barrier for its widespread use. To facilitate ReqOn instance creations and manipulation, a Java-based tool was developed that uses OWL application program interface for interacting with the ontology.

The tool provides two different interactive user interfaces for functional and nonfunctional requirements. Figure 3 shows the functional requirement editor. The user simply has to identify the grammatical components of the requirement statement such as subject, object, verb, and adjunct. The user is encouraged to reuse the existing verbs in the provided verb taxonomy. However, if the appropriate verb cannot be found in the taxonomy, the tool can accept new verbs introduced by the user. In this way, the ontology evolves as new instances of requirements are added to the ontology. The tool and the ontology were validated through creating more than 300 instances of both func-

Table 3. *Protocol for creating requirement in ReqOn*

1. Identify the type of the requirement statement.
2. Rephrase the statement if necessary to match with one of the requirement patterns.
   a. Don't use the modals "can" or "must." Use simple present (third person) for the verb tense. For example, instead of "the cup can hold liquid," use "the cup holds liquid."
   b. If the user is implied in a requirement statement, make it explicit.
   c. Rewrite the statements in active voice (not passive voice). For example, instead of "the electric wok is handled easily by left-handed users," use "left-hand users easily handle the electric wok."
   d. When the product is the subject, start the statement with the name of the product.
3. Identifying requirements components.
   e. *Product:* It is the product for which the requirement is defined.
   f. *Subject:* the subject associated with a verb; it is either the product itself, a part of the product, or the user. In a nonfunctional requirement, subject is the entity to which the attribute pertains.
      i. A requirement can have multiple subjects, but it can only have one *primary* subject. The primary subject is the entity that is directly involved in the action. For example, in the statement "the pin of the paper punch makes holes in paper," both the paper punch and the pin are subjects but the pin is the primary subject because it performs the action "making hole" directly.
   g. *Object:* the object associated with transitive verbs; the primary object is the direct object that received the action of the action verb. If the direct object receives the action through a chain of objects, then those objects are regarded as secondary (indirect) objects.
   h. *Verb:* describes the action; it can be transitive or intransitive. Requirements with linking verbs (such as "is" and "are") are often represented as nonfunctional requirements (e.g., bicycle "is" easy to repair).
   i. *Adjunct:* identify the adjuncts that further modify the verb, object, or subject.
      i. For transitive verbs, try to use the verbs already available in the functional basis taxonomy.
   j. *Attribute:* a feature, property, quality, or component related to the product or its parts; a quality attribute can also be written as a Boolean attribute. For example, *bicycle has low weight* can be written as *bicycle is light* (isLight attribute with True value).

tional and nonfunctional requirements related to about 20 products. The students from a senior design course participated in the test and validation process.

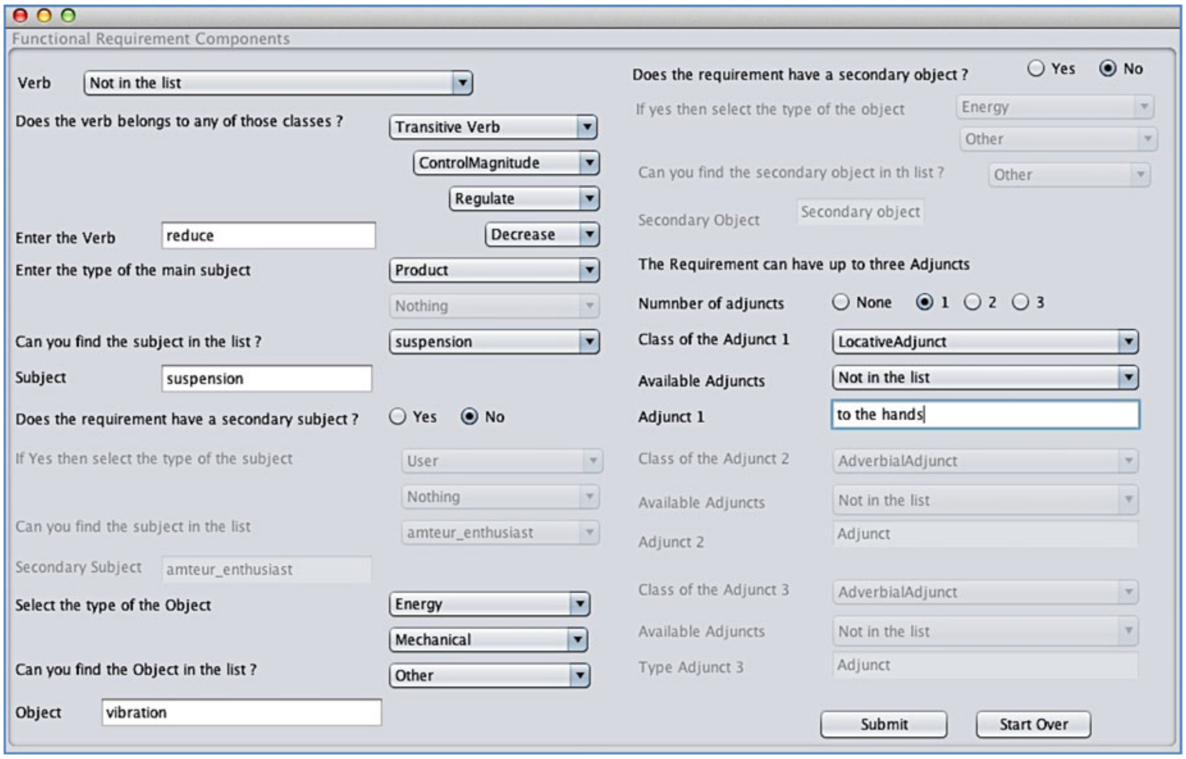
4. INFORMATION CONTENT MEASUREMENT

One metric for evaluating the performance of engineering design organizations is information generation and transformation rate (Collopy & Eames, 2001). A design project results in generation of different types of artifacts that embody design information. Hence, in order to study information transformation rate, first, the information content of the created artifacts should be objectively measurable so that given a step, where, say, requirements are transformed into function models, the amount of information input to and output from the step could be measured. Design artifacts are represented in various forms, such as text, sketch, or graphs (Chandrasegaran et al., 2013). This research is based on the premise that information is a form-neutral entity, and therefore, a design artifact such as a requirement statement, conveys the same amount of information to the designer irrespective of its form such as text or sketch, and that this amount could be measured. ReqOn provides a form-neutral and formal representation that is amenable to automated information content measurement. It exposes various parts of speech in the requirement statement as ontological classes that can be contemplated as containers of information. Because requirements typically evolve during the lifetime of a design project, it is useful to monitor the growth of information content of requirements as a way for evaluating the performance of the design team. It should be noted that maximizing the information content of requirements, particularly in the early stages of design, is undesirable because it could prevent creative thinking. However, as designers gain more insight into customer needs, both explicit and implicit, and learn about internal and external constraints, design requirements are typically loaded with more information.

The proposed information content measurement technique in this work is based on the entropy metric used in Shannon's information theory (Shannon, 1948). In information theory, entropy is the measure of uncertainty in a model. Shannon's metric measures information contained in a finite message composed of discrete symbols drawn from a finite vocabulary, such as the dots and dashes in a telegraph message (Shannon, 1948). According to this theory, the entropy of a discrete random variable X with a probability distribution $p(x)$ is defined by

$$H(X) \equiv -\sum_{x \in X} p(x) \log_2 p(x). \qquad (1)$$

Values of X with higher likelihood of occurrence have lower entropy according to this definition. Shannon's metric intends to measure the information content of a message in terms of the size of the unique vocabulary that the message is drawn from. In the context of measuring the information content of engineering requirements, each requirement statement can be treated as a finite message, composed of distinct terms drawn from a finite vocabulary such as ReqOn. The ReqOn representation of a requirement statement is essentially a graph in which graph nodes are the ontological instances and the arcs are the relationships between those instances. Figure 4 show a graph associated with a functional requirement. Measuring the information



Functional Requirement Components

Verb	Not in the list
Does the verb belongs to any of those classes ?	Transitive Verb
	ControlMagnitude
	Regulate
Enter the Verb	reduce
	Decrease
Enter the type of the main subject	Product
	Nothing
Can you find the subject in the list ?	suspension
Subject	suspension
Does the requirement have a secondary subject ?	Yes [✓ No]
If Yes then select the type of the subject	User
	Nothing
Can you find the subject in the list	amteur_enthusiast
Secondary Subject	amteur_enthusiast
Select the type of the Object	Energy
	Mechanical
Can you find the Object in the list ?	Other
Object	vibration

Does the requirement have a secondary object ?	Yes [✓ No]
If yes then select the type of the object	Energy
	Other
Can you find the secondary object in th list ?	Other
Secondary Object	Secondary object
The Requirement can have up to three Adjuncts	
Numnber of adjuncts	None [✓ 1] 2 3
Class of the Adjunct 1	LocativeAdjunct
Available Adjuncts	Not in the list
Adjunct 1	to the hands
Class of the Adjunct 2	AdverbialAdjunct
Available Adjuncts	Not in the list
Adjunct 2	Adjunct
Class of the Adjunct 3	AdverbialAdjunct
Available Adjuncts	Not in the list
Type Adjunct 3	Adjunct

Fig. 3. Screenshot of the requirement editor for functional requirement.

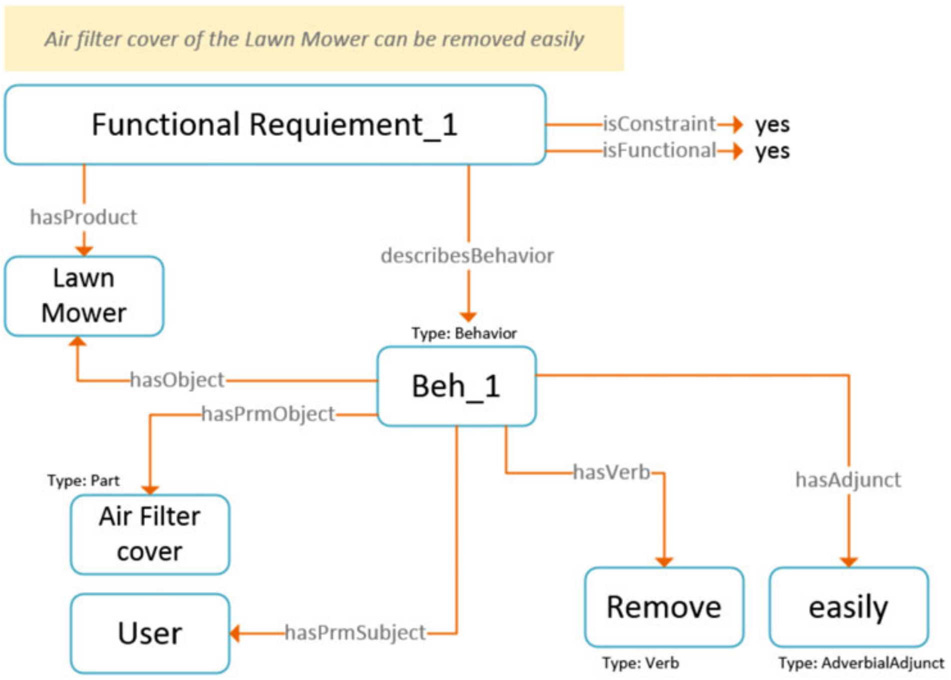


Fig. 4. Graph associated with an example functional requirement.

content of a requirement statement entails measuring the information content of the nodes of this graph.

Each node of the graph can be treated as a variable. The proposed information content measurement method is based on the assumption that the value of each design variable is selected from a set of known and finite options defined by the ontology. To assign a value to the variable, the type (class) of value should be determined and then a particular value needs to be selected from the set of possible values for that particular class. The first selection requires traversing the class structure and arriving at a particular type (node), and the second selection entails exploring the solution space for the selected class. Because the information content of each node is a function of the inherent uncertainty (or entropy) of the node, the underlying probability distribution of each node has to be known. A uniform probability distribution is assumed for the leaf classes of the ontology. In this way, the probability of an upper level class can be calculated through aggregation of the probabilities of its direct and indirect subclasses. The uniform probability is based on the assumption that all classes in the reference vocabulary can be encountered with equal likelihood. This assumption makes sense in the context of conceptual design because designers are usually encourages to avoid any bias in favor of any particular design idea during the ideation process.

The entropy associated with the structure of a class is referred to as *taxonomy entropy* in this work, and the entropy attributed to the number of direct class instances is called *size entropy*. The classes that have more complex subclass structure introduce more uncertainty. In addition, the classes that are instantiated more frequently have higher entropy because the probability of encountering a particular instance would be low. The taxonomy entropy of a class is based on the probability that a particular class is selected when traversing the class structure, and the size entropy is based on the probability that the selected class assumes a certain value. The total entropy of a class (Ec_i) is calculated as the summation of taxonomy entropy ($\mathrm{TE}c_i$) and size ($\mathrm{SE}c_i$) entropy.

$$Ec_i = \mathrm{TE}c_i + \mathrm{SE}c_i. \tag{2}$$

4.1. Calculating taxonomy entropy

The following steps are used for calculating the taxonomy entropy of a class. The `Verb` class is used as an example here. The hierarchical structure of `Verb` is shown later in Eq. (5).

- Count the number of leaf classes under each parent class (N). A leaf class is one that does not have any subclasses. For example *NonTransitiveVerb*, *Divide*, and *Import* are examples of leaf classes. In Figure 5, parent class Verb has 36 leaves.
- Assign a probability of (1/number of leaf under parent class $= 1/N$) to each leaf. For example, assign probability of 1/36 to *NonTransitiveVerb*, *Divide*, and *Import*. It is assumed that all leaf nodes have equal likelihood of occurrence (uniform probability distribution).
- For the rest of the subclasses under the parent class, count the number of leaves (n) under them. For example, the count of leaves under class *Branch* is $n = 4$.
- Probability of all subclasses other than leaf is the number of leaf nodes under the selected class divided by the total number of leaf nodes under its parent class (n/N). For example, probability of occurrence of class *Branch* $= 4/36$.

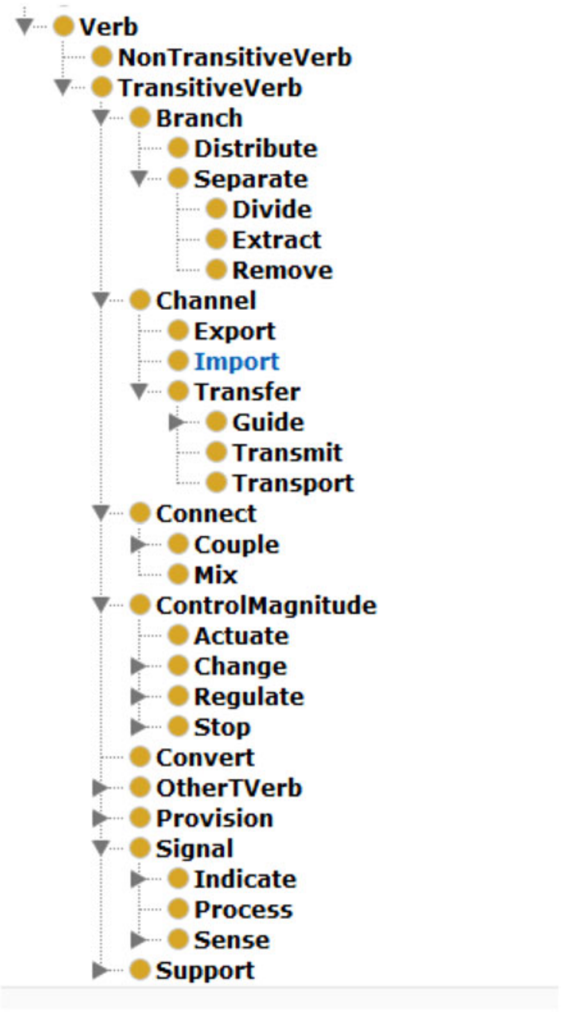


Fig. 5. Hierarchical structure of the verb class.

- After determining the probability of occurrence of the class, the entropy measure is applied for calculating the taxonomy entropy of the class.

$$\mathrm{TE}c_i = -\log_2(Pc_i) \tag{3}$$

where $\mathrm{TE}c_i$ is the taxonomy entropy of the ith class c_i and Pc_i is the probability of occurrence of class c_i. For example, the TE of class Branch ($\mathrm{TE}_{\mathrm{branch}}$) is calculated as follows:

$$\mathrm{TE}_{\mathrm{Branch}} = -\log_2 P_{\mathrm{Branch}} = -\log_2 \frac{4}{36} \approx 2.97 \text{ bits}. \tag{4}$$

4.2. Calculating size entropy

The size of a class refers to the number of individuals, or instances, of the class at any given time. The size entropy of class c_i is calculated using Eq. (5).

$$\mathrm{SE}c_i = -\log_2\left(\frac{1}{Nc_i}\right), \tag{5}$$

where Nc_i is the number of instances for class c_i. For example, if there are nine instances under the class `Material`, then

$$\mathrm{SE}c_{\mathrm{material}} = -\log_2 \frac{1}{9} = 2.197 \text{ bits.} \tag{6}$$

For compound classes such as `Behavior`, size and taxonomy entropy are measured through summation of the entropies associated with the constituting classes, namely, `Subject`, `PrimarySubject`, `Object`, `PrimaryObject`, `Verb`, and `Adjunct`.

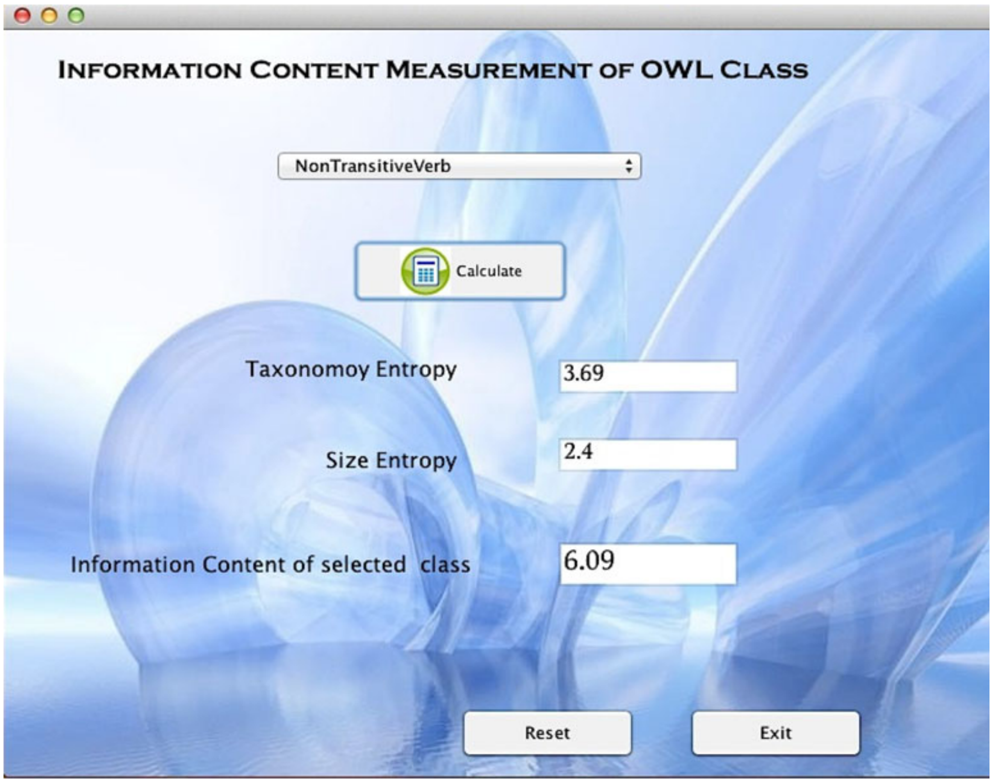
$$E_{\mathrm{Behavior}} = E_{\mathrm{Subject}} + E_{\mathrm{PrimarySubject}} + E_{\mathrm{Object}} + E_{\mathrm{PrimaryObject}} + E_{\mathrm{Verb}} + E_{\mathrm{Adjunct}}. \tag{7}$$

A Java-based tool was developed based on the proposed algorithm. The tool measures the information content of the requirements represented ontologically. The developed tool uses OWL application program interface for interacting with the ontology. The user needs to translate the engineering requirements written in natural language into ReqOn representation. The tool receives an OWL/XML file as the input a measures the information content of the selected requirement statements or classes as shown in Figure 6 and Figure 7. For example, the information content of the requirement statement "*the suspension can carry riders weighting up to 250 lbs.*" is measured to be 22.12 bits. It should be noted that this value should be treated as a relative value. A similar requirement statement, such as "*the suspension supports riders*," contains 17.21 bits of information meaning that, in comparison to the previous requirement, is less informative. Different components of information content for these two examples are provided in Table 4.

The proposed information content measure can be used for various purposes such as comparing the performance of different engineering design teams, evaluating the productivity of design projects with respect to information gain, and comparing different families of products with respect to information content measure.

5. SPECIFICITY AND COMPLETENESS EVALUATION

The information content measure that was introduced in the previous section can be used for evaluating the overall performance of engineering design teams with respect to the rate of information gain or uncertainty reduction. However, there is a need to evaluate the quality of the generated requirement quantitatively. Joshi and Summers (2014) developed a method for evaluation of specificity and completeness of requirements written in natural language. As the number of requirements in a project grows, manual analysis and evaluation of requirements becomes error prone and tedious. In the presence of a formal ontology, it is possible to automatically evaluate the quality of the requirement statements. Specificity and completeness are used as the metrics of quality evaluation in this work.



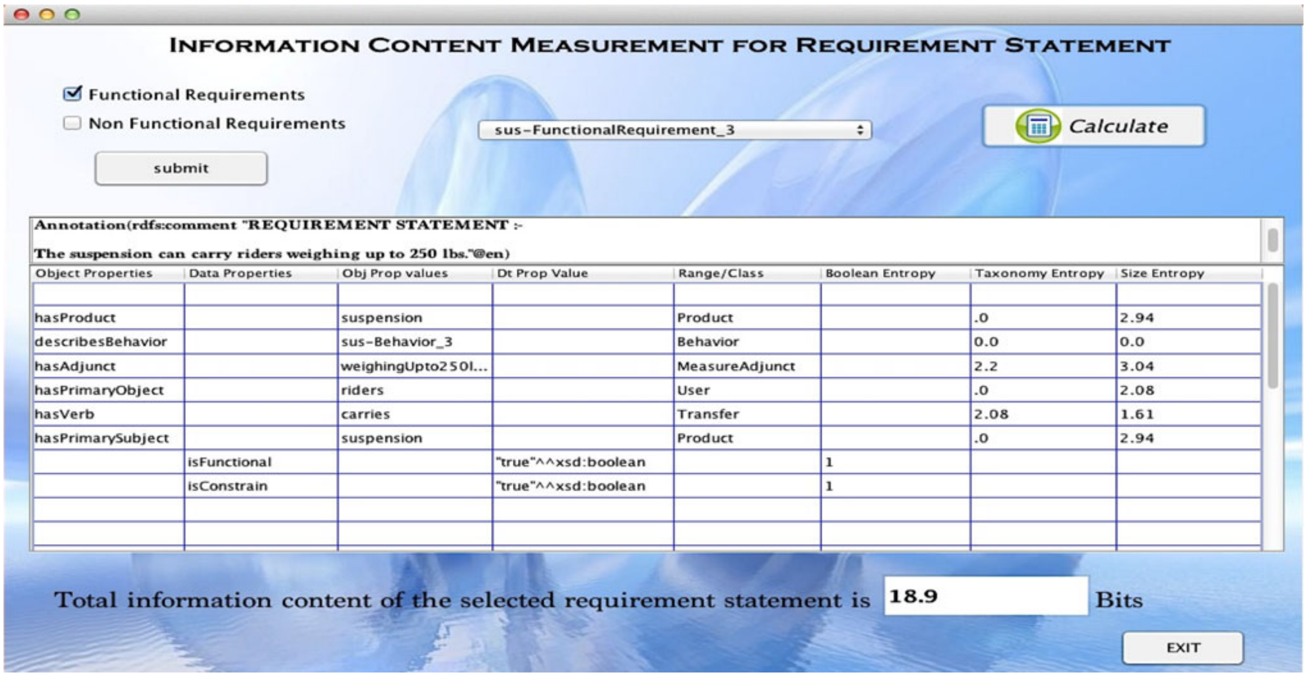
NonTransitiveVerb

Taxonomoy Entropy 3.69

Size Entropy 2.4

Information Content of selected class 6.09

Fig. 6. Screenshot of the information content measurement tool for classes.


INFORMATION CONTENT MEASUREMENT FOR REQUIREMENT STATEMENT
☑ Functional Requirements
☐ Non Functional Requirements
submit
sus-FunctionalRequirement_3
Calculate
EXIT

Annotation(rdfs:comment "REQUIREMENT STATEMENT :-

The suspension can carry riders weighing up to 250 lbs."@en)

Object Properties	Data Properties	Obj Prop values	Dt Prop Value	Range/Class	Boolean Entropy	Taxonomy Entropy	Size Entropy
hasProduct		suspension		Product		.0	2.94
describesBehavior		sus-Behavior_3		Behavior		0.0	0.0
hasAdjunct		weighingUpto250l...		MeasureAdjunct		2.2	3.04
hasPrimaryObject		riders		User		.0	2.08
hasVerb		carries		Transfer		2.08	1.61
hasPrimarySubject		suspension		Product		.0	2.94
	isFunctional		"true"^^xsd:boolean		1		
	isConstrain		"true"^^xsd:boolean		1		

Total information content of the selected requirement statement is 18.9 Bits

Fig. 7. Screenshot of the information content measurement tool for full requirement statements.

5.1. Completeness analysis

Completeness analysis is conducted through checking the presence of the necessary elements in the requirement statement. Requirements are often incomplete in the early stages of design to promote innovation and creative thinking. However, the focus of the completeness analysis in this paper is on structural and syntactic completeness and not technical completeness. If a requirement contains all of the required elements, depending on its type, it is considered to be a complete requirement structurally. Otherwise, it is incomplete. Table 5 and Table 6 show the essential properties of functional and nonfunctional requirements, respectively.

To assign a completeness score to each instance of the *Requirement* class, a Boolean data property named `hasCompletenessScore` was introduced. The domain of the property `hasCompletenessScore` is the `Requirement` class, and its range is between 0 and 1. A functional requirement will have a completeness score of 1 if it describes a behavior, which has exactly one product, one primary sub-

Table 4. *Different components of IC for two example requirements*

Object Property	Data Property	Property Value	Class	TEc_i	SEc_i	Ec_i
Functional Requirement: Suspension Can Carry Riders Weighing <250 lb						
hasPrimarySubject		Suspension	Product	0	3.3	3.3
hasPrimarObject		Riders	User	0	2.08	2.08
hasProduct		Suspension	Product	0	3.3	3.3
hasVerb		Carries	Transfer	2.94	2.4	5.34
hasAdjunct		Weighing <250 lb	Measure adjunct	2.89	3.3	6.19
	isFunctional	True	NA			1
	isConstraint	True	NA			1
Total				5.83	14.38	22.21
Functional Requirement: Suspension Supports Riders						
hasPrimarySubject		Suspension	Product	0	3.3	3.3
hasPrimarObject		Riders	User	0	2.08	2.08
hasProduct		Suspension	Product	0	3.3	3.3
hasVerb		Supports	Secure	4.74	1.79	6.53
	isFunctional	True				1
	isConstraint	True				1
Total				4.74	10.47	17.21

Table 5. *Cardinality restriction of essential properties for functional requirement*

Property Name	Property Type	Domain	Range	Cardinality	Restriction
hasProduct	Object	`Requirement`	`Product`	1	`hasProduct` exactly 1 `Product`
describesBehavior	Object	`Functional Requirement`	`Behavior`	1	`describesbehavior` exactly 1 `Behavior`
hasPrimarySubject	Object	`Behavior`	`Subject*`	1	`hasPrimarySubject` exactly 1 `Subject`
hasVerb	Object	`Behavior`	`Verb`	1	`hasVerb` exactly 1 Verb
hasPrimaryObject	Object	`Behavior`	`Object*`	1	`hasPrimaryObject` exactly 1 `Product`

Table 6. *Cardinality restriction of essential properties for nonfunctional requirement*

Property Name	Property Type	Domain	Range	Cardinality	Restriction
`hasProduct`	Object	`Requirement`	`Product`	1	`hasProduct` exactly 1 `Product`
`hasPrimarySubject`	Object	`Requirement`	`Subject`	1	`hasPrimarySubjec`t exactly 1 `Subject`
`describesAttribute`	Object	`Nonfunctional Requirement`	`Attribute`	1	`describesAttribute` minimum 1 `Attribute`

ject, one verb, and one primary object (if the verb is transitive). A nonfunctional requirement will have a completeness score of 1 if it has exactly one product, one primary subject, and describes at least one attribute. Table 7 shows examples of complete functional and nonfunctional requirements. Adjuncts contribute to the specificity of requirements, and they are not included in completeness analysis. A requirement can be structurally complete without an adjunct.

Completeness score computation and assignment is implemented through Semantic Web Rule Language (SWRL) rules as shown in Table 8. The first two rules compute the completeness score for functional and nonfunctional requirements, and the last two rules classify requirements as complete or incomplete requirements. Incomplete requirements are returned to the design team for revision.

5.2. Specificity analysis

Complete requirements can be further analyzed to evaluate their specificity. Specificity reflects the level of details incorporated in the requirement. More specific requirements are more informative. For complete functional requirements, five criteria are used for specificity analysis, namely, existence of secondary subject, depth of verb, existence of secondary object, number of adjunct, and existence of a measure adjunct.

5.2.1. Existence of more than one subject

Requirements with more than one subject are more specific or informative. For example, the requirement statement "*hand truck has a base pad that holds large and odd-shaped loads securely*," has a primary subject, *base pad*, and a secondary subject, *hand truck*, whereas, the requirement statement

Table 7. *Example of complete functional and nonfunctional requirement*

Functional Requirement Example 1: Suspension Reduces Vibration to the Hands		
hasProduct	Exactly 1	Suspension
describesBehavior	Exactly 1	Reduces vibration to the hands
hasPrimarySubject	Exactly 1	Suspension
hasVerb	Exactly 1	Reduces
hasPrimaryObject	Exactly 1	hands
Nonfunctional Requirement Example 1: Suspension Weighs Between 5 and 10 lb		
hasProduct	Exactly 1	Suspension
hasPrimarySubject	Exactly 1	Suspension
describesAttribute	Minimum 1	Weight
hasUpperValue	Optional	10
hasLowerValue	Optional	5
hasUnit	Optional	Pounds

Table 8. *SWRL rules for completeness score assignment*

Rule 1
FunctionalRequirement(?*f*) ∧ *hasProduct*(?*f*, ?*x*) ∧ *describesBehavior*(?*f*, ?*b*) ∧ *hasPrimarySubject*(?*b*, ?*p*) ∧ *hasVerb*(?*b*, ?*v*) ∧ *hasPrimaryObject*(?*b*, ?*o*) → *hasCompletenessScore*(?*f*, 1)
Rule 2
NonFunctionalRequirement(?*n*) ∧ *hasProduct*(?*n*, ?*x*) ∧ *describesAttribute*(?*n*, ?*a*) ∧ *hasPrimarySubject*(?*n*, ?*p*) → *hasCompletenessScore*(?*n*, 1)
Rule 3
Requirement(?*x*) ∧ *hasCompletenessScore*(?*x*, 1) → *Complete*(?*x*)
Rule 4
Requirement(?*x*) ∧ *hasCompletenessScore*(?*x*, 0) → *Incomplete*(?*x*)

Table 9. *Algorithm to calculate specificity score of functional requirement*

Algorithm 1

```
Individual
: R Types:Functional Requirement , describesBehavior value "behavior1"
Individual : behavior1 , Types:Behavior
if (behavior1 hasPrimarySubject exactly 1 Subject and
        hasSecondarySubject some Subject ){
$score_i = 20$ };
else if (behavior1 hasPrimarySubject exactly 1 Subject ){
$score_1 = 10$};
else $score_1 = 0$;
return $score_1$ ;
```

Algorithm 2

```
Individual
: R Types:Functional Requirement , describesBehavior value "behavior1"
Individual : behavior1 , Types:Behavior , hasVerb exactly 1 Verb
Individual : verb1 , Types:Verb
Literal : d xsd:integer $d \leftarrow$ depth of Verb
$score_2 = 10 * d$ ;
return $score_2$ ;
```

Algorithm 3

```
Individual
: R Types:Functional Requirement , describesBehavior value "behavior1"
Individual : behavior1 , Types: Behavior
if (behavior1 hasPrimaryObject exactly 1 Object and
        hasSecondaryObject some Object ){
$score_i = 20$ };
else if (behavior1 hasPrimaryObject exactly 1 Object ){
$score_i = 10$};
else $score_3 = 0$;
return $score_3$;
```

Algorithm 4

```
Individual
: R Types:Functional Requirement , describesBehavior value "behavior1"
Individual : behavior1 , Types: Behavior , hasAdjunct some Adjunct
Literal : n xsd:integer
$n \leftarrow$ Count of Adjunct
Error! Bookmark not defined. ;
else $score_4 = 20$};
return $score_4$ ;
```

Algorithm 5

```
Individual
: R Types:Functional Requirement , describesBehavior value "behavior1"
Individual : behavior1 , Types: Behavior
if (behavior1 hasAdjunct some MeasureAdjunct) {
$score_5 = 10$ };
else $score_5 = 0$;
return $score_5$ ;
```

Algorithm 6

$$Total\,Score = \sum_{i-1}^{5} W_i \times Score_i$$

$$Total\,Score = 5 \times score_1 \times + 4 \times score_2 + 3 \times score_3 + 2 \times score_4 + 1 \times score_5 \quad (8)$$

Maximum possible Total Score = 370
After Normalizing,

$$\textbf{\textit{Specificity Score}} = \frac{\textbf{\textit{Total Score}}}{\mathbf{370}} \quad (9)$$

"*hand truck holds large and odd-shaped loads securely*," has only a primary subject, *hand truck*. The first requirement describes a function of a component of the product while the second requirement describes a function of the product itself. As discussed before, a more specific requirement is not necessarily preferred over a less specific one due to confining the feasible design space. Designers are usually encouraged to start with a less specific list of requirements for more effective use of design freedom. In the *hand truck* example, the more specific requirement forces the designer to incorporate a *base pad* into the design concept.

5.2.2. Depth of verb

The depth criterion is used as a measure of specificity because it can be argued that deeper classes in a taxonomy are more specific than top-level classes. The proposed ontology uses hierarchical structure for some of classes such as `Verb`, `Energy`, `Material`, `Adjunct`, and `Attribute`. The class `Verb` is the most significant constituent of a functional requirement. Therefore, its depth is used as a criterion for specificity analysis.

5.2.3. Existence of more than one object

Functional requirements with a transitive verb have at least one object. Requirements with more than one object are deemed more informative because they further delineate the design problem. For example, the requirement statement "*The suspension preserves the steering characteristics of the bike*" has a primary object "*Steering Characteristic*" and a secondary object *Bike*.

5.2.4. Number of adjuncts

The specificity of functional requirements can be further improved through using adjuncts. For example, in the requirement statement "*the hand truck holds boxes securely on steep slopes*," *securely* is an `AdverbialAdjunct`, while *on steep slopes* is a `LocativeAdjunct`. A study of more than 200 well-defined requirements revealed that a typical requirement with a reasonable level of specificity has two adjuncts.

Existence of a measure adjunct. The requirements that contain a *`Measure Adjuncts`* are typically among the most informative requirements. For example, the requirement statement "*The suspension has a maximum vertical deflection at the seat mount of 8 mm at 250 lbs. static load*" is pointing to an important design variable (maximum vertical deflection) with important implications for the final design. This type of requirement is typically generated during the later stages of the concept design phase.

The specificity of functional and nonfunctional requirements is quantified through the specificity score. A continuous scale of 0 to 1 is used to represent the specificity score of the requirements. Table 9 shows the procedure used for calculating the specificity score of functional requirement. A similar procedure is used for nonfunctional requirements.

Table 10. *Example requirements with low, medium, and high specificity*

Requirement	Specificity Score	Specificity Level
1. The suspension is easy to install.	0.47	Low
2. The suspension fits a wide variety of tires.	0.59	Medium
3. The suspension allows easy traversal on slow difficult terrain.	0.75	High

The weights used in Eqs. (8) and (9) depend on the nature of the design project, and they are recommended to the user by the system. For example, for new design projects where the design is conceived functionally without any reference to physical forms, more weight is put on verbs. Using the proposed scoring method, requirements can be classified into three classes: highly specific (specificity score > 0.75), moderately specific ($0.75 >$ specificity score > 0.5), and not specific ($0.5 >$ specificity score). SWRL rules are used to determine the equivalent specificity class of a requirement. Table 10 shows examples of requirements for a bike suspension with different levels of specificity based on the proposed scoring method.

6. REQUIREMENT CLASSIFICATION

Automated requirement classification is another utility of ReqOn. Requirement classification and organization facilitates search and retrieval of requirements. An efficient and intelligent requirement search mechanism helps designers explore past design solutions that have addressed similar design problems. Based on Pahl et al.'s approach (1984) engineering requirements can be classified under different categories such as safety, performance, production, geometry, and ergonomics depending on their nature. For example, the requirements that belong to the *Kinematics* category typically deal with the type of motion of the product, direction of motion, velocity, and acceleration. As another example, *Environmental* requirements are the requirements that address the need for controlling the adverse environmental impacts of the product. Figure 8 shows the different categories of functional and nonfunctional requirements in this work.

6.1. Classification of nonfunctional requirements

The type of nonfunctional requirements is determined based on the type of the attribute used in the requirement. A two-step method is used for classifying nonfunctional requirements. The type of attribute is determined through a keyword matching approach, and then the requirements are classified using SWRL rules. A Java-based tool is developed for this purpose. The tool analyzes each instance of the `Attribute` class and then matches it with a set of predefined keywords. Examples of keywords pertaining to different attribute classes are shown in Table 11.

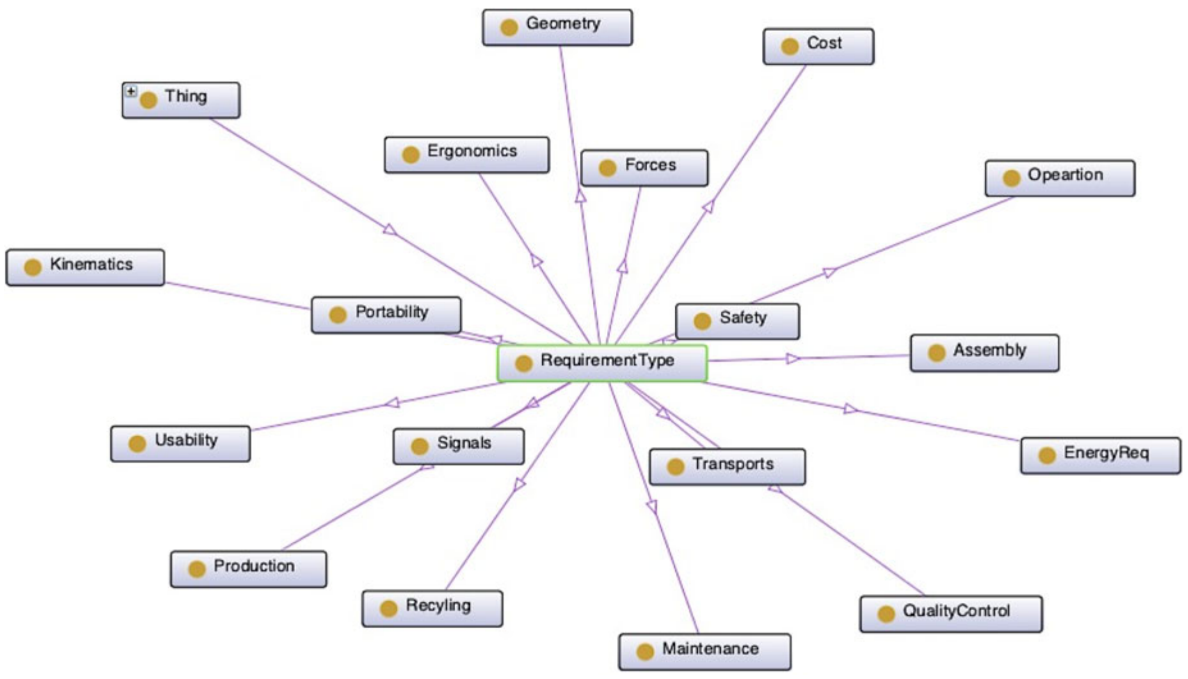


Fig. 8. Classification of requirement.

Once the attribute type is identified, SWRL rules will be used to infer the requirement type. Examples of such rules are given in Table 12. For example, if a nonfunctional requirement describes an attribute that belongs to the class *Size*, the requirement is categorized under the Geometry class. Therefore, requirement statement "*The frame of the ingot oven is not taller than 51/2 feet*" is classified as a geometric requirement.

6.2. Classification of functional requirements

For functional requirements, verbs are used for inferring the type of the requirement. Unlike attribute instances, verb instances do not need to be classified because their type is already defined at the time of instantiation in ReqOn. Therefore, SWRL rules can be directly applied to requirements for classification purpose. The developed tool parses the requirement statement in its ontological form and extracts the verb instance and identifies its type. For example, in the requirement statement "*The suspension traverse easily on a difficult terrain*," the verb *traverse* is an instance of the `Transport` class. As a result, through executing the appropriate SWRL rule, this requirement is classified as a *Transport* requirement. Examples of SWRL rule for functional requirement classification are given in Table 13.

Table 11. *Example keywords for selected attributes*

Attribute Subclass	Keywords	Related Requirement Type
Arrangement	Arrangement, display, setup, alignment, organization, order, group, etc.	Geometry
Capacity	Capacity, volume, etc.	Energy
Color	Color, paint, hue, tint, tone, shade, pigment, stain, dye, etc.	Ergonomics
Consumption	Consumption, expend, dissipation, utilization, etc.	Energy
Cooling	Cooling, cool, refrigerate, chill, cool off, cold, etc.	Energy
Deformation	Deform, deformation, buckle, contort, warp, impair, twist, distort, bend, deflect, out of shape, disfigure, etc.	Force
Heating	Heating, warm, reheat, warm up, heat up, etc.	Energy
Load	Load, cargo, consignment, goods, bundle, strain, etc.	Force
Pressure	Pressure, stress, force, thrust, etc.	Force
Portability	Portable, portability, mobile, mobility, movable, movability, adjustability, adaptability, etc.	Portability
Price	Cost, price, expense, charge, fee, fare, sum, amount, estimate, expenditure, etc.	Cost

Table 12. *Example of SWRL rules of requirement classification (nonfunctional)*

Rule 1
$NonFunctionalRequirement(?x) \wedge Size(?y) \wedge describesAttribute(?x, ?y) \rightarrow Geometry(?x)$
Rule 2
$Affordability(?y) \wedge NonFunctionalRequirement(?x) \wedge describesAttribute(?x, ?y) \rightarrow Cost(?x)$
Rule 3
$Price(?y) \wedge NonFunctionalRequirement(?x) \wedge describesAttribute(?x, ?y) \rightarrow Cost(?x)$
Rule 4
$Installation(?y) \wedge NonFunctionalRequirement(?x) \wedge describesAttribute(?x, ?y) \rightarrow Assembly(?x)$
Rule 5
$AssemblyMeasure(?y) \wedge NonFunctionalRequirement(?x) \wedge describesAttribute(?x, ?y) \rightarrow Assembly(?x)$

Table 13. *Example of SWRL rules of requirement classification (functional)*

Verb Class	Requirement Class	SWRL Rule
Distribute	Operation	$Distribute(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Operation(?x)$
Transport	Transport	$Transport(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Transports(?x)$
Display	Signal	$Signal(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Signals(?x)$
Actuate	Kinematics	$Actuate(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Kinematics(?x)$

7. CONCLUSIONS

In this work, a novel method for representation, evaluation, and classification of engineering requirements was introduced. The core technical contributions of this work are twofold: developing a comprehensive ontology for requirement representation based on OWL and developing quantitative methods and metrics for requirement evaluation supported by automated ontological reasoning. The tools, models, and methods developed in this work enable more intelligent management of engineering requirements particularly in the early stages of design process.

A Java-based automated tool was built to translate natural language requirement statements into OWL ontology. The tool is based on the linguistic structure of a requirement statement, and it was developed in such a way that the user does not require any knowledge of OWL ontology modeling to use it. Further, necessary methods and metrics to measure the information content of a requirement statement were established. A semiautomatic tool was created to measure the information content of a single requirement statement or a whole set of requirements for a product. The proposed information content metric can be used for evaluating the performance of design teams with respect to information generation rate. Furthermore, to evaluate the quality of a requirement statement, necessary metrics and rules were developed to measure the completeness and specificity of a requirement statement. A tool was developed to evaluate and assert the completeness and specificity of a requirement. In addition, a method for automated classification of requirements using ontological reasoning was introduced.

It should be noted that the methodologies presented in this work are tailored for requirement statements that already follow a semistructured syntax and grammar. For more unstructured texts and nontextual information, such as those found in technical standards or service guidelines, a more complete set of protocols should be developed. It can be argued that ontologies can be used for representation of different types of artifact because they model the world at the most abstract and conceptual level. Therefore, the proposed approach can be applied to other types of design artifacts, particularly textual design artifacts.

There are multiple possibilities for extension of this work in the future. The proposed ontology can be used for ensuring the consistency of product specifications and resolving potential conflicts and ambiguities. Further exterminations and analysis are required to study how the information content of requirements for a given product correlates with the complexity of the products. Although the proposed requirements ontology was developed to support intelligent requirement analysis, it could be used for enabling semantic information exchange and knowledge management and reuse during the requirement planning phase. A formal ontology with explicit semantics not only provides the requirement planning process with more structure but also facilitates retrieval and reuse of the requirements from similar design projects. If engineering requirements are mapped to different design features of the existing products in the design repository, designers can adopt the existing concepts, or their variations, to address new design problems.

Extension of the ontology defines another avenue for future work. The ontology is rich with respect to the vocabulary for functional requirements because it is based on the vocabulary of the FB, but the nonfunctional side of the ontology needs further expansion. In particular, there is a need for extending the `Attribute` class of the ontology and include a taxonomy that covers various type of attributes such as attributes durability, recyclability, serviceability, color, and ease of use. The ontology needs to be evolved continually such that it can capture the needs, desires, and beliefs of customers more accurately. Developing the methodology for ontology evolution, by itself, is research problem that needs to be addressed independently.

ACKNOWLEDGEMENT

Funding for this research was provided by the National Science Foundation under Award CMMI-1334259.

REFERENCES

Chandrasegaran, S.K., Ramani, K., Sriram, R.D., Horváth, I., Bernard, A., Harik, R.F., & Gao, W. (2013). The evolution, challenges, and future of knowledge representation in product design systems. *Computer-Aided Design 45(2)*, 204–228.

Collopy, P.D., & Eames, D.J.H. (2001). Aerospace manufacturing cost prediction from a measure of part definition information. *Proc. SAE World Aviation Congr.—2001 Aerospace Congr.*, Seattle, WA, September 10–14.

Darlington, M.J., & Culley, S.J. (2008). Investigating ontology development for engineering design support. *Advanced Engineering Informatics 22(1)*, 112–134.

Hauge, P.L., & Stauffer, L.A. (1993). ELK: a method for eliciting knowledge from customers. *ASME Design Engineering 53*, 73–81.

Hirtz, J., Stone, R.B., McAdams, D.A., Szykman, S., & Wood, K.L. (2002). A functional basis for engineering design: reconciling and evolving previous efforts. *Research in Engineering Design-Theory Applications and Concurrent Engineering 13(2)*, 65–82. doi:10.1007/S00163-001-0008k-3

Jianxin, J., & Chun-Hsien, C. (2006). Customer requirement management in product development: a review of research issues. *Concurrent Engineering: Research and Applications 14(3)*, 173–185. doi:10.1177/1063293X06068357

Joshi, S., & Summers, J.D. (2014). Tracking project health using completeness and specificity of requirements: a case study. *Proc. ASME 2014 Int. Design Engineering Technical Conf./Computers and Information Engineering Conf.*, Buffalo, NY, August 17–20.

Jureta, I.J., Mylopoulos, J., & Faulkner, S. (2009). A core ontology for requirements. *Applied Ontology 4(3–4)*, 169–244. doi:10.3233/ao-2009-0069

Kossmann, M., Odeh, M., Wong, R., Gillies, A., & IEEE. (2008). Ontology-driven requirements engineering: building the OntoREM meta model. *Proc. 2008 3rd Int. Conf. Information and Communication Technologies: From Theory to Applications*, pp. 1378–1383, Damascus, Syria, April 7–11.

Lamar, C., & Mocko, G.M. (2010). Linguistic analysis of natural language engineering requirement statements. *Proc. 8th Int. Symp. Tools and Methods of Competitive Engineering, TMCE 2010*, Ancona, Italy, April 12–16.

Lin, J., Fox, M.S., & Bilgic, T. (1996). A requirement ontology for engineering design. *Concurrent Engineering 4(3)*, 279–291.

Micouin, P. (2008). Toward a property based requirements theory: system requirements structured as a semilattice. *Systems Engineering 11(3)*, 235–245. doi:10.1002/sys.20097

Mir, M.S., Agarwal, N., & Iqbal, K. (2011). Applied ontology for requirments engineering: an approach to semantic integration of requirements model with system model. *Proc. 15th IASTED Int. Conf. Software Engineering and Applications, SEA 2011*, Dallas, TX, December 14–16.

Morkos, B., Shankar, P., & Summers, J.D. (2012). Predicting requirement change propagation, using higher order design structure matrices: an industry case study. *Journal of Engineering Design 23(12)*, 905–926.

Pahl, G., Beitz, W., & Wallace, K. (1984). *Engineering Design*: London: Design Council.

Qureshi, N.A., Jureta, I.J., & Perini, A. (2011). Requirements engineering for self-adaptive systems: core ontology and problem statement. *Proc. 23rd Int. Advanced Information Systems Engineering Conf., CAiSE 2011*, Berlin, June 20–24.

Schatz, B., Fleischmann, A., Geisberger, E., & Pister, M. (2005). Model-based requirements engineering with AutoRAID. *Proc. 35th Jahrestagung der Gesellschaft fur Informatik e.V. (GI): Informatik LIVE!, INFORMATIK 2005* [35th Annual Conf. German Informatics Society (GI): Informatics LIVE!, INFORMATIK 2005], Bonn, September 19–22.

Sen, C., Caldwell, B.W., Summers, J.D., & Mocko, G.M. (2010). Evaluation of the functional basis using an information theoretic approach. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing 24(1)*, 87–105.

Shannon, C.E. (1948). *A Mathematical Theory of Communication* (Vol. 5). New York: ACM.

Tseng, M.M., & Jianxin, J. (1998). Computer-aided requirement management for product definition: a methodology and implementation. *Concurrent Engineering: Research and Applications 6(2)*, 145–160. doi:10.1177/1063293X9800600205

Yan, W., Chen, C.H., & Khoo, L.P. (2001). A radial basis function neural network multicultural factors evaluation engine for product concept development. *Expert Systems 18(5)*, 219–232.

Yannou, B. (2012). Requirements management within a full model-based engineering approach. *Systems Engineering 15(2)*, 119–139. doi:10.1002/sys.20198

Zhang, Y., & Zhang, W. (2007). Description logic representation for requirement specification. *Proc. 7th Int. Conf. Computational Science, ICCS 2007*, Beijing, May 27–30.

Alolika Mukhopadhyay is currently a PhD student in the Department of Mechanical and Industrial Engineering at Northeastern University. She received her MS degree in technology management from Texas State University.

Farhad Ameri is an Associate Professor of manufacturing engineering and technology in the Department of Engineering Technology at Texas State University and the head of the Engineering Informatics (INFONEER) Research Group. He received his doctoral degree in manufacturing engineering from the University of Michigan. Dr. Ameri's research interests include design and manufacturing informatics, smart manufacturing systems, and design theory and methodology.