AN ONTOLOGICAL APPROACH TO ENGINEERING REQUIREMENT REPRESENTATION AND ANALYSIS

by

Alolika Mukhopadhyay, B.TECH.

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Technology Management
August 2015

Committee Members:

Farhad Ameri, Chair

Vedaraman Sriraman

Jaymeen Shah

COPYRIGHT

by

Alolika Mukhopadhyay

2015

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

This work is protected by the Copyright Laws of the United States (Public Law 94-553, section 107). Consistent with fair use as defined in the Copyright Laws, brief quotations from this material are allowed with proper acknowledgment. Use of this material for financial gain without the author's express written permission is not allowed.

Duplication Permission

As the copyright holder of this work I, Alolika Mukhopadhyay, refuse permission to copy in excess of the "Fair Use" exemption without my written permission.

DEDICATION

I would like to dedicate my thesis to my husband and my family. My husband Shubhankar has always been there for supporting me and encouraging me through all the challenges of graduate life. I am truly thankful for having him in my life. I want to extend my feeling of gratitude towards my Dad for being my inspiration and my Mom for her endless love, and care. I also want to dedicate my thesis to my sister Arunima, who never left my side and always cheered me up.

ACKNOWLEDGEMENTS

I have no words to express my gratitude towards Dr. Ameri for being such a wonderful supervisor. I am thankful to him for giving me the opportunity to work in his lab. I consider myself blessed to have him as my advisor. He always believed in me and encouraged me to perform my best. He will always inspire me by being an outstanding teacher, a great researcher and above all a great human being.

I would like to thank my thesis committee members Dr. Sriraman for enlightening me the glance of research and Dr. Shah for his valuable insights and guidance.

I would also like to thank Dr. Chiradeep Sen for his support, stimulating discussions and all the suggestions he provided to accomplish my research. I thank my fellow lab mates in Texas State for being with me all the time.

Finally, I want to recognize that this research would not have been possible without the assistance of Texas State University, Department of Engineering Technology at TXST, and the National Science Foundation, and express my gratitude to them.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF ABBREVIATIONS

Abbreviation	Description
API	Application Program Interface
OWL	Web Ontology Language
OWL DL	Web Ontology Language (Description Logic)
ReqOn	Requirement Ontology
SWRL	Semantic Web Rule Language

ABSTRACT

The objective of this research is to develop an ontological method for measuring the information content of engineering design requirements, assessing their completeness and specificity, and automatically classifying them under predetermined classes. Information content can be used as a metric for evaluating the performance of engineering design teams with respect to information generation rate. Requirement plays an important role into idea generation. An incomplete requirement is not useful for a designer and might be misleading and also requirements should be specific or informative enough to efficiently narrow down the design space. A two-step method will be proposed for information content measurement. First, the textual requirements will be converted into an ontological representation and then the information metric will be applied to them. A Java-based tool will be developed for the automated measurement of the information content of requirements based on their ontological representation and proposed metrics. Also ontological reasoning techniques based semantic rules and axioms will be adopted for evaluating completeness and specificity of engineering requirements and classifying them under predefined classes. Multiple experiments will be designed and conducted to validate the proposed methods and metrics.

CHAPTER 1

RESEARCH PERSPECTIVES

1.1 Introduction:

Engineering requirements describe the conditions and capabilities that a design artifact should meet in order to satisfy implicit and explicit customer needs. 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 adequacy of the generated requirements in terms of specificity and completeness is often appraised on a consensual basis. Additionally, the requirements are usually represented informally in plain English without following any standard protocol or vocabulary. Even in the same company, different design teams may follow different methods and conventions for representing engineering requirements. In the absence of formal methods and models for engineering requirement representation, organization, search, and retrieval of requirements becomes inefficient and tedious.

The objectives of this research are: 1) to develop a formal ontology for standard representation of engineering requirements, 2) to develop the method and metrics necessary for measuring the information content of engineering requirements, and 3)

using ontological reasoning techniques for measuring the specificity and complexity of the requirement statement and requirement classification.

1.1.1 Motivations for Information Content Measurement:

One of the main focuses of this work is on developing the necessary metrics for measuring the information content of engineering requirements. A reliable set of metrics for information content measurement can be used for evaluating the performance of engineering design teams. The concept of measuring the information transferred over a channel using probability theory is well established and 'bit' is used as a unit for the measurement. However, it is not sufficient to measure the pragmatic information content of design documents and artifacts produced during the design process. Design is an iterative and complex process and consists of a series of converging and diverging processes. In novel product design projects, the steps of the design process are not preordained, they are iterative and complex, and they usually vary between design problems. Consequently, the information transforms are also not easily visible. At the beginning of the design, which is the planning phase, the level of uncertainty remains at its summit. This uncertainty rises due to the lack of information. As the design unfolds, the designer gathers more information about the design space and possible solutions and eventually uncertainty goes down to zero and the design terminates when enough information is collected for determining all the design variables. It will be useful if uncertainty reduction (or information growth) rate during product development process can be quantified and visualized. This will help project managers evaluate and compare the performance of engineering design teams based on a measure that is directly related

to the inherent capabilities of the design teams such as learning, information processing and knowledge generation. To this end, it is necessary to measure the information content of the various design artifacts, such as requirements, function models, CAD models, graphs, sketches and product layouts in a formal and replicable manner. The focus of this work is on the artifacts that are represented textually such as engineering requirement. The proposed methods and metrics in this work should be applicable to all types of textual documents.

An information-theoretic approach is adopted for evaluating the performance of the requirement planning phase. Since 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. It is not always easy to track the information transformation process or fully understand its dynamics since it is a complex phenomenon involving multiple domain experts with varying levels of expertise and experience. However, the amount of information generated at different timestamps during the design process should be quantifiable. More effective design teams generate and transform information at a faster rate. Also, the information generated by them is of higher value for downstream processes and results in better decisions. Design information is embodied in design artifacts. Hence, in order to study information transformation rate, first, the information content of the artifacts that are being evolved throughout the design projects 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.

1.1.2 Motivations for Measuring Specificity and Completeness:

Requirement Engineering is one of the crucial steps in Engineering Design. Requirements reflect the desired functions and characteristics of the product. A set of requirements delineates the constraints and criteria that define the design space. A requirement statement should be written in a clear and unambiguous way and provides enough details such that the designer can develop a solid understanding of feasible design solutions. Imprecise and ambiguous requirements can lead to poor designs. In spite of having significant impact on design decisions, requirement planning are often neglected by the designer and conducted in a discrete manner without much structure. As a result, requirements are often subject to incompleteness, inconsistency, and conflict. The adequacy of the generated requirements in terms of specificity and completeness is often appraised subjectively. But it is very important to reduce the ambiguity and inconsistencies in the earlier stages in order to avoid huge cost to fix the design in the advanced stages. Therefore, it is necessary to develop the methods and metrics needed for evaluating the completeness and specificity of engineering requirements. As the size and the number of design projects increase in a design organization, the number, complexity, and variety of the requirements statements increase as well. Therefore, the proposed methods for requirement evaluation should lend themselves to automation in order to enable designers to quickly evaluate large sets of requirements.

1.1.3 Motivations for Requirement Classification:

Over the years, requirement planning for software development has matured to the point where they are well understood and have proper taxonomy for classification. The developers have the proficiency to categorize them into various categories and reuse them if necessary. This is not the case in engineering design. Requirement categorization facilitates search and retrieval of requirement. Also it enables designers to learn how the past designs have addressed similar design problems. Requirements can be classified under different categories such as safety, performance, production, service, and agronomy. Requirement written in a textual format do not lend themselves to automatic classification. If requirements are represented ontologically, they can be automatically classified, thus improving search-ability and reusability.

1.2 Research Question:

The underlying research questions for this work include:

- What are the components of a formal ontology for requirements modeling?
- What is a good metric for measuring the information content of engineering requirements?
- How may the completeness and specificity of a requirement statement be measured using the formal requirement ontology?
- How to use ontological reasoning techniques to classify requirements?

1.3 Methodologies:

A design problem can be represented in many different ways. In contrast, the solution conjecture may also take different forms such as sketch, text or graphs etc. In

this research, one of the key assumptions is Form-neutrality, which means the information contained in different design artifacts is same irrespective of their type. Therefore, all design artifacts need to be converted into a form neutral representation without loosing any information. In this research the focus is on requirement statements and an OWL ontology will be developed for ontology representation

A three-staged method is used to classify the requirements and measure their information content, specificity and completeness. The first stage involved in this method is aimed at identifying different components of a requirement statement and developing an OWL ontology for requirement modeling. A set of two or three requirements clustered together, is broken down into separate requirements and further elements of each requirement statement is dispersed. Parts of speech tagging is used to spot the building blocks of requirement statements. Depending on different identified elements of the requirement statement, a formal structure of the ontology is developed at this point. Sentence structure of numerous requirements is also being studied to verify the ontology structure. The second stage takes into account the way of developing necessary method and metrics to measure the information content of a single requirement or a set of requirements. Shannon's information metric based on used vocabulary is applied to determine the metrics. After translating the requirements into OWL ontology different containers of information such as entities, relations and attributes are recognized and textual requirements are exposed from the plain English sentences. At this stage the Shannon's information metric is applied to them to compute the information content in bits scale. A semi automatic java tool using ontology and OWL API is also being

developed to do the task semi-automatically. The last stage is focused on measuring the quality of the requirements in terms of specificity and completeness. With the purpose of measuring requirements quality, SWRL rules are added to the ontology. An OWL API interface is also being used to reason through the requirements and return a specificity and completeness score for each requirement statement. Finally all these stages will add up to a syntax for functional and nonfunctional requirements, a metric for measuring information content and query based system to measure the specificity and completeness of the requirements. An experimental validation is also used to examine the accuracy of the developed ontology model.

1.4 Research Tasks:

The work included for this research is broken down into 6 tasks as listed below:

Requirement Modeling

Task 1. Creating the formal ontology model using protégé

Task 2. Populating the ontology with various requirements to validate the structure of the ontology

Task 3. Developing the metric to measure the information content, completeness and specificity

Task 4. Implementation of the metrics

Task 5. Developing a tool to create an interactive interface to measure the Information content using JAVA and OWLAPI

Task 6. Experimental Validation

1.5 Assumptions:

General assumptions of this research are:

1. Form neutrality of information is one of the key assumptions in this study and therefore, a design artifact such as a requirement statement conveys the same amount of information to the designer irrespective of it form such as text or sketch, and that this amount could be measured
2. The information entropy of each class of the ontology depends on the structure of the class and the number of instances of the class that are available in the ontology at any given timestamp.
3. All nodes in the ontology follow a uniform distribution that means they have equal likelihood of occurrence.

1.6 Limitation:

- The metrics proposed in this work provide relative measures of the information content.

1.7 Delimitations:

- The focus of this work is on the artifacts that are represented textually such as engineering requirement.
- The products that are included in this study are typical consumer products with low-medium complexity.

1.8 Preliminary Results:

This section provides a high-level overview of a draft ontology developed for formal representation of engineering requirements. The primary objective of developing

the Requirement Ontology (ReqOn) is not to create a comprehensive vocabulary that provides a complete coverage of all terms and relations used for describing engineering requirements of various products. Rather, the objective is to create a formal representation amenable to automated information content measurement. With this objective in mind, ReqOn is designed such that it can break a requirement statement into its elemental containers of information. These containers are essentially ontology classes or concepts. A linguistic and grammatical approach is adopted for ontology conceptualization. Therefore, parts of speech (verbs and nouns) and grammatical functions (subject, object, complement, and adjuncts) define the major classes of the ontology.

The scope of ReqOn is currently limited to consumer products with medium complexity. However, ReqOn can be evolved into a comprehensive design requirements ontology in the future that could be used for communicating design requirements among product stakeholders. An evolutionary approach was adopted for developing ReqOn starting with a flat ontology and then categorizing the instances into appropriate groups and eventually forming class-subclass relationships to increase the taxonomical depth of the ontology. In ReqOn, each requirement statement is represented by the Requirement class, which has two disjoint subclasses, namely, Functional Requirement and Nonfunctional Requirement. A functional requirement describes the functions and behaviors of the product, whereas nonfunctional requirements describe the attributes of the product such as size, color, or recyclability. In order to define the structure of the requirement ontology different types of requirements of various consumer products are

collected and their basic components are identified. Components of different types of functional requirements are shown in Table 1 and Table 2 shows the components of different nonfunctional requirements. Based on the similarities in structure of these requirement statements, the structure of the formal requirement ontology could be defined.

Table 1: Different types of Functional Requirement

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
Product	Electric Kettle
Primary Subject	Left-handed User
Verb	handles
Object	Electric kettle
Adverbial Adjunct	easily
Example 2: Electric Wok has a handle that the user grips easily.	
Product	Electric Kettle
Subject	user
Verb	grips
Object	handle
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

Table 2- Continued: Different types of Nonfunctional Requirements

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
hasMaterial	plastic

CHAPTER 2

REQUIREMENT ONTOLOGY

2.1 Introduction:

Conceptualization of a set of objects is abstract, but ontology is the explicit definition of these objects and their interconnectivity (Ameri & Summers, 2008). Ontologies are the single integrated view of a particular knowledge domain and they are analogous to the conceptual schema of a database system. Conceptual schema is the theoretical definition of the whole project and it contains set of concepts, their relations and a set of assertion regarding their nature (Halpin, 1996). Conceptual schema allows software applications to access the data without sharing the structure of the data. Similarly, ontologies also provide semantic interoperability and logical reasoning ability (Noy & McGuinness, 2001). In semantic web, ontologies have been used for so many applications but in engineering design the use of ontologies are relatively new. Therefore the main objective of this study is to develop formal ontology for standard representation of engineering requirements and also to utilize the ontology to study various characteristics of requirement statements using software applications. This chapter presents the main framework of the ontology, definition of all essential elements of the ontology, the association among the elements, and the implementation of these concepts to build the ontology.

2.2 Background - Engineering Requirements Modeling:

Engineering requirements describe the attributes, behaviors, and functionalities a product must fulfill to satisfy the needs of multiple stakeholders including manufacturing engineers, sales and service staff, and end users. Engineering requirements are derived from customer need statements. Customer need statements, represented in natural language, are often imprecise and ambiguous and contain contradictory information (Tseng & Jiao, 1998a). Researchers have developed formal models with the objective of improving the process of requirement elicitation, analysis, communication, validation, and reuse. However, there is no definite structure for requirement modeling and representation in engineering design (Jiao & Chen, 2006). 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 (Kossmann, Wong, Odeh, & Gillies, 2008; Mir, Agarwal, & Iqbal, ; Qureshi, Jureta, & Perini, 2011). Although requirement modeling in software engineering has fundamental differences with that in engineering design, there are some ontologies in software engineering domain, such as CORE(Qureshi et al., 2011), that can be applied to engineering design due to their high-level conceptualization. CORE is based on DOLCE (Gangemi, Guarino, Masolo, Oltramari, & Schneider, 2002), a foundational ontology that contains even more general concepts that are the same across all knowledge domains.

Lamar (Lamar, 2009) 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. Lamar decomposes 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 (Morkos, Shankar, & Summers, 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's model include subject, modifier, verb (modal and transitive), object and condition.

Lin et al. (Lin, Fox, & Bilgic, 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 (Darlington & Culley, 2008), proposed 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 its elemental components. Other related works in requirement representation include the requirement taxonomy(Hauge & Stauffer, 1993), the customer attribute hierarchy (Yan, Chen, & Khoo, 2001) and the functional requirement topology(Tseng & Jiao, 1998b).

2.3 Requirement Ontology (ReqOn):

Uschold and King Ontology Building method is adopted to develop the ReqOn (Uschold & King, 1995). This 'skeletal' methodology combines four ad hoc processes to build the ontology.

- Identify Purpose and Scope
- Build Ontology
    - Ontology Capture
    - Ontology Coding
    - Ontology Integration
- Evaluate Ontology
- Document Ontology

2.3.1 Identify Purpose and Scope of the Ontology:

The primary objective of developing the Requirement Ontology (ReqOn) is not to create a comprehensive vocabulary that provides a complete coverage of all terms and relations used for describing engineering requirements of various products. Rather, the

objective is to create a formal representation amenable to automated information content measurement. With this objective in mind, ReqOn is designed such that it can break a requirement statement into its elemental containers of information. These containers are essentially ontology classes or concepts. A linguistic and grammatical approach is adopted for ontology conceptualization (Morkos et al., 2012). Therefore, parts of speech (verbs and nouns) and grammatical functions (subject, object, complement, and adjuncts) define the major classes of the ontology. The scope of ReqOn is currently limited to consumer products with medium complexity. However, ReqOn can be evolved into a comprehensive design requirements ontology in the future that could be used for communicating design requirements among product stakeholders. An evolutionary approach was adopted for developing ReqOn starting with a flat ontology and then categorizing the instances into appropriate groups and eventually forming class-subclass relationships to increase the taxonomical depth of the ontology.

2.3.2 Build Ontology:

2.3.2.1 Ontology Capture:

In ReqOn, each requirement statement is represented by the `Requirement` class, which has two disjoint subclasses, namely, `FunctionalRequirement` and `NonFunctionalRequirement`. A functional requirement describes the functions and behaviors of the product, whereas nonfunctional requirements describe the attributes of the product such as size, color, or recyclability. The concept diagrams in [Figure 1] and [Figure 2] show the properties of the `FunctionalRequirement` and `NonFunctionalRequirement` classes respectively.

Both types of requirements inherit `hasProduct` and `isFunctional` properties from their common super-class (i.e., `Requirement`). `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. `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, Caldwell, Summers, & Mocko, 2010a) .The `Behavior` class captures this process by encoding the predicate elements of the requirement statement. 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 embedded in the behavior. Each behavior has exactly one action verb, either transitive or nontransitive. Since the Functional Basis (FB) (Sen et al., 2010a) 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`. `

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 has its `hasProduct` property set to the wheel of a bicycle, then the wheel is the primary

subject and the bicycle is the secondary subject, since the requirement directly applies to the wheel and not to the bicycle. Each requirement statement that has a transitive verb needs a primary object and may have one or more secondary objects. Both object and subject can have different types such as user, product, part, material, energy, or signal. For example, consider the reqirement "The Electric Wok has a lid that can be flipped easily". In this requirement statement "flip "is `TransitiveVerb`, "lid" is `PrimaryObject`, and electric wok is `Object`. The taxonomy of the material, energy, and signal classes are directly imported from FB. It is important to incorporate various class hierarchies in ReqOn since it allows for more accurate evaluation of the information content based on the depth of classes in the hierarchy.

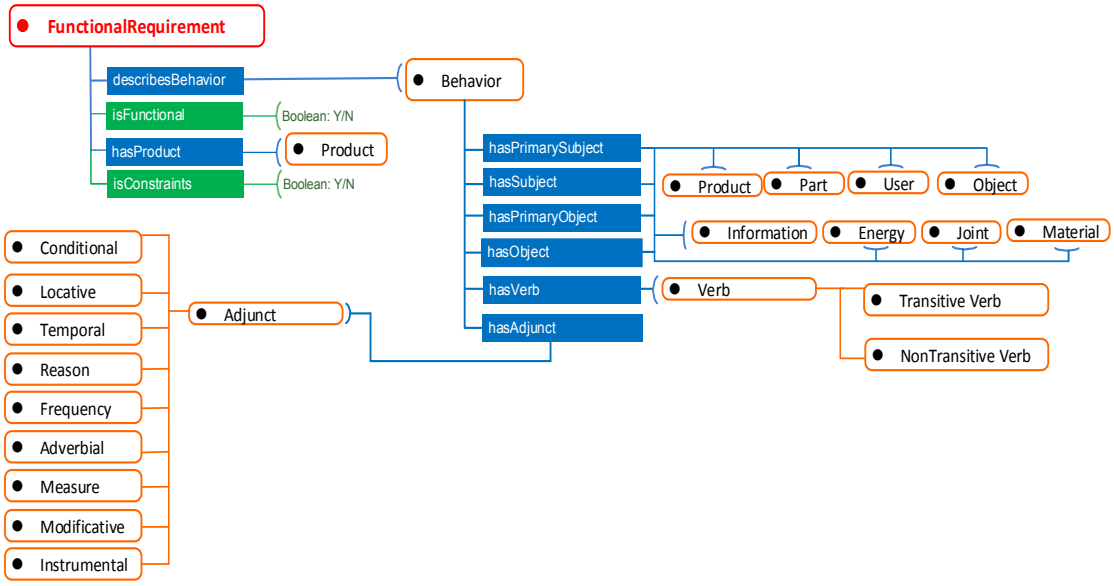

Figure 1: Concept diagram for the FunctionalRequirement Class

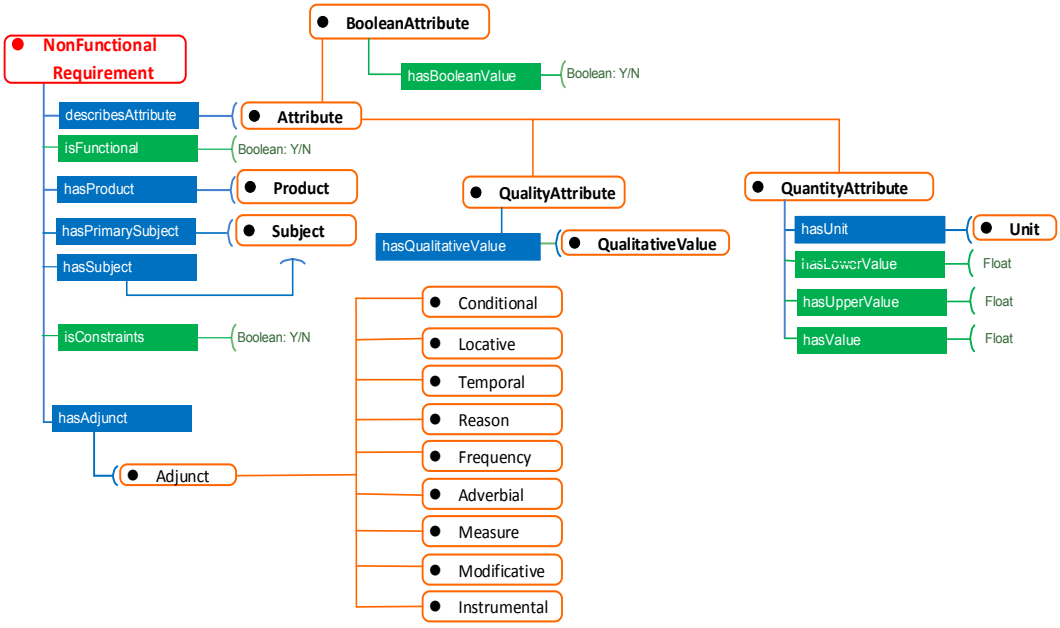

Figure 2: Concept diagram for the NonFunctionalRequirement Class

The specificity of functional requirements can be improved by using Adjunct. An adjunct usually modifies the 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 AdverbialAdjunct while "on steep slopes" is a LocativeAdjunct. "Hand truck" and "box" are subject and object respectively. As the requirements evolve, designers add more details in the requirement through introducing various types of adjuncts. [Table 1] describes three possible types of functional requirement that can be represented in ReqOn.

A NonFunctionalRequirement uses the describesAttribute 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. Protocol for creating requirement in ReqOn [Table 3] describes the protocol for identifying different components of requirement statements and converting them into ontological representation.

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". Uses 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.

Table 3-Continued: Protocol for creating requirement in ReqOn

3- Identifying requirements components:

a. Product: *It is the product for which the requirement is defined*.

b. Subject: *The subject associated with a verb. It is either the product itself, or a part of the product, or the user. In a non-functional requirement, subject is the entity that the attribute pertains to.*

   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, since it performs the action "making hole" directly.

c. 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.

d. Verb: *describes the action*. It can be transitive or intransitive. Requirements with linking verbs (such as "is and "are") are often represented as non-functional requirements (Example: Bicycle "is" easy to repair).

e. 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.

f. Attribute: *A feature, property, quality, or component related to the product or its parts.*

   i. A quality attributes 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).

2.3.2.2 Ontology Coding:

Ontology coding represents implementation of the concept developed during ontology capture phase. This stage involves selecting a representation language and writing the codes (Ameri, Urbanovsky, & McArthur, 2012; Gruber, 1993). In this study OWL DL (DL stands for "Description Logic") is selected as the ontology representation

language. OWL DL is a subset of OWL ("Web Ontology Language"), which allows reasoning engines to support reasoning and can detect semantic inconsistencies in the ontology. It also supports SPARQL queries to sort the data.

Syntax of description logic is based on simple mathematical logics such as subset, union, intersection, universal and existential concept etc. (Ameri & Summers, 2008). Identifying semantic structure, classes, class hierarchies, object and data properties, instances etc. creates ReqOn. In this section the core components of the ReqOn is defined formally using OWL DL and necessary axioms are also provided. Definition of some core concept included in the ontology is shown in [Table 4]. Each class in the ontology represents a unique concept and each property mirrors the association among these concepts. To uniquely identify each concept a set of OWL Ontology Class Axioms, OWL Ontology Property Axioms and OWL DL Restrictions are required.

Table 4: Definition of some of the core classes in ReqOn

Concept	Definition
Functional Requirement	A requirement that describes operation and activities that the product has to perform.
Behavior	A component of functional requirement through which the function is achieved
Subject	Subject is the product, part, user, material, energy or information that is doing or being something
Object	Object is the product, part, user, material, energy or information, which is acted upon by the subject
Transitive Verb	Transitive verb is the verb, which takes at least one object
NonTransitive Verb	NonTransitive verbs do not have any object
Nonfunctional Requirement	Nonfunctional requirements are requirements that describe one or more characteristics of the product

Table 4 - Continued: Definition of some of the core classes in ReqOn

Concept	Definition
Attribute	Attributes are concepts that do not exist on their own; rather, they are parameters that characterize the subject
Value	Value indicates the magnitude of an attribute. Values can be quantitative (e.g., "85 °C") or qualitative (e.g., "high" temperature), and thus, they can occur in the text as nouns, adjectives, or adverbs.
Adjunct	Adjuncts are optional and usually modifies the verb and indicates the time, manner, place, frequency, reason, degree, or condition pertaining to the requirement

2.3.2.2.1 Ontology Building Blocks:

Definition (1): `Requirements` are classified into two disjoint classes and they are completely constraint. `Requirement` is a statement that is associated with exactly one product, it can be functional or nonfunctional and it can be a constraint or criteria.

$$Requirement = Statement \cap (\exists hasProduct.Product) \cap (\forall isFunctional.xsd{:}\,boolean\ or\ isNonFunctional.xsd{:}\,boolean) \cap (\forall isConstrain.xsd{:}\,boolean\ or\ isCriteria.xsd{:}\,boolean)$$

$$< owl{:}\,Class\ rdf{:}\,ID = "Requirement"/ >$$
$$< owl{:}\,Class\ rdf{:}\,ID = "FunctionalRequirement"/ >$$
$$< rdfs{:}\,subclassOf\ rdf{:}\,resource = "\#Requirement\ " >$$
$$</\text{owl: Class} >$$
$$< owl{:}\,Class\ rdf{:}\,ID = "NonFunctionalRequirement"/ >$$
$$< rdfs{:}\,subclassOf\ rdf{:}\,resource = "\#Requirement\ " >$$
$$</\text{owl: Class} >$$
$$< owl{:}\,Class\ rdf{:}\,ID = "Product"/ >$$
$$< owl{:}\,ObjectProperty\ \ rdf{:}\,ID = "hasProduct"/ >$$
$$< rdf{:}\,type\ rdf{:}\,resource = "\&owl;FunctionalProperty"\ / >$$
$$< rdfs{:}\,Domain\ \ \ rdf{:}\,resource = "\#Requirement\ " >$$

```
        < rdfs: Range     rdf: resource = "#Product " >
</owl: ObjectProperty >
< owl: DatatypeProperty  rdf: ID = "isFunctional"/ >
        < rdfs: Domain   rdf: resource = "#Requirement " >
        < rdfs: Range     rdf: resource = "& xsd; boolean " >
</owl: DatatypeProperty >
< owl: DatatypeProperty  rdf: ID = "isConstrain"/ >
        < rdfs: Domain   rdf: resource = "#Requirement " >
        < rdfs: Range     rdf: resource = "& xsd; boolean " >
</owl: DatatypeProperty >
```

Definition (2): `Functional requirement` inherits all characteristics form its parent class requirement. Functional requirement describes exactly one behavior of the product.

```
Functional Requirement
            = Requirement ∩ (= 1 describesBehavior. Behavior)
            ∩ (∀isFunctional. true )

< owl: Class rdf: ID = "Behavior"/ >
< owl: ObjectProperty  rdf: ID = "describesBehavior"/ >
        < rdf: type rdf: resource = "&owl; FunctionalProperty" />
        < rdfs: Domain   rdf: resource = "#FunctionalRequirement " >
        < rdfs: Range     rdf: resource = "#Behavior" >
                          </owl: ObjectProperty >
```

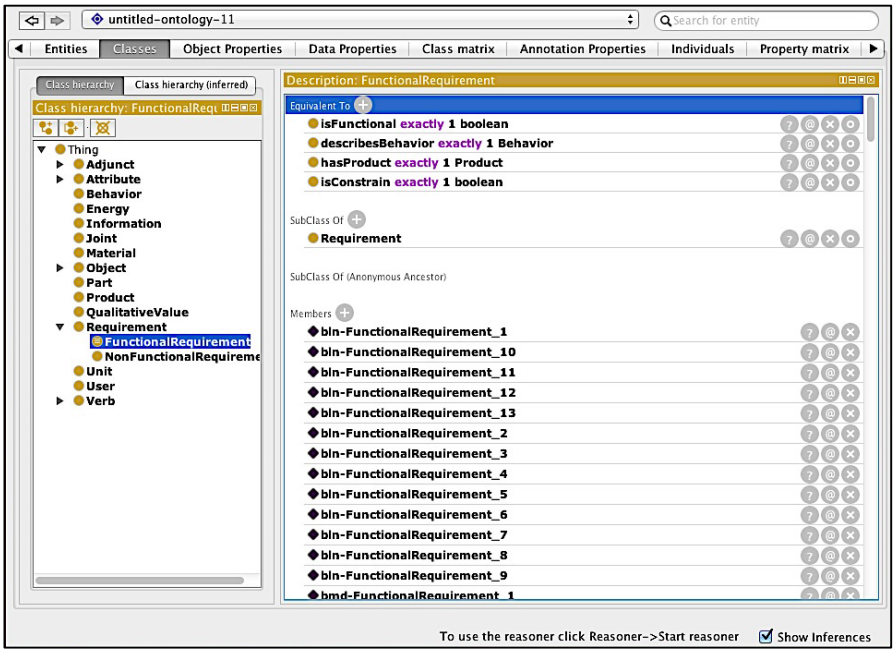

Figure 3: Screenshot of Class expression editor for Functional Requirement in Protégé 4.3

Definition (3): *Behavior* is a component of *functional requirement* that contains exactly one primary subject, some secondary subject, exactly one primary object, some secondary object, exactly one action verb and some adjuncts.

$$Behavior = Component \cap (= 1\ hasPrimarySubject.Subject) \cap (= 1\ hasPrimaryObject.Object) \cap (\exists hasSecondarySubject.Subject) \cap (\exists hasSecondaryObject.Object) \cap (= 1\ hasVerb.Verb) \cap (\exists hasAdjunct.Adjunct)$$

$$< owl{:}Class\ rdf{:}ID = "Part" / >$$
$$< owl{:}Class\ rdf{:}ID = "Material" / >$$
$$< owl{:}Class\ rdf{:}ID = "Joint" / >$$
$$< owl{:}Class\ rdf{:}ID = "Energy" / >$$

```
< owl: Class rdf: ID = "Information"/ >
< owl: Class rdf: ID = "User"/ >
< owl: Class rdf: ID = "Verb"/ >
< owl: Class rdf: ID = "Adjunct"/ >

< owl: Class rdf: ID = "Subject"/ >
      < owl: unionOf rdf: parseType = Collection >
          < owl: Class rdf: about = "#Part"/ >
          < owl: Class rdf: about = "#Material"/ >
          < owl: Class rdf: about = "#Joint"/ >
          < owl: Class rdf: about = "#Energy"/ >
          < owl: Class rdf: about = "#Information"/ >
          < owl: Class rdf: about = "#User"/ >
          < owl: Class rdf: about = "#Product"/ >
      </owl: unionOf >
</owl: Class >

(Similarly, object is also union of Part, Product,
Material, Joint, Energy, Information and User.)

< owl: ObjectProperty rdf: ID = "hasPrimarySubject"/ >
          < rdf: type rdf: resource = "&owl; FunctionalProperty" />
          < rdfs: Domain   rdf: resource = "#Behavior" >
          < rdfs: Range    rdf: resource = "#Subject " >

</owl: ObjectProperty >
< owl: ObjectProperty rdf: ID = "hasPrimaryObject"/ >
          < rdf: type rdf: resource = "&owl; FunctionalProperty" />
          < rdfs: Domain   rdf: resource = "#Behavior" >
          < rdfs: Range    rdf: resource = "#Object " >
                          </owl: ObjectProperty >
< owl: ObjectProperty rdf: ID = "hasSecondarySubject"/ >
          < rdfs: Domain   rdf: resource = "#Behavior" >
          < rdfs: Range    rdf: resource = "#Subject " >
</owl: ObjectProperty >
< owl: ObjectProperty rdf: ID = "hasPrimarySubject"/ >
          < rdf: type rdf: resource = "&owl; FunctionalProperty" />
```

```
        < rdfs: Domain   rdf: resource = "#Behavior" >
        < rdfs: Range    rdf: resource = "#Subject " >
</owl: ObjectProperty >
< owl: ObjectProperty  rdf: ID = "hasSecondaryObject"/ >
        < rdfs: Domain   rdf: resource = "#Behavior" >
        < rdfs: Range    rdf: resource = "#Object " >
</owl: ObjectProperty >
< owl: ObjectProperty  rdf: ID = "hasVerb"/ >
        < rdf: type rdf: resource = "&owl; FunctionalProperty" />
        < rdfs: Domain   rdf: resource = "#Behavior" >
        < rdfs: Range    rdf: resource = "#Verb" >
</owl: ObjectProperty >
< owl: ObjectProperty  rdf: ID = "hasAdjunct"/ >
        < rdf: type rdf: resource = "&owl; FunctionalProperty" />
        < rdfs: Domain   rdf: resource = "#Behavior" >
        < rdfs: Range    rdf: resource = "#Adjunct" >
</owl: ObjectProperty >
```

Definition (4): `Nonfunctional requirement` also inherits all properties of requirement. Nonfunctional requirement contains exactly one primary subject, exactly one primary object, some secondary subject, some secondary object, some adjunct and minimum one attribute.

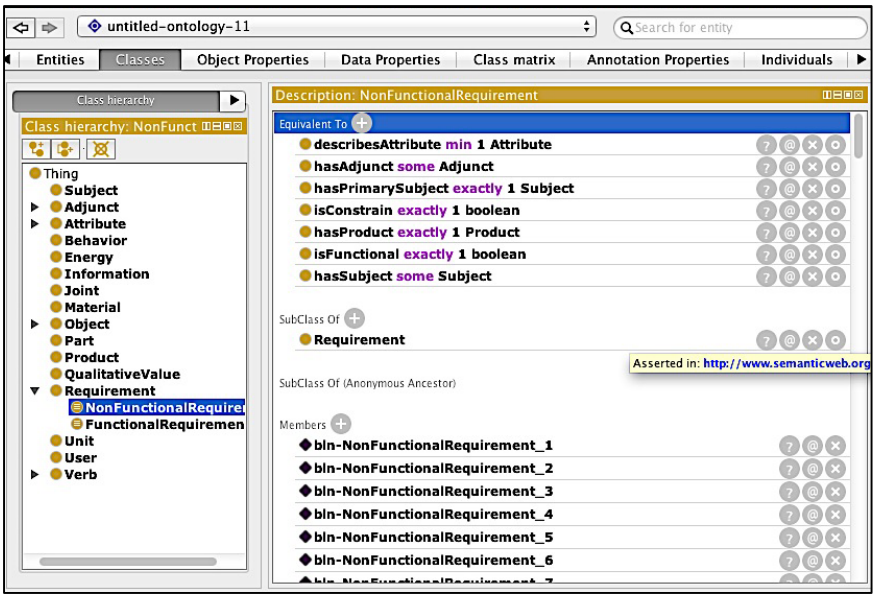

Figure 4: Screenshot of class editor for Nonfunctional Requirement in Protégé

$$Nonfunctional\ Requirement$$
$$= Requirement \cap (= 1\ hasPrimarySubject.Subject)$$
$$\cap\ (\exists hasSecondarySubject.Subject)$$
$$\cap (\geq 1\ describesAttribute.Attribute) \cap (\exists hasAdjunct.Adjunct)$$

(For Nonfunctional requirement hasPrimarySubject, hasSecondarySubject, and hasAdjunct coding are similar to the properties of Behavior. Thereby, to avoid redundancy only the property describesAttribute is shown here)

$$< owl{:}Class\ rdf{:}ID = "Attribute"/ >$$
$$< owl{:}ObjectProperty\ \ rdf{:}ID = "describesAttribute"/ >$$
$$< rdfs{:}Domain\ \ \ rdf{:}resource = "\#NonFunctionalRequirement\ ">$$
$$< rdfs{:}Range\ \ \ \ rdf{:}resource = "\#Attribute" >$$
$$</owl{:}ObjectProperty >$$
$$< owl{:}Class\ rdf{:}ID = "Attribute"/ >$$
$$< owl{:}Restriction\ >$$
$$< owl{:}onProperty\ rdf{:}resource = "\#describesAttribute"\ >$$
$$< owl{:}minCardinality\ \ rdf{:}datatype = \&\ xsd;\ nonNegativeInteger > 1$$
$$</owl{:}minCardinality\ >$$

```
        </owl:Restriction >
</owl:Class >
```

Definition (5): `Attribute` defines the characteristics of subject and they can be qualitative, quantitative or Boolean. `Qualitative Attribute` is a type of attribute that has exactly one qualitative value. `Quantity Attribute` is a type of attribute that has either a value or a lower or upper value or both. Further, numeric value has exactly one unit. `Boolean Attribute` is a type of attribute that has exactly one Boolean Value.

$$Quality\ Attribute = Attribute \cap (= 1\ hasQualitativeValue.QualitativeValue)$$

```
< owl:Class rdf:ID = "QualityAttribute"/ >
          < rdfs:subclassOf rdf:resource = "#Attribute ">
          </rdfs:subclassOf >
</owl: Class >
```

$$Quantity\ Attribute = Attribute \cap (\exists\ hasValue.(Value\ \cap = 1\ hasUnit.Unit)$$

```
< owl:Class rdf:ID = "QualitativeValue"/ >
< owl:ObjectProperty  rdf:ID = "hasQualitativeValue"/ >
          < rdfs:Domain   rdf:resource = "#QualityAttribute ">
          < rdfs:Range     rdf:resource = "#QualitativeValue">
</owl:ObjectProperty >
< owl:Class rdf:ID = "QuantityAttribute"/ >
          < rdfs:subclassOf rdf:resource = "#Attribute ">
          </rdfs:subclassOf >
</owl: Class >
< owl:DatatypeProperty  rdf:ID = "hasValue"/ >
          < rdfs:Domain   rdf:resource = "#QuantityAttribute">
          < rdfs:Range     rdf:resource = "&xsd;float ">
```

$</owl:DatatypeProperty>$

$Boolean\ Attribute = Attribute \cap (= 1\ hasBooleanValue.xsd:Boolean)$

$<owl:Class\ rdf:ID = "BooleanAttribute"/>$
$<rdfs:subclassOf\ rdf:resource = "\#Attribute\ ">$
$</rdfs:subclassOf>$
</owl: Class >
$<owl:DatatypeProperty\ rdf:ID = "hasBooleanValue"/>$
$<rdfs:Domain\ \ rdf:resource = "\#QuantityAttribute">$
$<rdfs:Range\ \ \ rdf:resource = "\&xsd;boolean\ ">$
$</owl:DatatypeProperty>$

2.3.3 Evaluate Ontology:

In order to be useful, the method must be objective, i.e., not influenced by the designer's preference or interpretation: a property reflected in its ability to produce ontologies that are similar between designers. Also the requirements can be written in various ways and structure of all requirements is not identical. Thereby, it is very important that the developed ontology can successfully accommodate a wide range of requirements while maintain its consistency. As a preliminary assessment of this reliability, requirements from different design projects were imported into the ontology. At present, this dynamic ReqOn consists of 247 functional requirements and 182 nonfunctional requirements for a total of 27 products. It also offers a standard vocabulary composed of 113 instances for verb (Sen et al., 2010a) and 126 instances for attribute. Apart from that it has several instances for all classes that are available to the designer to reuse. [Table 5] shows a few instances and their definition inherited from their type.

Table 5: ReqOn definition of different individual

Individual	Name	ReqOn Description
The electric kettle boils water quickly	Requirement	$= Statement$ $\cap\ (\exists hasProduct.electric\ Kettle)$ $\cap\ (\forall isFunctional.true)$ $\cap\ (\forall isConstrain.true)$
The electric kettle boils water quickly	Functional Requirement	$= Requirement$ $\cap \begin{pmatrix} = 1\ describesBehavior.boils\ water \\ quickly \end{pmatrix}$ $\cap\ (\forall isFunctional.true\ )$
Boils water quickly	Behavior	$= Component \cap \begin{pmatrix} = 1\ hasPrimarySubject. \\ electric\ kettle \end{pmatrix}$ $\cap\ (= 1\ hasPrimaryObject.water)$ $\cap\ (= 1\ hasVerb.boils)$ $\cap\ (\exists hasAdjunct.quickly)$
The electric kettle is light	Nonfunctional Requirement	$= Requirement$ $\cap\ (= 1\ hasPrimarySubject.electric\ kettle)$ $\cap\ (\geq 1\ describesAttribute.Weight)$
Weight (light)	Quality Attribute	$= Attribute$ $\cap\ (= 1\ hasQualitativeValue.light)$

2.4 Requirement Reuse:

Requirement reuse is one of the key assets for the designer to effectively design a novel product. Due to the potential benefits of requirement reuse, it gained noticeable recognition and attracted researchers. Currently, there are so many tools exist that facilitate requirement reuse as a functionality such as DOORS, JAMA, P&PM, ViReq etc (Rolland & Proix, 1992). However, all of these tools are specially designed for software requirements and they are not made to manage the requirements related to the engineering product design and development. [Table 6**]** shows a SPARQL query that returns the functional requirement, which contains nontransitive verb and also sorts the

result in an ascending order of requirement and for each requirement in ascending order of the verb.

Table 6: SPARQL query that returns the entire set of functional requirement, which contains Nontransitive verb, with their corresponding behavior and verb

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ReqOn: <http://www.semanticweb.org/fa11/ontologies/2014/1/untitled-ontology-11#>

SELECT  ?FunctionalRequirement ?Behavior ?Verb
        WHERE { ?FunctionalRequirement ReqOn:describesBehavior ?Behavior.
                ?Behavior ReqOn:hasVerb ?Verb .
                ?Verb rdf:type ReqOn:NonTransitiveVerb  }
ORDER BY  ?FunctionalRequirement (?Verb)
```

In this circumstance, developing an effective search engine, which is specially designed for requirements, will be beneficial. At the semantic level, ReqOn can retrieve its data using RDF query languages such as SPARQL and thereby increase its effectiveness (Motik, Sattler, & Studer, 2005).

2.5 Implementation:

A java and OWL API tool has been developed based on the structure of the ontology demonstrated in the previous sections. It translates the text into ontology using OWL API. The tool will provide two different interactive windows for functional and

nonfunctional requirement based on the user selection. [Figure 5] and [Figure 6] show the functional and nonfunctional requirement editors respectively.

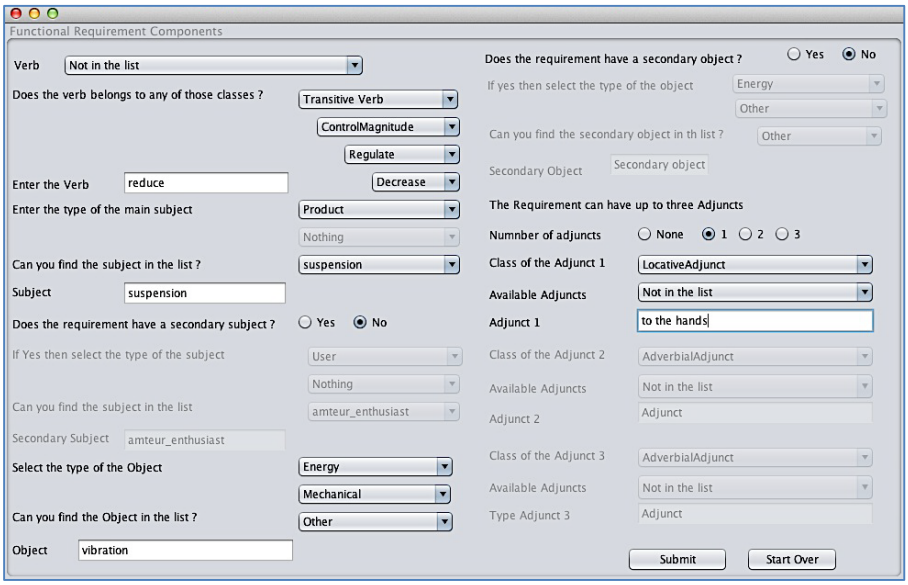

Figure 5: Screenshot of the requirement editor for functional requirement

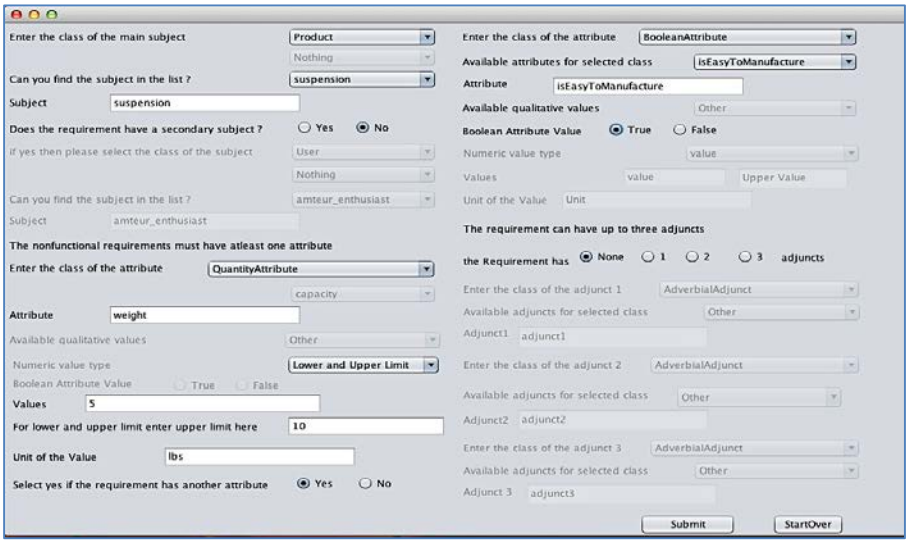

Figure 6: Screenshot of the requirement editor for functional requirement

The tool has a default ontology saved into it and users also have the flexibility to use any other owl file instead of the default ontology. The user has to enter the requirement statement and other components of the requirement as text. To use the tool the user does not require any knowledge of ontology. The user has to identify the grammatical components of the sentence such as subject, object, verb, and adjunct etc. [Figure 7**]** shows the run time environment for the tool.

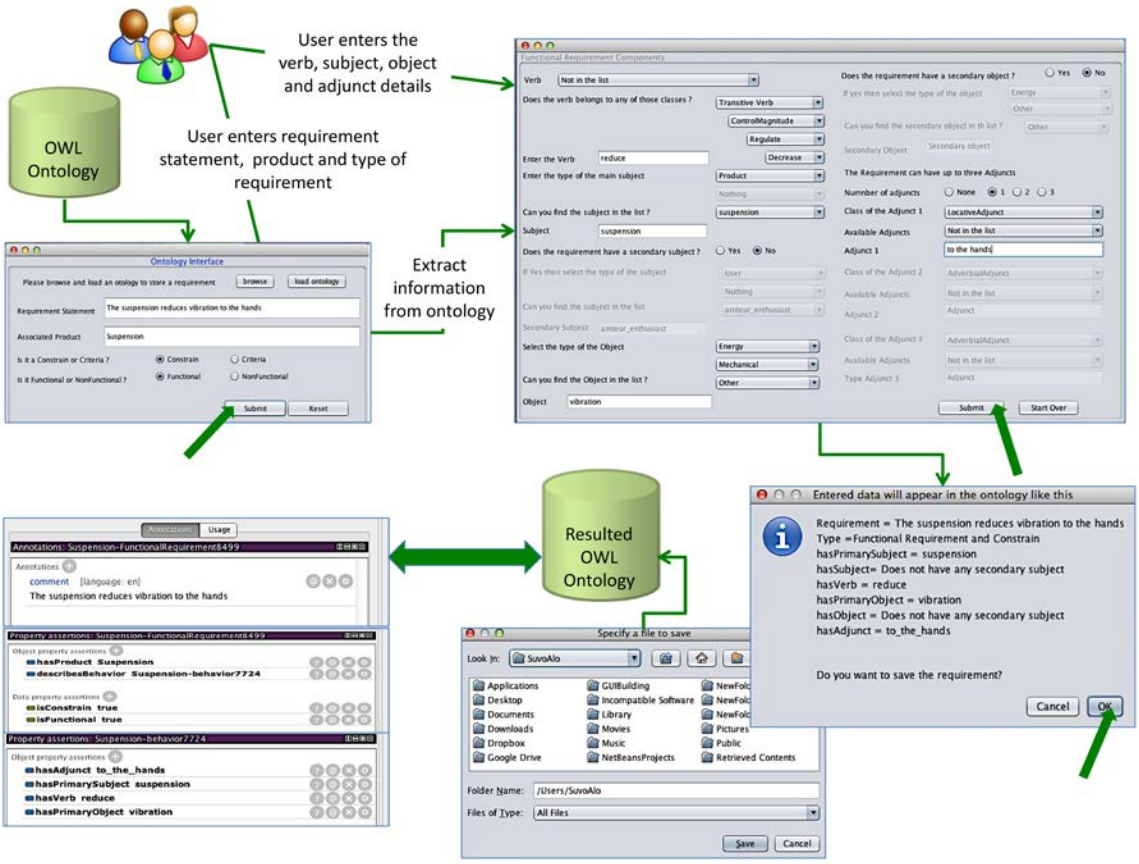

Figure 7:Run time environment of the java and OWL API based tool to convert textual requirement into owl ontology

By user defined components the tool will first try to utilize the exiting instances of the ontology in order to avoid duplication of instances and if no similar instances exists in the ontology then the tool will create a new instance. For each new instance the tool will also assert the appropriate class of the instance using OWLClassAssertionAxiom. After converting the text into owl instances the tool will assert the value for each property of the requirement using OWLObjectPropertyAssertionAxiom and OWLDataPropertyAssertionAxiom. After converting text into ontology the tool can save changes either on the uploaded ontology or to a different file by using user defined file directory. [Figure 8] shows a nonfunctional requirement "The suspension weighs between 5-10 lbs" in protégé 4.3.This requirement is translated into the ontology using the developed tool.

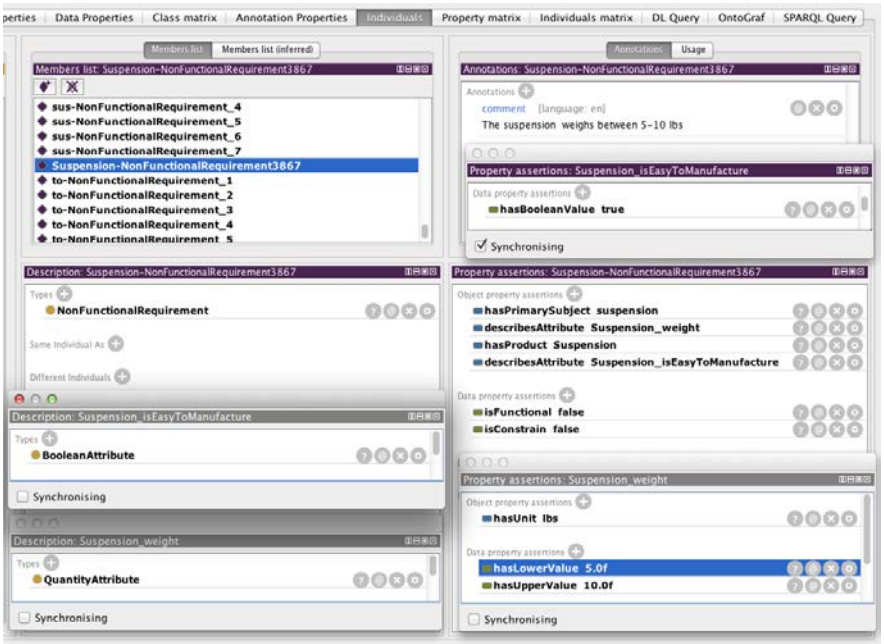

Figure 8: Screenshot of a nonfunctional requirement converted using the tool

2.6 Conclusion:

The objective of this chapter was to propose a systematic approach to develop ReqOn, as ontology for requirements generated at the early stage of the design. This chapter also provides a protocol for writing and modifying design requirements. ReqOn is limited to the requirement statements but it can be extended for technical specifications and guidelines or other relevant text using similar concept. The proposed approach suggests a four-step method to conceptualize, develop, validate and document the ontology. Further, the ontology developing method portrayed three tasks for capturing, coding and integrating the ontology. OWL DL is used as the language to code ReqOn and the ontology was populated with several requirements to ensure the validity of the ontology concept. Requirements used in this study were collected from different design projects and some of them were also created following the proposed protocol to check the usefulness of the protocol. It was also observed that the requirements created following the protocol was nearly similar to the requirements collected from the design projects. Additionally, it is possible to import them into the ReqOn successfully without losing any element. ReqOn provides a common vocabulary that can be reused while creating requirements and reuse functionality can be achieved by using SPARQL query. Use of SPARQL query enhances the flexibility of retrieving data and thereby, reusing the requirements and their components.

CHAPTER 3

INFORMATION CONTENT MEASUREMENT

3.1 Introduction:

In this chapter, the details of the proposed methods for information content measurement are described. As discussed earlier, information content can be used as measure of specificity or in formativeness of engineering documents and artifacts. The overall goal of the proposed methods is to provide a formal and replicable means for measuring the information content of engineering requirements. Design is an information-transforming process. It starts with a need or a problem statement (information) and ends with specification of a solution (information). In novel product design, especially within the early stages, the steps between these two terminals are not preordained, they are iterative and complex, and they usually vary between design problems. Consequently, the information transforms are not easily visible. In order to study these transforms, first, the information content of the artifacts used in early design 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. This need sets the overall motivation of this research.

Within each artifact, information can be present in various forms, such as text, sketch, or graphs (Suh, 1990) . *Form-neutrality* of information is one of the key assumptions of this work. Form-neutrality indicates that the amount of information contained in a design artifact does not depend on its form; being text, sketch, or graph. Therefore, if all design artifacts can be translated into a standard and formal

representation, such as an OWL ontology, no information loss should occur during this transformation. The ontology that is introduced in this paper is focused on requirement representation. Different OWL ontologies can be created for different type of design artifacts.

It is, therefore, the job of the information metric to ascertain this form-neutrality. To this end, we propose a two-step process of measuring information [Figure 9]. First, a form-neutral representation that could be used to translate any type of design information into a neutral form is used. Next, Shannon's metrics of information content are applied to this neutral form.

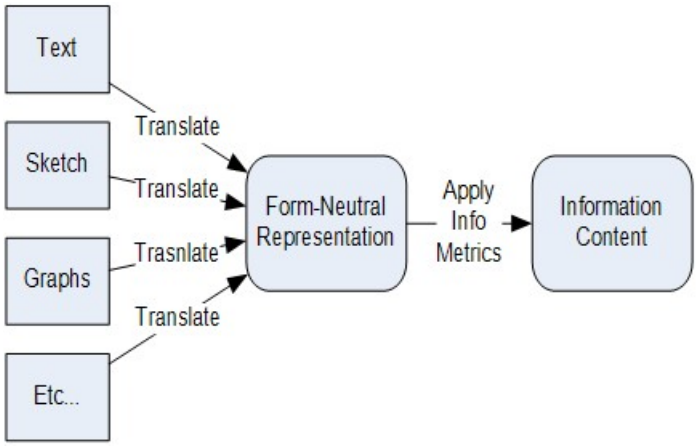

Figure 9: Two-step approach of information measurement

3.2 Scope:

In this section, the proposed approach is demonstrated using text-based input information and the ReqOn Ontology REQUIREMENT ONTOLOGY as the form-neutral representation.. The products that are included in this study are typical consumer products with low-to-medium complexity.

3.3 Related Work:

The notion of information content in the context of engineering design was first introduced by Suh (Suh, 2001) in his seminal work on Axiomatic Design. Based on the Information Axiom, the second axiom in the Axiomatic Design theory, the best design is the one with minimum information content. In his earlier work on using the axiomatic approach to improve the productivity of manufacturing systems in 1978, Suh (Suh, 2001) loosely defined the information content of a product as the instructions necessary for fully describing the product and its associated manufacturing and assembly processes. Based on the information axiom, minimizing information means loosening tolerances, simplifying shapes, accepting rough surfaces, and reducing the number of instructions required for processing and inserting the parts. However, he didn't provide any rigorous formalism for quantifying the information content in different design documents.

In 1980, Wilson (D. R. Wilson & Massachusetts Institute of Technology. Department of Mechanical Engineering, 1980) proposed a metric for indirect measurement of information content as the logarithm of the inverse of the probability of satisfying a tolerance. This definition, although limited to the geometric features of design artifacts, was later embraced by the Axiomatic Design theory and it was further generalized and

redefined as the inverse of the logarithm of the probability that the design successfully meets its functional requirements (Suh, 2001). Higher probability of success means less information content, which, in turn, implies less complex products. This measure of information provided a basis for other researchers to further investigate and enhance the underlying mathematical and stochastic models of the information axiom (El-Haik & Yang, 1999; Frey, Jahangir, & Engelhardt, 2000). Associating product information content to its complexities proved to be a reasonable approach for information quantification and uncertainty analysis in engineering design and researchers used the information axiom in different applications such as predicting manufacturing time and cost (Collopy & Eames, 2001). However, due to the generic nature of the proposed measure, applying it to different representations of design artifacts especially in early design is not always trivial. For example, one may ask how the "probability of success" should be interpreted when measuring the information content of a requirements list or an early sketch of the product. To this end, there is a need for more direct and absolute metrics for measuring the information content of the intermediate design artifacts created at various stages of design process and represented using different formalisms.

In function-based design, metrics of information content of function structure graphs have been proposed (Sen, Summers, & Mocko, 2010; Sen, Caldwell, Summers, & Mocko, 2010b). These metrics build on Shannon's notion of information entropy (Shannon, 2001) and can separately account for information from the functions and flows (El-Haik & Yang, 1999) and that from the topological connections of these graphs (Collopy & Eames, 2001) based on a fixed vocabulary of functions and flows.

Intuitively, information content represents the number of questions that are answered by a function model for a designer who receives the model as a message containing discrete elements such as functions and flows (El-Haik & Yang, 1999).

3.4 Information Content Measurement:

Within each artifact, information can be present 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 it form such as text or sketch, and that this amount could be measured. The described ontology 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 than can be contemplated as containers of information.

The proposed information content measurement technique in this work is based on the entropy metric used in Shannon's Information Theory (Shannon, 2001). In information theory, entropy is the measure of uncertainty in a model. On the contrary, uncertainty reduction is the amount of information required to diminish the uncertainty. 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, 2001). According to this theory, the entropy of a discrete random variable X with a probability distribution p(x) is defined by:

$$H_x \equiv -\sum_{x \in X} p(X)\ \log_2 p(x) = E\ \log_2 \frac{1}{p(X)} \qquad \text{Equation 1}$$

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 word drawn from a finite vocabulary. For a requirement statement written in plain English, the reference vocabulary would be the entire dictionary of the English language. The requirement ontology, as a subset of the English language is used as the reference vocabulary in this wok. Requirement statements are considered as a message, each instances (i) as a discrete symbol and classes (c) are the set of instances in which instance i is comprised. Further, information is contained in each node and in the topological connections. In this section, two different approaches are taken to measure the entropy of different entities of the OWL ontology – uniform random distribution approach and decision-making approach. Each approach is explained is details with relevant calculations.

3.4.1 Uniform Random Distribution Approach:

The main assumption of this approach is the uniform probability distribution of the entities that means all the leaf classes of the ontology have equal chance of occurring. ReqOn has different owl classes and subclasses. Each class in the ontology represents a variable with some inherent entropy. In this approach, the entropy of each class is

considered to be depended on the structure of the class and the number of instances of the class that are available in the ontology at any given timestamp. 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 sub-class structure introduce more uncertainty. Also the classes that are instantiated more frequently have higher entropy because the probability of encountering of a particular instance would be low. The taxonomy entropy of a class is based on the probability that a particular class is selected when formulating a requirement and the size entropy is based on probability that the selected class assumes a certain value. The total entropy of a class (Ec_i) is calculated as the summation of taxonomy (TEc_i) and size (SEc_i) entropy [Equation 2].

$$Ec_i = TEc_i + SEc_i \qquad \text{Equation 2}$$

3.4.1.1 Taxonomy Entropy:

The taxonomy entropy of a class depends on the structure of the class. Classes with deeper subclass structures tend to have higher taxonomy entropy. The following steps are used to calculate the taxonomy entropy of a class. The Verb class is used as an example here. The hierarchical structure of Verb is shown in **[**Figure 10].

- Count the number of leaf classes under each parent class (N). A leaf class is one that doesn't have any subclasses. For example `NonTransitiveVerb`, `Divide`, and `Import` are examples of leaf classes. In [Figure 10] 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).
- Next, 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 total number of leaf nodes under its parent class (n/N). For example, Probability of occurrence of class `Branch` = 4/36.
- After determining the probability of occurrence of the class, the entropy measure is applied for calculating the taxonomy entropy of the class.

$$TEc_i = -log_2(Pc_i) \qquad \text{Equation 3}$$

Where, TEc_i is the Taxonomy Entropy of the i^{th} class c_i and Pc_i is the probability of occurrence of class c_i. For example, the Taxonomy Entropy of class Branch (TE_{branch}) is calculated as follows:

$$TE_{Branch} = -log_2 P_{Branch} = -log_2 \frac{4}{36} \approx 2.97\ bis \qquad \text{Equation 4}$$

3.4.1.2 Size Entropy:

Entropy measures can be applied to measuring the size entropy of each class as well. Size of a class implies the number of individuals, or instances, under the class at any instant in time. Size does not consider the individuals that are under direct or indirect subclasses of a class. To measure the size entropy, the first step is to calculate the probability of occurrence of each individual under the class. The size entropy of class c_i is calculated using [Equation 5].

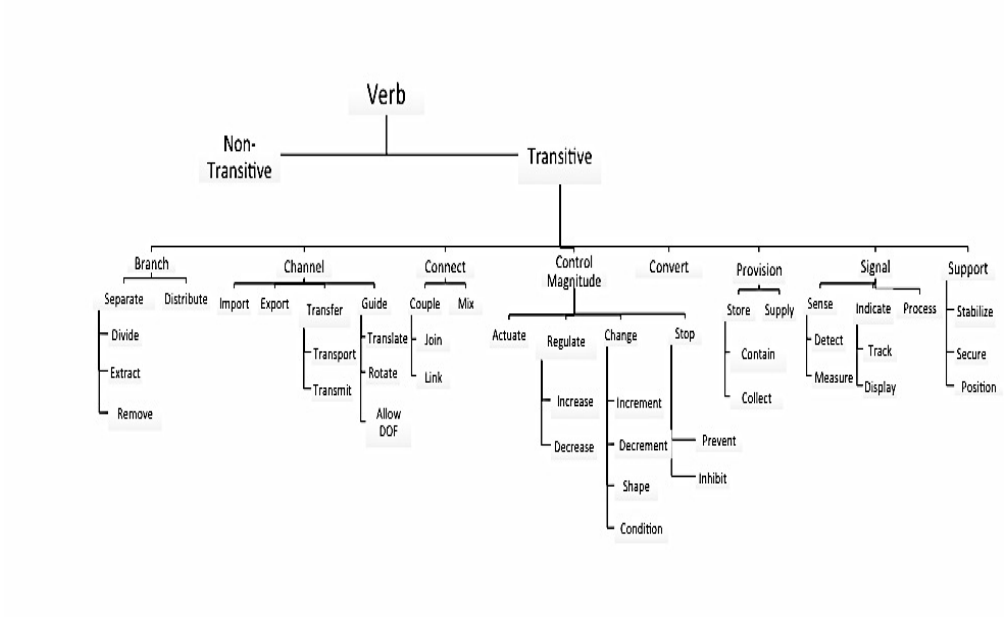

Figure 10: The Hierarchical Structure of Verb Class

$$SEc_i = -log_2(\frac{1}{Nc_i}) \qquad \text{Equation 5}$$

For example, if there are nine instances of the class Product in the ontology, then:

$$SEc_{Product} = -log_2 \frac{1}{9} = 2.197\ bits$$

Equation 6

Size entropy and Taxonomy entropy of all classes such as Verb, Product, or Material can be measured using the above procedures. But for the 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.*

$$E_{Behavior} = E_{Subject} + E_{Primary\ Subject} + E_{Object} + E_{Primary\ Object} + E_{Verb} + E_{Adjunct}$$

Equation 7

3.4.2 Sequential Selection Approach:

In this approach, it is assumed that a design decision is made through making a selection from a finite set of options. For example, when selecting the material for a product, the designer chooses a material from the available set of materials such a aluminum, ABS plastic, or wood. In most occasions, the designer needs to traverse a taxonomy (such as material taxonomy of function taxonomy) in order to make the final decision. A selection is made at each level of the taxonomy and once the required depth is gained, the decision is finalized. Decision making approach is a more direct way of measuring the information content. In this approach, Shannon's theory (Shannon, 2001) is applied as well to measure the information content but the concept of measuring the probability is different from the previous approach. In the sequential selection approach,

the classed of the ontology do not follow a uniform probability distribution. Instead, the probability of occurrence of each class in the ontology depends on the number of direct and indirect instances of that class. Conventionally, hierarchical structures are represented using a tree diagram and for each element of a requirement statement, the designer has to make a series of decisions to get to the required depth. For reaching the classes that are positioned at lower levels, more decisions need to be made. In this approach, it is also assumed that the probability of occurrence of each parent class located at the top of the tree network (for owl ontology the classes those are directly under OWL:Thing), will be 1. The basis for this assumption was that selecting the top class would not be considered as a decision; since there is only one unique node present. For example, selection of the "verb" class by itself will not be considered as a decision while a functional requirement is being composed. Hence, there is 100% chance that the designer will select some type of "Verb" class in the requirement statement. It is evident that the probability increases as the designer moves up the hierarchy and thereby its information content decreases. For most of the classes in the developed ontology, hierarchical structures with their relations were explicitly defined. However, some atomic classes such as *Product, Part, Qualitative Value, User, and Unit* also exist in the ontology that have no sub-class structure. Probabilities of occurrence of these atomic classes are 1 and no further calculation is required for them. However, for rest of the classes, first a top down approach is applied to create a standard metric to measure the probability of occurrence of each class at each level of the tree. It can also be argued that the probability of encountering an instance of child class depends on the

chance of selecting its parent. To realize this concept a bottom up method is adopted to trace their super classes and the sequence of decisions acquired to reach the super classes. After that the entropy involved in each decision was measured. More precisely, the bottom up method includes determining the direct super class in the hierarchy, adding up their information content, and the process repeats until the top level of the hierarchy is reached. Further, the tree structure of the `Verb` class is used to demonstrate the method in details.

3.4.2.1.1 Class Entropy (Top down approach):

It involves moving through the tree from top level to the bottom and calculating probability of occurrence of all classes at each level separately. In order to determine the probability of occurrence, the following tasks have to be performed.

1. The first task is to count the number of direct and indirect instances (n) of each class (c). This task can be performed programmatically by using Algorithm 1. For example, instance count of class `Verb` will be the summation of instances those are directly under `Verb` (0) and summation of instances of its subclasses (113). Therefore, the total instance count for class `Verb` will be 113. Similarly, n for Transitive Verb will be 95, Non Transitive Verb will be18, and for Branch will be10 etc.

2. For parent classes, which are at the top level of the hierarchy, have the probability of occurrence one. For example, *Verb* is at the top level of the hierarchical structure for *Verb* class. Hence, the probability of class *Verb* will be one.
3. Determine the equivalent classes of each class at each level except the top level. Algorithm 2 can be used to determine the equivalent classes in OWL API. For example, at level 2 equivalent classes of class Transitive Verb will be *Nontransitive Verb*. Similarly, at level 3 the sibling classes *Branch, Channel, Connect, Control Magnitude, Convert, Provision, Signal, Support,* and *Other Verb* are all equivalent to each other.
4. Calculate the summation of instance (N) of equivalent classes including the class of interest at each level. Algorithm 3 can be used to calculate (N) for each class in OWL API. For example, number of instances of all equivalent class at level 2 will be 113, since, the number of instances of Transitive Verb and NonTransitive Verb are 95 and 18, respectively. Similarly, at level 3, N for *Branch, Channel, Connect, Control Magnitude, Convert, Provision, Signal, Support,* and *Other Verb* will be 95.
5. Compute the probability of occurrence of each class using Equation 17. For example, probability of occurrence of Transitive Verb is equal to (95/113 =. 841), for Non Transitive Verb (18/113 =. 159), and for Branch (10/95 = .105) etc.
6. After determining the probability of occurrence of the class, the entropy measure is applied [Equation 1] for calculating the information contained in each class. For example, class entropy of *Verb* will be 0, for *Transitive Verb* (-log2 .841 =

.2498 bits), for `NonTransitive Verb` (-log2 .159 = 2.653 bits), and for `Branch` (-log2 .105 = 3.25 bits) etc. Similarly, class entropy of each class can be calculated using [Equation 9].

Algorithm 1:

$double\ count\ = 0; count\ \leftarrow Number\ of\ instances\ of\ class\ c$

$OWLClassExpression : c$

$Set < OWLNamedIndividual > \boldsymbol{instance}$

$= reasoner.getInstances(c, false).getFlattened();$

$\boldsymbol{for}\ each\ element\ \boldsymbol{i}\ of\ the\ set < OWLNamedIndividual > \boldsymbol{instance}\ check$

$\boldsymbol{if}\ (!\, i.isBottomEntity())\ \{$

$count = count + 1\ \};$

$\boldsymbol{return}\ count;$

Algorithm 2:

$OWLClassExpression : c$

$Set < OWLClass > \boldsymbol{parent} = reasoner.getEquivalentClasses(c).getEntities();$

$\boldsymbol{return}\ parent\ ;$

Algorithm 3:

$OWLClass\ i\ \leftarrow a\ , b\ , c\ , d \ldots\ldots\ldots n$

$double\ x = 0;$

$double\ count_i\ = 0;\ count_i\ \leftarrow Number\ of\ instances\ of\ class\ i$

Apply Algorithm 1 to $Set < OWLClass > \boldsymbol{parent}$ to get the instance count of each equivalent class i including the class of interest.

$$x = \sum_{i=a}^{n} count_a + count_b + count_c + \cdots \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + count_n;$$

$$P(Class)_c = \frac{n}{N} \qquad \text{Equation 8}$$

$n = Number\ of\ instances\ of\ class\ c$

$N = Number\ of\ instances\ of\ equivalent\ classes\ of\ class\ c$

$$IC(Class_c) = -log_2 P_c \qquad \text{Equation 9}$$

3.4.2.1.2 Decision Entropy (Bottom up approach):

This part of the method includes tracking back the super class and measuring the information involved in the decisions of selecting these super classes. To implement this concept, it is necessary to find out the direct super class of each child class and measure the entropy of the super class. This process continues until it reaches to the top parent, where the information content is 0. The necessary tasks for this method are –

1. Consider the tree from bottom level towards the top level and determine the super class of each element at the bottom most level. For Example, [Figure 10] has 5 levels including the top level Verb. In this hierarchical structure the fifth level, which is the bottom most level contain classes *Divide, Extract, Remove, Transport, Transmit, Translate, Rotate, Allow DOF, Join, Link, Increase, Decrease, Increment, Decrement, Shape, Condition, prevent, inhibit, contain, collect, detect, measure, track, display, stabilize, secure*, and *position*. This step involves finding the direct super class of all the bottom most classes such

as super class of *Divide, Extract* and *Remove* will be *Separate*, super class of *Transmit* and *Transport* will be *Guide* etc.

2. Repeat the process until the super class is the top level class, which is directly under OWL:Thing. For example, super class of *Separate* is *Branch*, super class of *Branch* is *Transitive Verb,* and again super class of *Transitive Verb* is verb. Since, the super class of *Verb* is *OWL:Thing*, this step will terminate when it will encounter *OWL:Thing,* as the super class of *Verb*.
3. The next step is to measure the entropy lies under the decision of selecting each super class. For example, if designer pick an instance of class *Remove* as the verb for creating any functional requirement, the selection involves a sequence of decisions. [Equation 14**]** shows the chain of decisions had been made prior choosing the instance under class *Remove*. Further, it can be argued that the entropy of each decision will be equal to the information contained in each class selected by that decision [Equation 10]. In the **[**Equation 15**]** entropy of decision D_{Remove} is the quantity of information contained in the class *Transitive Verb, Branch and Separate*.

$$Decision\ (D) = \sum_{i=1}^{n} S_1 + S_2 + S_3 + \cdots \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + S_n \qquad \text{Equation 10}$$

$$IC(D) = \sum_{i=1}^{n} IC(S_1) + IC(S_2) + IC(S_3) + \cdots \ldots\ldots\ldots + IC(S_n) \qquad \text{Equation 11}$$

4. Finally, the entropy for decision factors can be obtained by using [Equation 11].

3.4.2.1.3 Total Entropy:

Total entropy of each element of the ontology will be the sum of the information contained in the class of the element and the information content of the decision associated in the selection of the element.

$$IC(Element) = IC(Class_c) + IC(D) \quad \text{Equation 12}$$

For example, total information content of an instance of class is shown in [Equation 13, Equation 14, and Equation 15].

$$clear \leftarrow Individual\ , Types: Remove \quad \text{Equation 13}$$

$$IC(Clear) = IC(Remove) + IC(D_{Remove})$$

$$Decision(D_{Remove}) = Transitive\ Verb \rightarrow Branch \rightarrow Separate \rightarrow Remove \quad \text{Equation 14}$$

$$IC(D_{Remove}) = \sum_{i=1}^{3} IC(Transitive\ Verb) + IC(Branch) + IC(Separate) \quad \text{Equation 15}$$

$$IC(Transitive\ Verb) = .25\ , \quad IC(Branch) = 3.24\ , \quad IC(Separate) = .13\ ,$$

$$IC(Remove) = 1.32$$

$$IC(D_{Remove}) = .25 + 3.24 + .13 = 3.62\ Bits$$

$$IC(Clear) = = 3.62 + 1.32 = 4.94\ Bits$$

3.5 Entropy of a Requirement Statement:

The information content of the functional requirement and nonfunctional requirement statement can be calculated following the below steps.

- Step 1: Identify if the requirement statement is functional or nonfunctional.
- Step 2: Measure size (**SEc_i**) and taxonomy entropy (**TEc_i**) of the atomic classes embedded in the requirement.
- Step 3: Add up the calculated entropies to calculate the aggregate entropy of the class.

[Equation 17 and Equation 18] are the equations for measuring entropy for functional and nonfunctional requirement statement, respectively. Table 7 shows an example of taxonomy and size entropy measurement of Functional and Nonfunctional requirement statement. Figure 11 shows the decision flowchart for measuring entropy of a requirement statement.

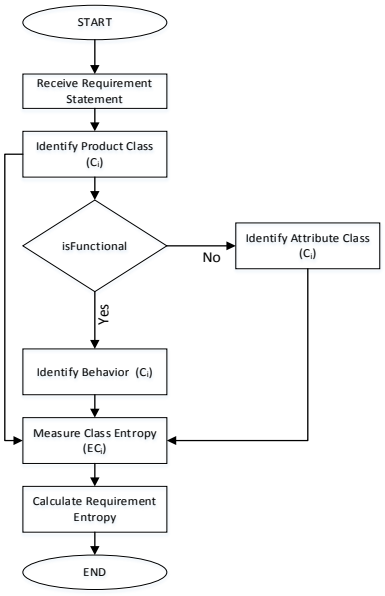

Figure 11: Flowchart for measuring IC of a Requirement Statement

$E_{Behavior} = E_{Subject} + E_{Primary\ Subject} + E_{Object} + E_{Primary\ Object} + E_{Verb} + E_{Adjunct}$	Equation 16
$E\_(Functional\ Requirement) = E\_product + E\_(Behavior) + 1(is\ Functional) + 1(isConstraint)$	Equation 17
$E_{Nonfunctional\ Requirement} = E_{Subject} + E_{Primary\ Subject} + E_{Attribute} + E_{Adjunct} + 1(is\ Functional) + 1(isConstraint)$	Equation 18

Table 7: Example of taxonomy and size entropy measurement of Functional and Nonfunctional statement

Functional Requirement Outside surface of the electric kettle remains cool enough to touch even when the water inside is boiling						
Object Property	**Data Property**	**Property Value**	**Class**	**TEc$_i$**	**SEc$_i$**	**Ec$_i$**
hasPrimarySubject		Outside Surface	Part	0	5.86	5.86
hasSubject		Electric Kettle	Product	0	4.58	4.58
hasProduct		Electric Kettle	Product	0	4.58	4.58
hasVerb		Remains	Non Transitive Verb	5.32	4.17	9.49
hasAdjunct		Cool enough to touch	Adverbial Adjunct	3.17	6.25	9.42
hasAdjunct		Even when the water is boiling	Conditional Adjunct	3.17	5.21	8.38
	isFunctional	true	N/A			1
	isConstraint	true	N/A			1
Total				**11.66**	**30.65**	**44.31**

Table 7- Continued: Example of taxonomy and size entropy measurement of Functional and Nonfunctional statement

Object Property	Data Property	Property Value	Class	TEc_i	SEc_i	Ec_i
NonFunctional Requirement Coffee maker has easy to view water window for easy filling of water						
hasPrimarySubject		Water Window	Part	0	5.86	5.86
hasSubject		Coffee Maker	Product	0	4.58	4.58
hasProduct		Coffee Maker	Product	0	4.58	4.58
describesAttribute		isEasytoView	Boolean Attribute	1.58	6.07	7.65
	hasBoolean Value	true	N/A			1
hasAdjunct		For easy filling of water	Reason Adjunct	3.17	4.09	7.26
	isFunctional	false				1
	isConstraint	false				1
Total				**4.75**	**25.18**	**32.93**

3.5.1 Comparison of Uniform Random Distribution Approach and Decision Making Approach:

To compare the results of the proposed methods, it is first necessary to normalize the calculated values based on each method. Normalization is done through dividing the information content calculated for a requirement statement by the maximum possible information content based on the current state of the ontology. **[**Figure 12] shows how the proposed methods correlate. The calculated values are based on the requirement set for the bike suspension. Functional and nonfunctional requirements for bicycle suspension were imported into the ontology and information content of each requirement were

measured using both methods separately and then the results were normalized to an equivalent scale of 0-1.

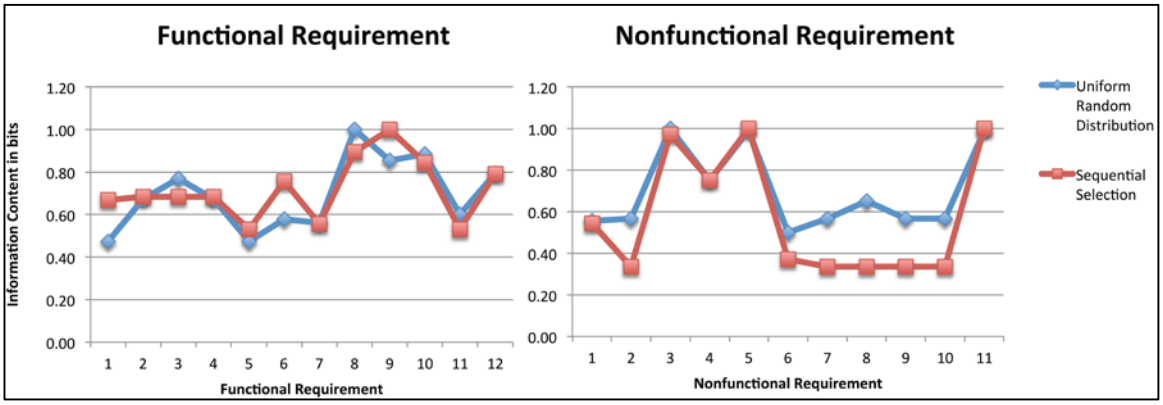

Figure 12: Comparison of Uniform Random Distribution Approach and Sequential Selection Approach for functional and nonfunctional requirements of suspension

$$Normalized\ IC(Element) = \frac{IC(Element)}{Max\ IC} \qquad \text{Equation 19}$$

From Figure 12] it can be contented that both techniques are valid. In the first method, class probability is based on the structure of the class taxonomy whereas in the second method, class probability is based on the number of instances of the class. In Figure 12] the information content lies within 0.4 to 1 bits for functional requirements and 0.5 to 1 bits for nonfunctional requirements in uniform random distribution approach, whereas, it varies between 0.5 to 1 bits for functional requirements and .34 to 1 bits for nonfunctional requirements in sequential selection approach. Information content measured using these two different approaches is almost similar in values. Therefor it can be asserted that the two approaches provide almost similar results for information content

measure with a little deviation. In summary both methods can be used to measure the information content of a requirement statement.

3.6 Implementation:

A Java-based tool has been developed based on the algorithm detailed in the previous section that can semi-automatically measure the information content of the requirements represented ontologically. The developed tool uses OWL API for interacting with the ontology. It is semi-automated in a sense that the user needs to translate the engineering requirements written in natural language into an ontological representation following the protocol described in REQUIREMENT ONTOLOGY. Protégé is used for creation of requirement instances in the ontology. The tool receives OWL/XML file created in Protégé as the input and measures the entropy of the selected entities. The developed tool enables the user to calculate the entropy of a particular class [Figure 13], a particular requirement [Figure 15], and the overall entropy of a selected product [Figure 14].

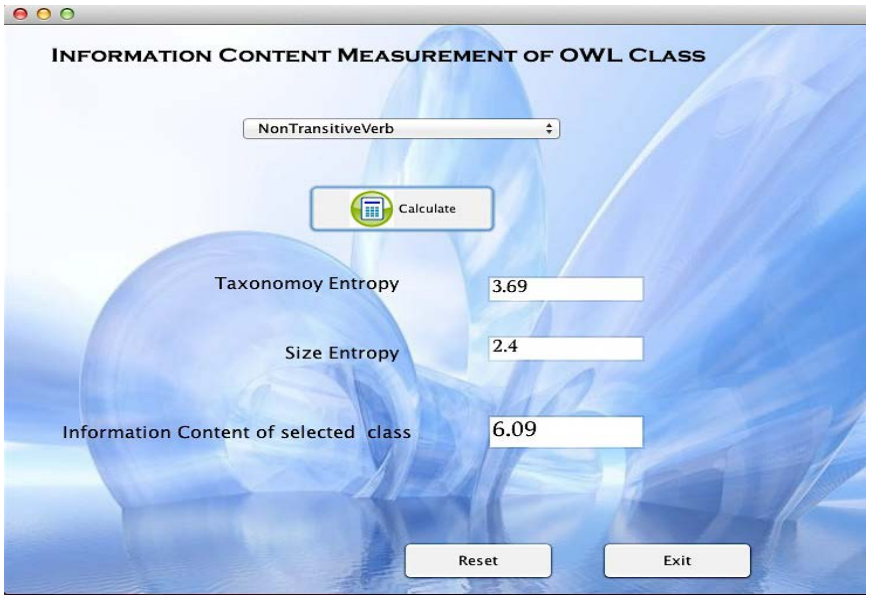

Figure 13: The interface for class entropy measurement

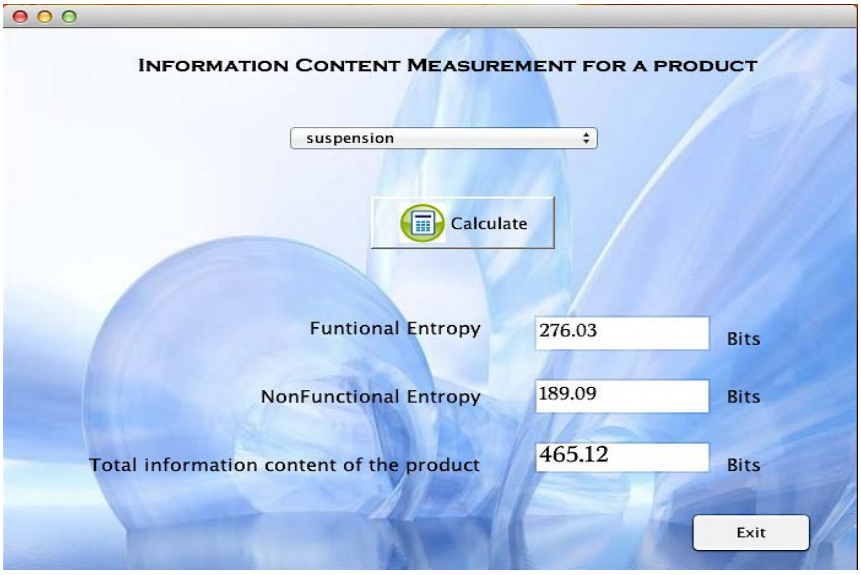

Figure 14: The interface for product entropy measurement

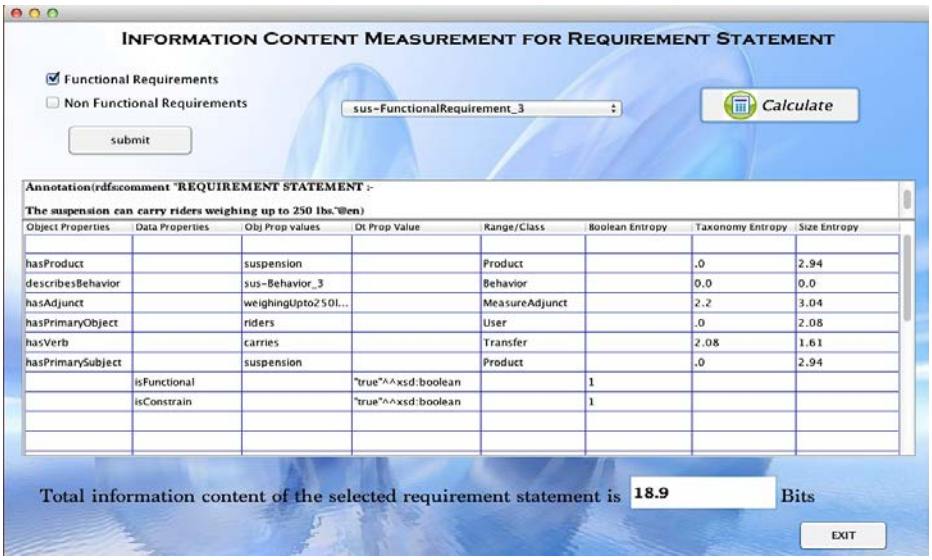

Figure 15: The interface for requirement entropy measurement

3.7 Experiments:

To validate the proposed ontology and the information metrics, a series of experiments were conducted.

3.7.1 Experiment 1: Information Content of Requirements for Different Product Families

In this experiment, two families of consumer products, namely, small kitchen appliances and small gardening equipment, were studied with respect to the information content of their functional and non-functional requirements. The objective of this experimentation was to investigate if information content is product-dependent quantity. In other words, the question is if the type and nature of the functions of product have any impact on the amount of information contained in their engineering requirements. To this end, 5-7

products were arbitrarily picked under each category. The selected products were fairly similar in terms of the level of complexity, number of parts, and type of user interaction. For each product, about 17-20 requirements were generated based on the structure suggested by the ontology. All requirement statements were imported to the ontology and their information content were measured via the developed tool. [Table 8] shows the summary of the results.

Table 8: Average Information content for functional and nonfunctional requirements of small kitchen appliances and small garden equipment of equal complexity level

Product Name	**No. of Req.**	**No. of Func. req.**	**Functional IC**	**No. of Non Func. Req.**	**Non Functional IC**	**Total IC**	**IC per Requirement**	**Avg. Functio-nal IC**	**Average Non Functional IC**
Kitchen Appliances									
Coffee maker	20	8	303.71	12	315.9	619.61	30.98	37.96	26.32
Electric Kettle	20	8	330.27	12	296.92	627.19	31.36	41.28	24.74
Blender	20	13	566.82	7	185.22	752.04	37.60	43.60	26.46
Toaster Oven	20	12	453.91	8	186.31	640.22	32.01	37.82	23.28
Rice Cooker	20	14	537.61	6	130.64	668.25	33.41	38.40	21.77
Ice Cream Maker	19	8	346.27	11	285.03	631.3	33.22	43.28	25.91
Electric Grill	18	13	561.98	5	121.59	683.57	37.97	43.22	24.31
Mean						**660.31**	**33.79**	**40.79**	**24.68**

Table 8 - Continued: Average Information content for functional and nonfunctional requirements of small kitchen appliances and small garden equipment of equal complexity level

Product Name	No. of Req.	No. of Func. req.	Functional IC	No. of Non Func. Req.	Non Functional IC	Total IC	IC per Requirement	Avg. Functio-nal IC	Average Non Functional IC
Gardening Equipment									
Lawn Mower	19	12	470.95	7	155.25	626.2	32.95	39.246	22.17
Electric Leaf Blower	17	12	409.73	5	111.69	521.42	30.67	34.144	22.33
Electric Snow Blower	19	15	634.65	4	79.28	713.93	37.57	42.310	19.82
Electric Pressure Washer	18	13	466.49	5	115.37	581.86	32.32	35.884	23.07
Electric Cultivator	19	16	648.77	3	67.11	715.88	37.67	40.548	22.37
Mean						**631.85**	**34.24**	**38.42**	**21.95**

Table 9: Two sample T- test for kitchen appliances vs. garden equipment for functional and nonfunctional requirement

Product Family	N	Mean	St. Dev.	SE Mean	Pooled St. Dev.	T Value	P Value
Functional							
Kitchen Appliances	7	40.80	2.67	1.0	2.9641	1.37	.202
Garden Equipment	5	38.43	3.36	1.5			
Difference = μ (Kitchen Appliances) - μ (Garden Equipment) Estimate for difference: 2.37 95% CI for difference: (-1.50, 6.24) T-Test of difference = 0 (vs ≠): T-Value = 1.37 P-Value = 0.202 , DF = 10							

Table 9 - Continued: Two sample T- test for kitchen appliances vs. garden equipment for functional and nonfunctional requirement

Nonfunctional							
Kitchen Appliances	7	24.69	1.73	.65	1.5521	3.01	.013
Garden Equipment	5	21.96	1.24	.56			
Difference = μ (Kitchen Appliances) - μ (Garden Equipment) Estimate for difference: 2.732 95% CI for difference: (0.708, 4.757) T-Test of difference = 0 (vs ≠): T-Value = 3.01 P-Value = 0.013, DF = 10							

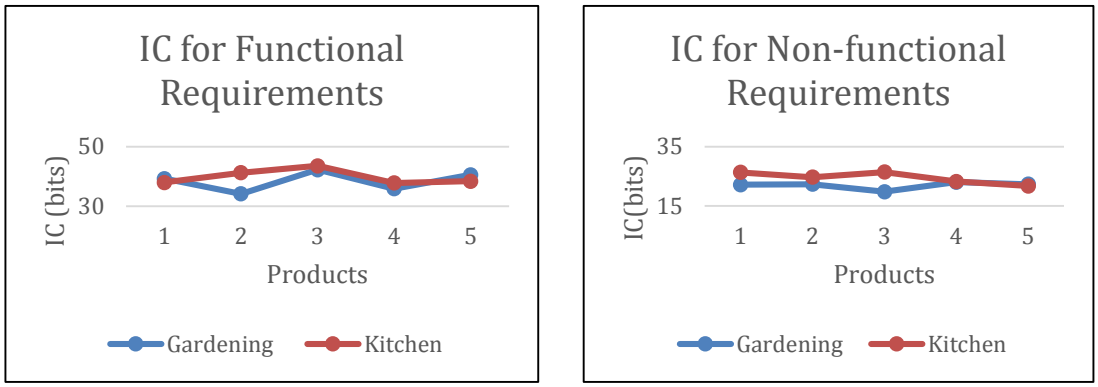

Figure 16: Plot of IC for functional and nonfunctional requirements for the two families of product

In order to verify if there exists any statistically significant difference between the information content of the two products families, a t-test was conducted for each type of requirement (i.e., functional and non-functional). It was assumed that both samples were normally distributed and standard deviations for both populations were equal. It was

hypothesized that the means of information content for both families of the products are the same. The results of the t-test are provided in Table 2 and Table 3. [Figure 16] show the plots of IC for functional and non-functional requirements. The average IC for functional requirement for all 10 data points vary between 34 and 44 bits whereas, for non-functional requirements this range is from 20 to 27 bits. Higher IC for functional requirements can be attributed to the more complicated structure of the functional requirement class in the ontology and deeper class structure.

3.7.1.1 Discussion:

Results of the t-test for functional requirements are summarized in [Table 9].

The calculated p-value was fairly large (0.202) therefore the null hypothesis is not rejected. Henceforth, with 95% confidence, we have strong evidence to conclude that mean of information content for functional requirements for both families of the products are the equals. For nonfunctional requirements, the calculated p-value was 0.013, which means the test statistics lies inside the rejection zone. Therefore, with 95% confidence, we have some evidence against the null hypothesis. Based on these two tests, it is not possible to arrive at a definitive conclusion about equality or inequality of the average information content of requirements for different families of products. However, given the relatively low-pooled standard deviation for the total IC in the studied sample, it can be concluded that products with similar complexities in terms of size, number of parts, and number of core functions, have similar ranges of IC for their requirements. Based on

this experiment, the average of total IC measured for sample products was 648 bits with a standard deviation of 63 bits. Therefore, a design team working on development of a similar product in terms of size and complexity, should expect to generate somewhere between 600 to 700 bits of information by the end of the requirement planning phase.

3.7.2 Experiment 2: Comparison of Performance of Different Senior Design Teams:

In this experiment, design requirements of three different products, from three different design teams were collected and compared with regard to the total information content of a product at initial stage of design. This experiment was intended to examine if there is any resemblance in total information content for diverse products. In order to do that, 3 different design teams are selected randomly and each team members were equally competitive and focused on design of one product. In their first week of design, requirements were gathered and refined in order to fit into the base ontology. All requirements were imported to the ontology and total information content for all products were measured using the developed tool. [Table 10] shows the summary of the result.

3.7.2.1 Discussion:

In this experiment three teams had three entirely different categories of products of different complexity. **[**Figure 17**]** indicates that the quantity of total information content of different design teams was significantly different. However, information content for each design team exhibited certain similar framework. For each team the design requirements were predominantly nonfunctional. This nature could be explained

by the fact that at the initial stage of design, teams only had little information that was regarding the physical attributes of the products rather than utilities. For example, Team#1 had 4 functional requirements and 9 nonfunctional requirements; Team#2 had 3 functional requirements and 5 nonfunctional requirements, whereas Team#3 had equal number (6) of functional and non-functional requirements. In Experiment-1, it was established that the functional requirements were more decisive and the average information content for each functional requirement was remarkably higher than the nonfunctional requirement irrespective of the category of the product. This clarifies the reason for highest information content value for Team#3 and lowest value for Team#2.

Table 10: Total information content for engineering design requirements of three different products, generated by three different design teams during their first week of design

Team	Product Name	No. of Req.	No. of Func. req.	Functional IC	No. of Non Func. Req.	Non Functional IC	Total IC
1	Bench Warmer	13	4	119.81	9	195.33	315.14
2	Robotic Drilling System	8	3	93.21	5	95.87	189.08
3	Ingot Growth Oven	12	6	200.49	6	161.71	362.2

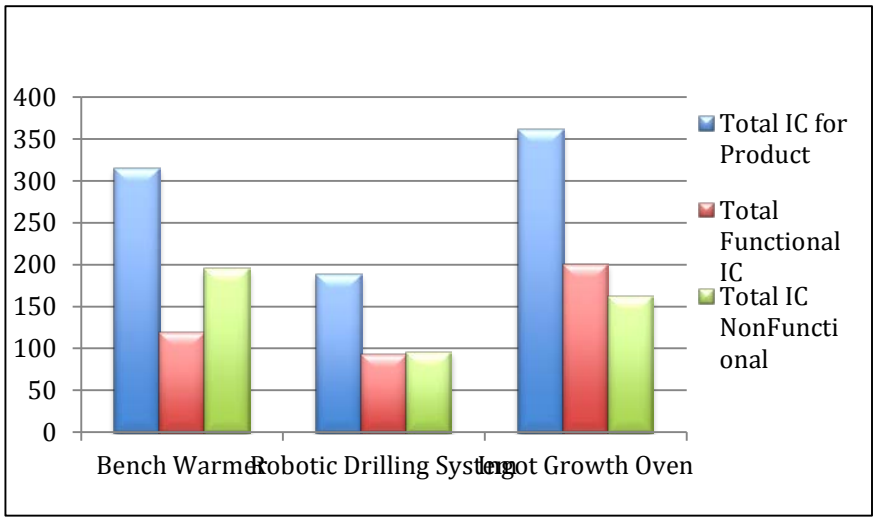

Figure 17: Comparison of information content of functional and nonfunctional requirements for three different products designed by three different design teams

Moreover it could be concluded that the total information content for functional requirements and nonfunctional requirements for every team is directly proportional to the number of functional and nonfunctional requirements, respectively. Consequently, it could be anticipated that the requirement gathering practice would be more rigorous in subsequent weeks and eventually the number of functional requirements would surpass the number of nonfunctional requirements. If it was possible to had equal number of functional and nonfunctional requirements for each team, then they might have similar value for the total information content. Therefore, it could not be affirmed that the total information content for a product fluctuates with expertise of design team; instead, it depends on the meticulousness of design progression.

3.7.3 Experiment 3: Evolution of Engineering Requirements

To validate the proposed ontology and the information metrics, an experiment was conducted for studying the evolution of requirements in a senior design project related to design of a flux measurement device. In this project, students had to update the requirements list for six consecutive weeks. The requirement list obtained by the end of the sixth week was considered to be the final list and was used as the reference for the rest of the project. The requirements list started with 13 requirement statements in week one and ended with 19 requirements statements by the end of week six. [Figure 18] shows the plot of IC from week one to week six. As can be seen in this figure, there is a steady growth from week one to week six in the information content of non-functional requirements from 219 bits in week one to 277 bits in week six. However, this is not the case for functional requirements as the IC declines from week three to five and then jumps back up in week six but still below the IC measured for week two.

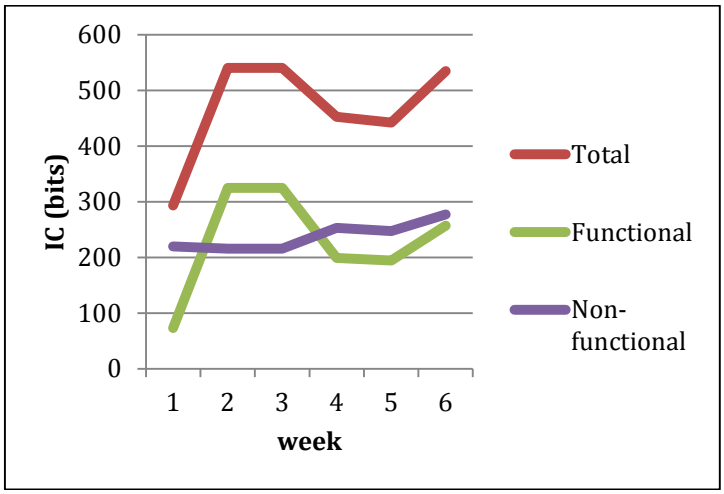

Figure 18: Plot for IC for flux measuring device for six consecutive weeks

3.7.3.1 Discussion:

Variation in the IC of the functional requirements can be attributed to the exploratory nature of the design project since the core functions of the product were not known a priori. In such projects, designers typically start with a set of functions as a "wish list" and then prune the list based on the exiting constraints to come up with more realistic and feasible design solutions. Therefore, it is very likely that some of the functions included in the initial set of requirements are eliminated in the final list or replaced by different functions. For example, in week 3 the team added, "The device must be able to measure inflow and outflow." However the team eliminated this requirement from their final set of requirements.

The steady growth observed in the information content of non-functional requirement can be an indication of gradual enrichment of the requirements as the designers learn more about the limitation and possibilities. This results in incremental information inflation each time the requirements list is updated.

For example, initially in week 2 the team included a non-functional requirement stating, " *The design must cost less than $500*." In week 3 they revised the same requirement, as "Material *and testing must cost less than $500*." Finally in week 7, they refined the requirement again and represented the requirement, as "Prototype *should cost less than $500 including materials and testing*." The results of this experiment reaffirm that the proposed method for information content measurement reflects the true fluctuations of information during the requirement-planning phase.

3.8 Conclusion:

In this chapter, a generic approach is proposed for measuring the information content of text-based design requirements by first translating the text to a form-neutral owl ontology model using protégé and then applying Shannon's information metrics to the model. Although the proposed approach is focused on engineering requirements, the underlying principles can be applied to any textual document created in design projects. It should be noted that the protocol presented in this work is tailored for requirement statements that already follow a semi-structured syntax and grammar. For more unstructured texts and non-textual information, such as those found in technical standards or service guidelines, a more complete set of protocols should be developed.

The metrics proposed in this work provide relative measures of the information content and should be treated as such. One explanation for the relative nature of the proposed measure for information content is that it varies with the size of the vocabulary captured. Therefore, the information content of the same requirement statement may change with time, depending on how many classes and instances exist in the ontology. For this reason, comparisons between two measure values of information content is meaningful only if they are calculated based on the same ontology.

There are multiple possibilities for extension of this work in the future. 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.

CHAPTER 4

REQUIREMENT SPECIFICITY, COMPLETENESS, AND CLASSIFICATION

4.1 Introduction:

This chapter describes the methods and metrics that are developed for automated classification and assessment of requirement statements in terms of completeness and specificity. An incomplete requirement statement is not useful for a designer and might be misleading. Also, requirements should be specific or informative enough to efficiently narrow down the design space. If a requirement statement is not informative enough, it won't serve as a useful constraint or criteria for the designer. Also, if the designer is provided with a set of highly specific requirements, then the designer won't have enough flexibility and freedom in exploring the solution space. Therefore, before handing off the requirements to the downstream design processes, they need to be validated through completeness and specificity analysis. The particular objective of the proposed method is to use ontological reasoning, enabled by semantic rules and axioms, for requirement assessment.

4.2 Related Work:

Requirement based research is mainly focused in three major areas: requirement statement, requirement document, and reasoning or queries with requirements (Joshi & Summers, 2014a). Research has been conducted to examine requirements as a statement with the intention of improving the quality of the requirement statement (Hooks, 1994;

Turk, 2006; Wiegers, 1999). Hooks (1994) (Hooks, 1994) demonstrated a set of required characteristics of a good requirement. In 1997, Wilson et al. (W. M. Wilson, Rosenberg, & Hyatt, 1997)compiled a list of nine indicators for quality attributes of requirement. Quality attributes are the characteristics that a good requirement should exhibit and these are considered as a baseline for analyzing natural language requirements. The quality indicators for individual requirement statements were based on frequency of using words, phrases and the structure. A tool was also developed by Software Assurance Technology center to measure the quality of the requirements based on the identified quality indicators.

In 1999, Wiegers (Wiegers, 1999) provides a comprehensive analysis of requirement writing procedure. He investigated some badly written requirements and showed how they can affect the overall health of the project. He also investigated several characteristics of high quality requirements and concluded that a good requirement should exhibit six important properties such as correctness, feasibility, verifiability, unambiguity, priority, and necessity. On the other hand, he identified completeness, consistency, modifiability and tractability as the characteristics of a quality requirement document as a whole.

In 2001, Fabrini et al. (Fabbrini, Fusani, Gnesi, & Lami, 2001) proposed a quality model for natural language requirement analysis and developed an automated tool "QuARS" to measure specificity of requirements. "QuARS" was made to detect the possible source of errors in a textual requirement and it was based on five logical

modules: “lexical analyzer, syntax analyzer, quality evaluator, special purpose grammar, and dictionaries”.

In another research on natural language requirements Lami (Lami, 2005) proposed a method and an automated tool to analyze the quality of natural language requirements in terms of consistency, completeness and ambiguity.

In 2009, Lamar (Lamar, 2009)described a method to analyze the natural language requirement linguistically and determined the “completeness, specificity, qualitativeness and quantitativeness”(Lamar, 2009)of engineering requirements. Lamar checked the “completeness” of a requirement statement based on the syntax proposed for requirements and for each complete requirement he measured specificity qualitatively. If a requirement is missing any component, it will become incomplete requirement and thereby increases vagueness. Hence, it is necessary to measure the completeness while analyzing requirements. In this study “specificity” was defined as “the amount of detail about a behavior or characteristic of a system or system component”. According to this study specificity is also directly proportional to the adjuncts. Further presence of numerical values increases specificity of functional requirements, whereas, presence of “Adjectival Noun” improved specificity of nonfunctional requirement.

In 2014, Joshi (Joshi & Summers, 2014b) introduced “completeness” and “specificity” as a baseline to measure the project health. To measure the completeness Joshi considered components such as “subject, modal and verb phrase” and specificity as count of “numbers” and “adjuncts”.

Though an extensive research was conducted on guidelines of writing 'good' requirements and analyzing natural language requirements in software engineering; analyzing requirements in mechanical engineering domain is little explored. Although several researches were conducted in a variety of perspective on analyzing natural language requirements, there is a lack of quantitative method. Hence, it is necessary to combine qualitative approach and quantitative approach and create a method that can analyze the natural language requirement in a qualitative manner and as well as can provide quantitative measure for each of the quality indicators.

4.3 Completeness and Specificity Measurement:

The proposed completeness and specificity measurement technique uses ontological representation of engineering requirements as the input. Requirement Ontology described in the REQUIREMENT ONTOLOGY is used to break down the requirement statement into its elements. For ontology conceptualization, a linguistic and grammatical approach was used. As discussed in the previous chapters, functional and nonfunctional requirements, both have some necessary components with limited cardinality and some optional components with or without any cardinality restrictions. However, the optional components are preferred in the ontology but not required to convert a requirement statement into consistent requirement ontology. They mostly enhance the quality of the requirement. Hence, it can be stated that the compulsory components contribute towards the completeness of a requirement statement, whereas the noncompulsory components improve the specificity or informativeness of a requirement.

4.3.1 Completeness:

Completeness analysis can be conducted through checking the presences of the necessary elements in the requirement statement. If a requirement contains all the required components, it is considered to be a complete requirement. [Table 11] and [Table 12] show all the essential properties of functional and nonfunctional requirements respectively. In Table 11 and Table 12, the domain for object properties *hasPrimarySubject* and *hasPrimaryObject* are *Subject* and *Object* respectively. These two classes are hypothetical and used in order to simplify Table 11. Both *Subject* and *Object* classes are the union of 7 other classes namely *Information, Energy, Material, Object, Part, Product*, and *User*.

Table 11: Cardinality restriction of essential properties for Functional Requirement

Property Name	Property Type	Range	Domain	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 12: Cardinality restriction of essential properties for Nonfunctional Requirement

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

One major assumptions of this research is that the requirements can be either complete or incomplete. Thus, the completeness score of requirements is either 0 (for an incomplete requirement) or 1 (for a complete requirement). To assign a score to each instance of the *Requirement* class, a Boolean data property termed *hasCompletenessScore* was introduced. The domain of the property *hasCompletenessScore* is *Requirement* class and its range is 1 or 0. A functional requirement will have a completeness score of 1, if it describes a behavior, which has exactly one product, has exactly one primary subject, has exactly one verb, has exactly one primary object, and it may have some (existential restriction) additional properties as well. A nonfunctional requirement will have a completeness score of 1 if it has exactly one product, has exactly one primary subject, and describes minimum one attribute. [Table 13] shows examples of complete functional and nonfunctional requirement.

Table 13: 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 - The suspension weighs between 5-10 lbs.		
hasProduct	Exactly 1	Suspension
hasPrimarySubject	Exactly 1	Suspension
describesAttribute	Minimum 1	Weight
hasUpperValue	Optional	10
hasLowerValue	Optional	5
hasUnit	Optional	Lbs.

Further, if a requirement has a completeness score of 1, then it will be considered as complete and the ontology reasoner will infer that particular instance of requirement as an instance of the complete class. Incomplete is the complement class of complete. In other words, *Complete* and *Incomplete* classes are mutually disjoint. Therefore, if a requirement does not belong to the *Complete* class, the reasoner will infer the requirement as an incomplete requirement. In this study, Pellet and HermiT reasoners are used. To infer if a requirement is a member of *Complete* or *Incomplete* class, SWRL (Semantic Web Rule Language) rules are used in the proposed ontology**. [**Equation 20, Equation 21, Equation 22] show the SWRL rules used in the ontology to determine if a requirement is complete. Since negation is not allowed in SWRL rules, a combination of

SWRL rules and java codes with OWL API was implemented to infer the incompleteness of a requirement [Equation 23and Equation 24].

$$FunctionalRequirement(?f) \ \wedge \ hasProduct(?f, ?x) \wedge describesBehavior(?f, ?b) \wedge \ hasPrimarySubject(?b, ?p) \wedge hasVerb(?b, ?v) \wedge hasPrimaryObject(?b, ?o) \rightarrow hasCompletenessScore(?f, 1) \quad \text{Equation 20}$$

$$NonFunctionalRequirement(?n) \ \wedge \ hasProduct(?n, ?x) \wedge describesAttribute(?n, ?a) \ \wedge hasPrimarySubject(?n, ?p) \rightarrow hasCompletenessScore(?n, 1) \quad \text{Equation 21}$$

$$Requirement(?x) \ \wedge \ hasCompletenessScore(?x, 1) \rightarrow \ Complete(?x) \quad \text{Equation 22}$$

$$X \leftarrow Requirement \ ; \ Y \leftarrow hasCompletenessScore[X] \ ; \ while(Y.isEmpty)\{ \ do \ Y = 0 \ \} \quad \text{Equation 23}$$

$$Requirement(?x) \ \wedge \ hasCompletenessScore \ (?x, 0) \rightarrow \ Incomplete(?x) \quad \text{Equation 24}$$

4.3.2 Specificity:

Specificity score of an incomplete requirement is meaningless and unnecessary. If a requirement is incomplete, it needs to be revised by the designer and no further quality evaluation is required. Therefore, the first step is to check if a requirement is complete or incomplete.

4.3.2.1 Specificity Criteria of Functional Requirement:

For complete functional requirements, five criteria were considered and weighted to measure the specificity. A weight is assigned to each criterion because for a functional requirement, existence of secondary subject can be more important than the number of adjuncts. By including a secondary subject may include a specific function of a part of the product, whereas, addition of adjuncts can provide more details about the function. In comparison, changing the secondary subject can significantly change the design solution, but changing or deleting an adjunct will marginally change the design.

Criteria 1. Existence of Secondary Subject
Criteria 2. Depth of Verb
Criteria 3. Existence of Secondary Object
Criteria 4. Number of Adjunct
Criteria 5. Existence of a Measure Adjunct

Criteria 1(Existence of Secondary Subject): Existence of a secondary subject can add more details into a requirement and make it more specific or informative. A requirement with a primary subject and a secondary subject can be more specific than a requirement comprising a primary subject only. For example, the requirement statement "Hand Truck has a base pad that moves many different sized object", has a primary subject "base pad" and a secondary subject "Hand Truck"; whereas the requirement statement "Hand Truck holds large/odd shape loads securely", has only primary subject "Hand Truck". The first requirement describes a function of a part of the product and the second requirement describes a function of the product itself. Hence, it can be reasoned that the presence of secondary subject adds more specificity to the requirement and makes it more

informative. As discussed before, a more specific requirement is not necessarily preferred over a less specific requirement. In the Hand Truck example, the more specific requirement forces the designer to incorporate a "base pad" into the design.

Criteria 2(Depth of Verb): The depth criterion is used as a measure of specificity since it can be argued that deeper classes in taxonomy are more specific than top-level classes. A class with a more elaborate and deeper sub-class structure poses more uncertainty. The proposed ontology uses hierarchical structure for some of classes such as Verb, Energy, Adjunct, and Attribute etc. Since, the functional requirement illustrates one or more function of a product and it is impossible to explain any function without using an appropriate verb, the Verb class can be considered as the most significant constituent for a functional requirement.

To measure the specificity, only the structure of the verb class is considered and stipulated depth of the verb is used for measurement. In the requirement ontology, a comprehensive classification of verb is adopted from functional basis (Sen et al., 2010b)Therefore, if the verb of a functional requirement can be classified under any of those subcategories, the requirement will definitely be more specific and adding depth will further increase the specificity.

Criteria 3(Existence of Secondary Object): For this criterion, it is assumed that the presence of a secondary object increases the specificity and informativeness of a requirement. 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”. If the secondary object is eliminated from the statement it will become (“The suspension preserves the steering characteristics”) and it conveys less specifics than the previous requirement. Hereby, secondary objects can be considered as another important contributor to the specificity of a functional requirement statement.

Criteria 4(Number of Adjunct): The specificity of functional requirements can be further improved by using adjunct. An adjunct usually modifies the 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 `AdverbialAdjunct` while “on steep slopes” is a `LocativeAdjunct`. “Hand truck” and “box” are subject and object respectively. As the requirements evolve, designers add more details to the requirement through introducing various types of adjuncts. Therefore, if the number of adjuncts increases, the specificity will also increase. It is assumed that a requirement typically has 2 adjuncts, which improves the specificity significantly but beyond that, specificity turns out to be stagnant.

Criteria 5(Existence of Measure Adjunct): This criterion is included based on the assumption that if a functional requirement includes an `Adjunct` of type `Measure Adjunct`, then it will be more specific rather than having any other type of `Adjunct`. It can be argued that specific numeric values and units are more informative. In the requirement statement “The suspension has a maximum vertical deflection at the seat

mount of 8 mm at 250 lb. static load", by including the `MeasureAdjunct` "at 250 lb. static load", the requirement is pointing to an important design variable with important implications for the final design. For example if static load quantity was replaced by, say, 500 lb., the design solution could be significantly altered. For example, removing "during hard cornering" (`TemporalAdjunct`) from the requirement statement "The suspension remains rigid during hard cornering" will not affect the design severely.

4.3.2.2 Specificity Criteria of Nonfunctional Requirement:

For complete nonfunctional requirements, four criteria are considered to measure the specificity.

Criteria 1. Existence of Secondary Subject
Criteria 2. Number of Attributes
Criteria 3. Number of Adjuncts
Criteria 4. Type of Attributes

Criteria 1 and Criteria 3:

Criteria1 and criteria3 are similar to the Criteria 1and Criteria 4 for functional requirements, respectively.

Criteria 2 (Number of Attributes): For nonfunctional requirements, `Attribute` is a set of parameters that characterize the entities. The main purpose of nonfunctional requirements is to describe quality characteristics of the product. For example, attributes such as height, weight, size, safety, ease of maintenance, affordability, usability, availability etc. are not the features of the product, but they are product's characteristics. It is impossible to write a specific nonfunctional requirement without any attributes.

Therefore, if a nonfunctional requirement defines two attributes, it is considered to be more precise and unambiguous in comparison to another requirement describing a single attribute. Diminishing ambiguity and increasing precision increases the specificity and informativeness of requirements. Hence, it can be asserted that the number of attributes is an important factor for measuring the specificity of a nonfunctional requirement.

Criteria 4 (Number of Adjuncts): During requirement collection process, customer may ask for a product, which has higher quality than the previous one without specifying in which basis they are measuring the quality of the product. Perhaps, the customer can say that they need a product, which is light in weight. For existing products it is not a huge problem, because the designer can rephrase the need and add a value based on their experience as they have a baseline for product's weight, which is the weight of the existing model of the product. But for novel products, this type of requirements does not convey much information. Similarly, the specificity also relies on the type of the attribute. It is presumed that the specificity of `QuantityAttribute` will be higher than the specificity of a `QualityAttribute` or a `BooleanAttribute`, as `QualityAttribute` and `BooleanAttribute` convey fewer details than `QuantityAttribute` and might not impose any rigorous boundaries on design parameters. Replacing or removing any quality or Boolean attribute will not modify the design significantly but revising a quantity attribute can alter the design solution. For example, "The suspension is light" enforces a loose constraint on design parameter

"weight", whereas the requirement "The weight of the suspension is less than 10 lbs" confine the "weight" with an upper threshold restriction. Further, "The suspension weighs between 5-10 lbs." will be more specific than the previous one since, it constraints the weight in between a maximum and a minimum value. Furthermore, the same requirement will be much more specific if it has an exact numeric value such as "The weight of the suspension is 6 lbs." Therefore, it is evident that the type of the attribute and its value influence the specificity of a nonfunctional requirement.

4.3.2.3 Specificity Score of Functional and Nonfunctional Requirement:

[Figure 19**]** demonstrates the flow diagram of completeness and specificity measurement method.

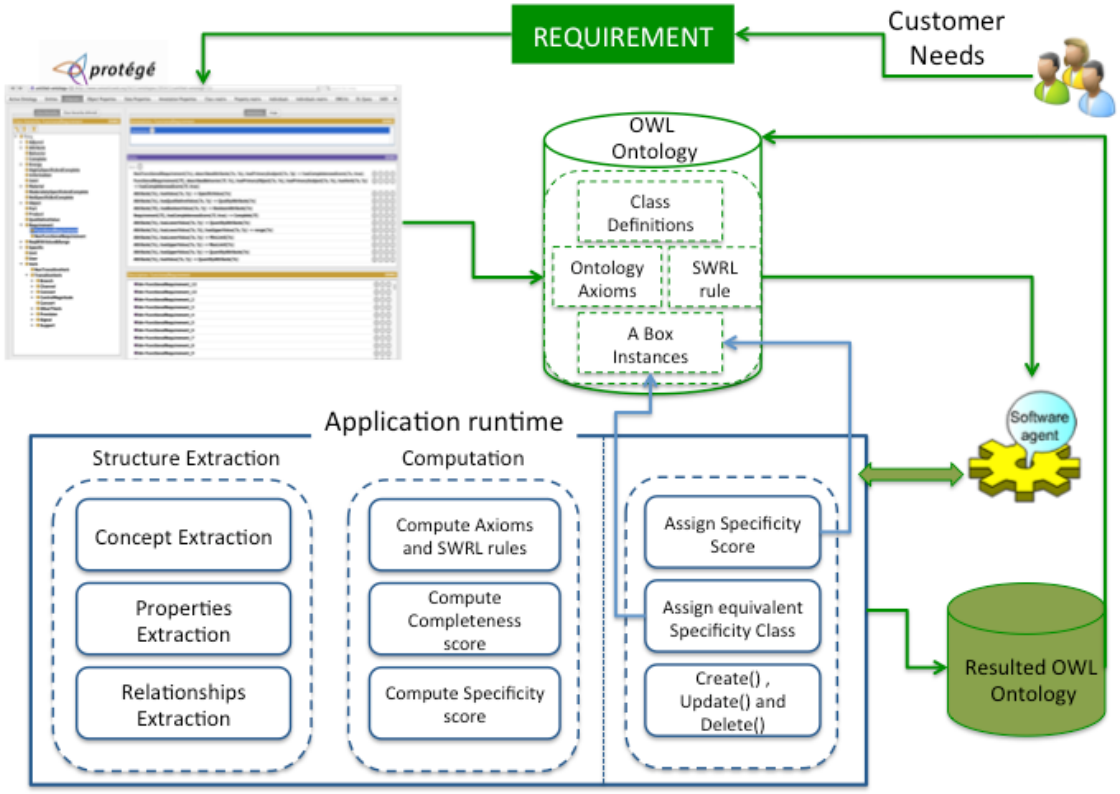

Figure 19: Flow chart for Specificity and Completeness measurement using java application

The specificity of functional and nonfunctional requirement is quantified through a specificity score. A continuous scale of 0-1 is used to represent the specificity score of the requirements. As discussed before, different criteria influence specificity score differently. [Table 14] and [Table 15] show the algorithm for calculating the specificity score of functional requirement and nonfunctional requirements respectively.

4.3.2.3.1 Functional Requirement

Different importance factors can be assigned to different criteria depending on their perceived importance. . An arbitrary scale of weight can be used, as long as the

scale is applied to all requirements and the scores are used only for comparison, instead of an absolute measure of specificity. In this paper, weight (W_i) of 5, 4, 3, 2, and 1 is assigned to criteria 1, criteria 2, criteria 3, criteria 4, and criteria 5, respectively. The final result will be in a range of 0-1 continuous scales

Table 14: Algorithm to calculate Specificity score of Functional Requirement

Algorithm 1.

$Individual$
$: R \quad Types: Functional\ Requirement, describesBehavior\ value\ "behavior1"$
$Individual : behavior1, \ Types: Behavior$
$\boldsymbol{if} \left(\begin{matrix} behavior1\ hasPrimarySubject\ \boldsymbol{exactly}\ \mathbf{1}\ Subject\ and \\ hasSecondarySubject\ \boldsymbol{some}\ Subject \end{matrix} \right) \{$
$score_i = 20 \};$
$\boldsymbol{else\ if}\ (behavior1\ hasPrimarySubject\ \boldsymbol{exactly}\ \mathbf{1}\ Subject\ )\{$
$score_1 = 10\};$
$\boldsymbol{else}\ score_1 = 0;$
$\boldsymbol{return}\ score_1\ ;$

Algorithm 2.

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

Algorithm 3.

$Individual$
$: R \quad Types: Functional\ Requirement, describesBehavior\ value\ "behavior1"$
$Individual : behavior1, \ Types: Behavior$
$\boldsymbol{if} \left(\begin{matrix} behavior1\ hasPrimaryObject\ \boldsymbol{exactly}\ \mathbf{1}\ Object\ and \\ hasSecondaryObject\ \boldsymbol{some}\ Object \end{matrix} \right) \{$
$score_i = 20 \};$
$\boldsymbol{else\ if}\ (behavior1\ hasPrimaryObject\ \boldsymbol{exactly}\ \mathbf{1}\ Object\ )\{$
$score_i = 10\};$
$\boldsymbol{else}\ score_3 = 0;$
$\boldsymbol{return}\ score_3;$

Table 14-Continued: Algorithm to calculate Specificity score of Functional Requirement

Algorithm 4.

$Individual$
$: R \quad Types: Functional\ Requirement\ , describesBehavior\ value\ "behavior1"$
$Individual : behavior1\ ,\ \ Types: Behavior\ \ , hasAdjunct\ \boldsymbol{some}\ Adjunct$
$Literal : n\ \ xsd: integer$
$n \leftarrow Count\ of\ Adjunct$
$\boldsymbol{if}\ (n < 3)\{score_4 = 10 * n\}\ ;$
$\boldsymbol{else}\ score_4 = 20\};$
$\boldsymbol{return}\ score_4\ \ ;$

Algorithm 5.

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

Algorithm 6.

$$Total\ Score = \sum_{i-1}^{5} W_i\ \times\ Score_i \qquad \text{Equation 25}$$

$Total\ Score = 5 \times score_1\ \times +\ 4 \times score_2\ +\ 3 \times score_3 +\ 2 \times score_4\ + 1 \times score_5$

$Total\ Score =\ W_1\ \times score_1\ \times +\ W_2\ \times score_2\ +\ W_3\ \times score_3 +\ W_4 \times score_4\ + W_5\ \times score_5$

$Maximum\ possible\ Total\ Score = 370$

$After\ Normalizing,$

$$\boldsymbol{Specificity\ \ Score}\ = \frac{\boldsymbol{Total\ Score}}{\boldsymbol{370}}$$

Equation 26

Table 15: Algorithm to calculate Specificity score of Nonfunctional Requirement

Algorithm 1. $Individual: R \quad Types: NonFunctional\ Requirement$ $Individual: behavior1,\ Types: Behavior$ $\boldsymbol{if}\ \begin{pmatrix} R\ hasPrimarySubject\ \boldsymbol{exactly}\ \mathbf{1}\ Subject\ and \\ hasSecondarySubject\ \boldsymbol{some}\ Subject \end{pmatrix}\ \{$ $score_i = 20\ \};$ $\boldsymbol{else\ if}\ (R\ hasPrimarySubject\ \boldsymbol{exactly}\ \mathbf{1}\ Subject\ )\{$ $score_1 = 10\};$ $\boldsymbol{else}\ score_i = 0;$ $\boldsymbol{return}\ score_1\ ;$
Algorithm 2. $Individual$ $: R \quad Types: NonFunctional\ Requirement\ , describesAttribute\ \mathbf{min\ 1}\ Attribute$ $Literal: n\ \ xsd: integer$ $n \leftarrow Number\ of\ Attribute$ $\boldsymbol{if}\ (n < 3)\{score_4 = 10 * n\};$ $\boldsymbol{else}\ score_2 = 20\};$ $\boldsymbol{return}\ score_2\ ;$
Algorithm 3. $Individual: R \quad Types: NonFunctional\ Requirement\ , hasAdjunct\ \boldsymbol{some}\ Adjunct$ $Literal: n\ \ xsd: integer$ $n \leftarrow Count\ of\ Adjunct$ $\boldsymbol{if}\ (n < 3)\{score_4 = 10 * n\};$ $\boldsymbol{else}\ score_3 = 20\};$ $\boldsymbol{return}\ score_3\ ;$

Table15-Continued: Algorithm to calculate Specificity score of Nonfunctional Requirement

Algorithm 4.

$Individual$: R $Types$: $NonFunctional\ Requirement$, $describesAttribute$ **min 1** $Attribute$
$Individual$: z $Types$: $QuantityAttribute$, $hasValue$ **eactly 1** $Literal$ $\boldsymbol{OR}$ $hasUpperValue$ $\boldsymbol{exactly}$ **1** Literal **OR** hasLowerValue **exactly 1** Literal **OR** hasRange **some** I
$\boldsymbol{if}$ ($z\ hasLowerValue$ $\boldsymbol{eaxctly}$ **1** $Literal$ | $hasUpperValue$ $\boldsymbol{exactly}$ **1** $Literal$) {
$score_4 = 10$ };
$\boldsymbol{else\ if}$ ($z\ hasLowerValue$ $\boldsymbol{eaxctly}$ **1** $Literal$ && $hasUpperValue$ $\boldsymbol{exactly}$ **1** $Literal$) {
$score_4 = 20$ };
$\boldsymbol{else\ if}$ ($z\ hasValue$ $\boldsymbol{eaxctly}$ **1** $Literal$) {
$score_4 = 30$ };
$\boldsymbol{else}$ $score_4 = 0$ };
$\boldsymbol{return}$ $score_4$;

Algorithm 5.

$$Total\ Score = \sum_{i-1}^{4} W_i \times Score_i \qquad \text{Equation 27}$$

$$Total\ Score = W_1 \times score_1 \times + W_2 \times score_2 + W_3 \times score_3 + W_4 \times score_4$$

$$Total\ Score = 4 \times score_1 \times + 3 \times score_2 + 2 \times score_3 + 1 \times score_4$$

$Maximum\ possible\ Total\ Score = 170$

$After\ Normalizing,$

$$\boldsymbol{Specificity\ Score} = \frac{\boldsymbol{Total\ Score}}{\boldsymbol{170}} \qquad \text{Equation 28}$$

4.3.2.3.2 Nonfunctional Requirement

Similar to the functional requirement, weights based on perceived importance are assigned to each of these four mentioned criteria for nonfunctional requirement. Different weights are assigned to the criteria depending on the specific design problem and designer. An arbitrary scale of weight is used, the scale is applied to all requirements

and the scores are used only for comparison, instead of using them as an absolute measure of specificity. In this paper, a weight (Wi) of 4, 3, 2, and 1 is assigned to criteria 1, criteria 2, criteria 3, and criteria 4, respectively.

$$Requirement(?x) \wedge hasSpecificityScore(?x, ?y) \wedge greaterThanOrEqual(?y, 0.75) \rightarrow HighlySpecific(?x) \quad \text{Equation 29}$$

$$Requirement(?x) \wedge hasSpecificityScore(?x, ?y) \wedge greaterThanOrEqual(?y, 0.5) \wedge lessThan(?y, 0.75) \rightarrow ModeratelySpecific(?x) \quad \text{Equation 30}$$

Using above-mentioned method, specificity of functional and nonfunctional requirements can be measured on a continuous scale between 0 and 1.

Using the proposed score, requirements can be classified into three classes: Highly Specific, Moderately Specific and Not Specific. A requirement having a specificity score greater than or equal to 0.75 is considered as highly specific, specificity score greater than equal to 0.5 but less than 0.75 is categorized as moderately specific, and specificity score less than 0.5 is considered to be Not Specific requirement. In the ontology, SWRL rules (Equation 29, Equation 30) are used to determine the equivalent specificity class of a requirement.

4.4 Result:

The size of the requirement ontology is dynamic and till now it contains around 247 functional requirements and 182 nonfunctional requirements for 27 different products. Measuring completeness and specificity of these 429 requirements manually is tedious and time consuming. Therefore, java tool is developed based on the proposed method for completeness and specificity measurement. OWL API is used to run through the structure of the ontology and access information from the ontology. With the help of the SWRL rules and software application, completeness and specificity scores are calculated and assigned to two empty properties *hasSpecificityScore* and *hasCompletenessScore*.

Table 16: Example of Completeness and Specificity Score Calculation

Functional Requirement: The suspension enables high speed descents on bumpy trails					
Condition Type/Decision	**Condition**	**Outcome → true/false**	**Basic Score (s)**	**Weight(w)**	**Wt Score (w X s)**
Completeness	hasProduct	True (Suspension)	1	N/A	1
	hasPrimarySubject	True (Suspension)			
	describesBehavior	True (high speed descents on bumpy trails)			
	hasVerb	True (enable)			
Decision	Is Complete and equivalent to Complete class				
Specificity Condition	hasPrimarySubject **AND** hasSecondarySubject	False	0	5	0
	hasPrimarySubject	True (Suspension)	10	5	50
	Level of Verb (n)	n = 3 (Actuate →Control Magnitude → Transitive Verb → Verb)	30	4	120
	hasPrimaryObject **AND** hasSecondaryObject	False	0	3	0
	hasPrimaryObject	True (descents)	10	3	30
	Number of Adjuncts (n)	n = 2 (bumpy trails , high speed)	20	2	40
	n>=3	False			
	Individual: X, Type: Measure Adjunct; behavior hasAdjunct X ,	False (High Speed → Adverbial Adjunct, Bumpy Trails → Locative Adjunct)	0	1	0
				Total Score	**240**

Table 16-Continued: Example of Completeness and Specificity Score Calculation

				Specificity Score (Total Score/370)	**0.648**
Nonfunctional Requirement: The suspension weighs between 5-10 lbs.					
Complete-ness	hasProduct	True (Suspension)	1	N/A	1
	hasPrimarySubject	True (Suspension)			
	describesAttribute	True (Weight)			
Decision	Is Complete and equivalent to Complete class				
Specificity Condition	hasPrimarySubject AND hasSecondarySubject	False	0	4	0
	hasPrimarySubject	True	10	4	40
	Number of Attribute (n)	n=1 (Attribute→ Weight)	10	3	30
	n<=3	True			
	Number of Adjunct (n)	n=0	0	2	0
	n<=3	True			
	Attribute → hasUpperValue **OR** hasLowerValue	True(Weight hasUpperValue=10 lbs AND haLowerValue = 5 lbs.)	0	1	0
	Attribute → hasUpperValue **AND** hasLowerValue	True(Weight hasUpperValue=10 lbs AND haLowerValue = 5 lbs.)	20	1	20
	Attribute → hasValue	False	0	1	0
				Total Score	**90**
				Specificity Score (Total Score/170)	**0.53**

4.5 Case Study: Bike Suspension

Column 2 in Table 17 shows the requirements for a bike suspension found in an engineering design text (Ulrich, 2003) and modified to add some complexity in the form of numeric constraints and conditional statements. These textual requirements are imported to the proposed requirement ontology and completeness and specificity scores are measured using the described method Specificity Score of Functional and Nonfunctional Requirement: Column 3, 4 and 6 of [Table 17] shows the derived completeness class, specificity score and equivalent specificity class for these requirements, respectively. [Figure 20] shows the comparison of specificity of functional and nonfunctional requirement for bike suspension.

Table 17: Example of textual requirements: bike suspension

Sl. No	Requirement Statement	Complete/ Incomplete	Specificity Score	Equivalent Specificity Class
	Functional Requirement			
1.	The suspension instills pride	C	0.51	Moderate
2.	The suspension works with fenders	C	0.54	Moderate
3.	The suspension fits a wide variety of bikes	C	0.59	Moderate
4.	The suspension fits a wide variety of tires.	C	0.59	Moderate
5.	The suspension can carry riders weighing up to 250 lbs	C	0.62	Moderate
6.	The suspension preserves the steering characteristics of the bike	C	0.62	Moderate

Table 17 - Continued: Example of textual requirements: bike suspension

Sl. No	Requirement Statement	Complete/ Incomplete	Specificity Score	Equivalent Specificity Class
7.	The suspension enables high speed descents on bumpy trails	C	0.64	Moderate
8.	The suspension provides stiff mounting points for the brakes	C	0.67	Moderate
9.	The suspension reduces vibration to the hands	C	0.72	Moderate
10.	The suspension fits a wide variety of wheels	C	0.72	Moderate
11.	The suspension allows easy replacement of worn parts	C	0.72	Moderate
12.	The suspension allows easy traversal on slow difficult terrain	C	0.75	Highly Specific
	Nonfunctional Requirement			
13.	The suspension lasts a long time	C	0.47	Not Specific
14.	Suspension allows sensitivity adjustment	C	0.47	Not Specific
15.	The suspension is easy to install	C	0.47	Not Specific
16.	The suspension can be easily accessed for maintenance	C	0.47	Not Specific
17.	Suspension is not contaminated by water	C	0.47	Not Specific
18.	The suspension weighs between 5-10 lbs.	C	0.52	Moderate
19.	The suspension remains rigid during hard cornering	C	0.58	Moderate
20.	The suspension can be maintained with readily available tools	C	0.7	Moderate
21.	Suspension is affordable for an amateur enthusiast	C	0.7	Moderate

Table 17 - Continued: Example of textual requirements: bike suspension

Sl. No	Requirement Statement	Complete/ Incomplete	Specificity Score	Equivalent Specificity Class
22.	The suspension has a maximum vertical deflection at the seat mount of 8 mm for 250 lb static load.	C	0.7	Moderate
23.	The suspension has a maximum vertical deflection at the seat mount of 5 mm for a 200lb static load	C	0.7	Moderate

Result shows that in a set of 23 requirements, composed of 12 functional requirements and 11 nonfunctional requirements, all requirements are complete. 11 out of these 12 functional requirements have a specificity score in a range of 0.36 to 0.48 and equivalent to Moderately Specific class, whereas, the remaining functional requirement is highly specific with a specificity score of 0.75.

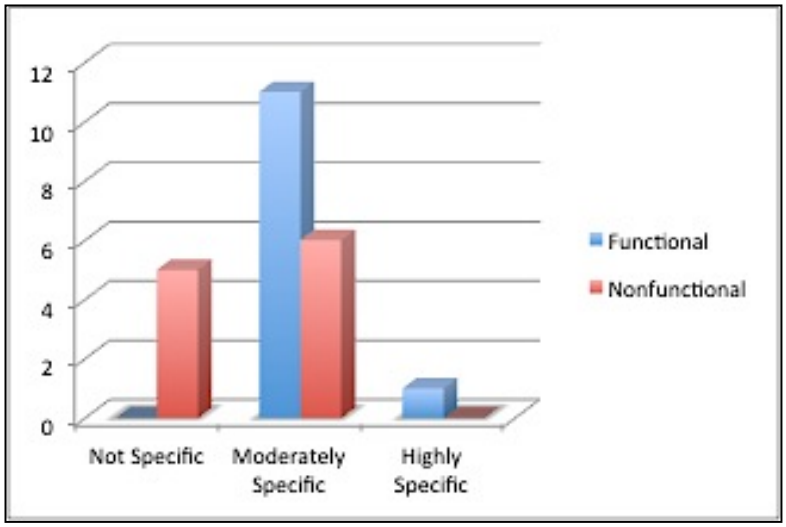

Figure 20: Comparison of Specificity of Functional and Nonfunctional Requirement for Bike Suspension requirements

Among the nonfunctional requirements, 5 requirements have a specificity score of 0.47 and equivalent to Not Specific class, and the remaining 6 requirements have a specificity score in a range of 0.52 to 0.7 and equivalent to Moderately Specific class. As discussed before, highly specific requirements do not necessarily improve the quality, novelty, or variety of final designs. Studying the impact of requirement specificity on the idea generation process is an important research problem, which is outside the scope of this work. In this research we are merely dealing with specificity (or informativeness) quantification.

4.6 Classification of Requirement:

Requirements are directly connected to design solutions. Thus, creating a structured requirement dictionary is essential effective design reuse. Currently there are multiple requirement management tool exist in the market such as "DOOR", "InteGREAT", "Blueprint Requirement Center", "Caliber" "Code beamer Requirement Management" etc. (Carrillo de Gea, Juan M et al., 2011) but they are intended to manage requirements for software industry. For engineering design, there are no such tools available in the market where requirements can be stored in an organized manner and can be retrieved easily. In this work, we propose an ontological approach to requirement classification. To increase the reusability of the requirements, it is important to store the requirements according to their type, rather storing them directly under *Functional* or *Nonfunctional Requirement* class. For example, a requirement statement "The

suspension can be easily accessed for maintenance" can be a member of a separate class *Maintenance* instead of simply placing them under *Nonfunctional Requirement* class. Therefore, in order classify different types of requirements nineteen distinct classes are introduced into the ontology. [Figure 21] shows the different categories of functional and nonfunctional requirement. Categorizing requirements will also help in mapping requirements from the ontology by running any reasoner (Pellet or HermiT, and FaCT++ etc.). Further, results can be sorted by using SPARQL queries. All the classes are overlapping and partially constrained.

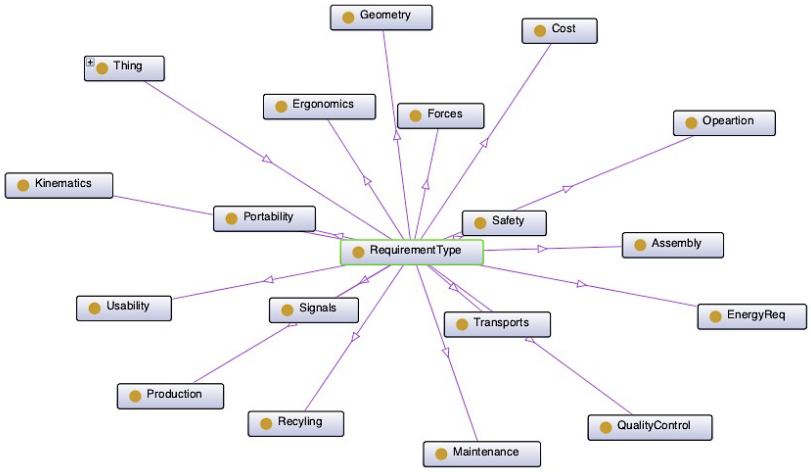

Figure 21: Classification of Requirement

4.6.1 Class Definition:

Requirements place restrictions on every aspect of the product such as the geometry, features, energy requirements, operation, manufacturing and assembly process,

safety, maintenance, reliability, reusability, lifecycle, and overall quality of the product. Each of these categories is represented as a subclass of Requirement Type. Each subclass contains certain type of requirements depending on their definition (Paul & Beitz, 1984).

Assembly: This class contains requirement those are directly or indirectly related to the assembly process of the product. Assembly requirement can be about the complexity of the assembly process or the type of assembly or if it is manual or automated etc. Therefore, if a requirement contains information about installation procedure, assembly method, any special regulation or guidelines, sitting or foundations etc. then it can be considered as a member of assembly requirement**.**

Cost: This class contains requirements related to the price or affordability of the product. Requirement can contain maximum permissible selling cost, material and tooling cost, manufacturing cost, depreciation, maintenance cost, cost of parts, prototype cost etc. Since, cost is an attribute of the product or part, the requirements comprising price details are mostly nonfunctional requirements. Hence, ontology can only accept nonfunctional requirements as cost requirements.

Energy Requirement: This class comprise requirements, which outlines the energy needs of the product or any part of the product, energy conversion during any function performed by the product, product performance parameters in terms of energy such as efficiency, consumption, frictional loss, ventilation, pressure, temperature, heating, cooling, capacity etc.

Ergonomics: This class represents requirements associated with the shape of the product, degree of comfort that a product should achieve, compatibility of the product to work with any other product or parts or environment etc.

Features: As the name of the class implies, it includes requirement that convey information regarding to the feature of the product.

Force: If a requirement will includes details about to the magnitude or direction of force, frequency, weight, load, deformation, stiffness, hardness, rigidity, elasticity, inertial forces or resonance etc., then it will be a member of force requirement.

Geometry: Requirements in this class explains the geometry of the product such as size, length, breadth, width, depth, height, space requirement, arrangement, connection etc.

Kinematics: This class represents requirement, which defines the type of motion of the product, direction of motion, velocity, acceleration etc.

Maintenance: Maintenance requirements recognize the needs pertaining to the servicing, servicing intervals if any, inspection, painting, cleaning, repairing etc.

Operation: Operation related requirements suggest functions need to be performed by the product, any operational parameters such as quietness, wear and tear and if the product has any special uses etc.

Portability/Mobility: Portability requirements are the requirements that describe the extent to which the customer can move the product anywhere without taking much effort.

Production: It involves requirement that indicates factory limitations if any, maximum permissible dimensions, and preferred production method, means of production, achievable quality and tolerances, wastage etc.

Quality Control: This class comprehends requirement that imitates the needs related to the testing and measuring, application of special regulations and standards, accuracy etc.

Recycling: Recycling requirements signify the needs for reuse, reprocessing, disposal, and storage etc.

Safety: This class includes requirements that identify needs pertinent to operational and environmental safety.

Signals: The requirements under this class indicate details of inputs and outputs, form, display, control equipment etc.

Transport: Transport requirements describe if there are any limitations due to lifting gear or wheels, means of transport etc.

Usability: Usability requirements involve the needs related to the user friendliness of a product such as ease of use, ease of learning, effectiveness of the product, error tolerant and user satisfaction level etc.

Setting up the type of requirement for a huge set of requirements manually is time taking and erroneous. The main issue associated with it is to identify the proper category and maintain the consistency of the ontology. In order to reduce manual efforts, the type of the requirements can be derived automatically using ontology reasoning method. In the developed ontology, functional and nonfunctional requirements both can be classified

under the requirement categories. Since, the requirement types are overlapping and partially constrained, a requirement can be part of two or more categories and also it is not necessary for each requirement to be a part of any of that requirement category. Two types of reasoning technique are used in this research –

1) Reasoning based on *Attributes* for *Nonfunctional Requirement*

2) Reasoning based on *Verb* for *Functional Requirement*

4.6.2 Ontology Reasoning based on Attribute:

Attributes are the inherent characteristics of a product or part. Each nonfunctional requirement in a requirement set describes one or more distinct attributes and the class of the requirement somewhat depends on the nature of the attributes specified by the requirement statement. In REQUIREMENT ONTOLOGY attributes are classified as quality attribute, quantity attribute or Boolean attribute according to their on 1their value. Further, attributes can be classified again according to their nature. The classification applied in this research involves two subclass *Tangible Attribute* and *Intangible Attribute*. Tangible attributes are the concrete, physical and objective characteristics of a product, whereas, intangible attributes are abstract, favorable and subjective attributes. (Lefkoff-Hagius & Mason, 1990). For example, a phone can be light, black and affordable. Among these three attributes, light and black indicates weight and color attributes those are tangible and affordable is intangible. Weight and color describes some physical properties of the product, but affordable is

mainly imaginary and favorable aspect of the product. These two subclasses are again subdivided into several other subclasses. [Table 18] shows the subclasses under *Tangible Attribute* and *Intangible Attribute.* Each subclass under *Attribute* is disjoint with all other sibling classes. Adding more subclasses add more structure to the ontology, however, it creates more complexity. For example, if *Attribute* class is divided into two different categories based on their value (*Quality*, *Quantity* and *Boolean*) and based on their nature (*Tangible* and *Nontangible*), the designer will have the responsibility to specify the type of each attribute according to these two different criteria.

Using reasoning can resolve the problem. In this study, keyword matching approach is used to derive the type of the attributes (Ducatel, Cui, & Azvine, 2006). A software tool is developed using JAVA and OWL API to determine the type of the requirement based on the attribute. The tool runs through the ontology and accesses each element of the ontology. In the runtime, it will analyze each instance of *Attribute* class and match them with similar words or synonyms from a set of predefined keywords. For each subclass of *Attribute,* a set of potential attributes are used for logical reasoning such as for *Weight* class "weight", "light", "heavy" etc. are used as keywords. After determining the type, the application will also assert the appropriate type of each instance of the attribute by ontology class assertion axiom e.g. User enters a requirement statement "The suspension can be maintained with readily available tools"

and asserts ''*isEasyToMaintain*'' as an attribute. The tool will match the attribute with the keyword maintain and assert *Maintenance* as the type of the attribute. The keyword matching method used in this study can accommodate plurals, stemming, synonyms, upper and lower case but it is sensitive to spelling. One major assumption for this technique was that the user would spell correctly while entering the attribute instance into the ontology. Subsequently, a basic assumption of this technique is associated with the naming convention. It is assumed that the user will use suitable names for the instances of the *Attribute*. Ontology can only have unique names of the classes and instances. Thereby, if an attribute is used in more than one requirement statement, appropriate suffix or prefix should be added in accordance with product name. Suppose, an attribute weight is used for two requirement statements ("The suspension weighs between 5-10 lbs" and "The phone weighs between 1-2 lbs"), and both of the attributes are same with a different values and are associated with two different products suspension and phone. The attribute weight can be written as sus_weight with a range 5-10 lbs for suspension and phone_weight with a range 1-2 lbs for phone instead of using weight and assigning two distinct sets of value to it.

Table 18: Attribute and Requirement Relation

Attribute Subclass	Keywords used	Related Requirement Type
Tangible Attributes		
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

Table 18-Continued: Attribute and Requirement Relation

Attribute Subclass	Keywords used	Related Requirement Type
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
Response Time	Time, response, prompt, quick etc.	Operation
Availability	Availability, accessibility, accessible, available, convenience, reachable, reachability, handy, feasible, feasibility, obtainable, obtainability etc.	Cost
Comfort	Comfort, comfortable, pleasant, comfy, satisfaction, relief, enjoy etc.	Ergonomics
Efficiency	Efficiency, efficient, productivity, output, expert, effective, capability, proficiency etc.	Energy
Installation/ Uninstallation	Install, uninstall, start, end, position, settle, plant, place, remove etc.	Assembly
Life	Life, duration, durability, durable, existence etc.	Quality Control
Manufacturability	Manufacturability, build ability, construct, fabricate, produce, create, make, weld ability etc.	Production
Portability	Portable, portability, mobile, mobility, movable, movability, adjustability, adaptability etc.	Portability
Price	Cost, price, expense, charge, fee, fare, sum, amount, estimate, expenditure etc.	Cost
Quietness	Quiet, noise, noisy, silent, loud etc.	Operation
Reliability		Quality Control
Reusability	Reliability, dependability, authenticity, genuine etc.	Recycle

Table 18-Continued: Attribute and Requirement Relation

Attribute Subclass	Keywords used	Related Requirement Type
Serviceability		Maintenance
Maintenance	Maintain, service, repair, replace etc.	
Cleaning	Clean, cleanse, scrub, rinse, disinfect, dry etc.	
Testability	Testability, test, measure etc.	Operation
User Friendliness	Use, ergonomic, simple, complex, automatic, manual, usability, simplicity etc.	Usability

After the attributes are placed under distinct subclasses, SWRL rules will be used to infer the requirement type. Relation of all subclasses of Tangible and Intangible Attribute is identified and SWRL rules are applied to get the inferred instances of *RequirementType* subclasses. [Table 18] shows the relation of different attribute classes with the requirement type classes. Attributes are generally noun e.g. "Height", "Weight", but Boolean attributes contain verb such as "*isEaseToUse*", "*isEasyToMaintain*", "*isLight*", "*isCommon*" etc. For this reason, along with nouns, few verbs and adjectives are also used as a keyword and the tool will match the substring of Boolean attributes with those keywords. [Equation 31, Equation 32, Equation 33, Equation 34,Equation 35] demonstrate the applied SWRL rules for Geometry, Cost and Assembly types. For example, the requirement sentence "The frame of the ingot oven is not taller than 51/2 feet" describes an attribute height, which is a subclass of size.

According to the SWRL rule illustrated in [Equation 31], if a nonfunctional requirement describes an attribute, which belongs to the class Size, the type of the requirement will be Geometry. Hence, the type of the requirement statement "The frame of the ingot oven is not taller than 51/2 feet" will be Geometry. Similar rules are used for all other subclasses and they are not presented in the paper for brevity of the paper. Few examples of reasoning requirement type by Attributes are shown in the [Table 19].

$$NonFunctionalRequirement(?x) \wedge Size(?y) \wedge describesAttribute(?x,?y) \rightarrow Geometry(?x) \quad \text{Equation 31}$$

$$Affordability(?y) \wedge NonFunctionalRequirement(?x) \wedge describesAttribute(?x,?y) -> Cost(?x) \quad \text{Equation 32}$$

$$Price(?y) \wedge NonFunctionalRequirement(?x) \wedge describesAttribute(?x,?y) -> Cost(?x) \quad \text{Equation 33}$$

$$Installation(?y) \wedge NonFunctionalRequirement(?x) \wedge describesAttribute(?x,?y) -> Assembly(?x) \quad \text{Equation 34}$$

$$AssemblyMeasure(?y) \wedge NonFunctionalRequirement(?x) \wedge describesAttribute(?x,?y) -> Assembly(?x) \quad \text{Equation 35}$$

Table 19:Example of Reasoning requirement type by attribute instances of requirement statement

Asserted Attribute	Similar Keyword	Attribute Subclass	Requirement Type
Requirement→ "The suspension weighs between 5-10 lbs"			
Weight	Weight	Weight	Geometry
Requirement→ "The suspension remains rigid during hard cornering"			
IsRigid	Rigid	Stiffness	Force
Requirement→ "The suspension can be easily accessed for maintenance"			
IsEasyToMaintain	Maintain	Maintenance Serviceability	Maintenance
Requirement→ "EW has high temperature response"			
Temperature Response	Temperature	Temperature	Energy
Requirement→ "Hand Truck balances safely and easily"			
SafetyOfBalance	Safe	Operation Safety –Safety Measure	Safety

4.6.3 Ontology Reasoning based on Verb:

Since, functional requirement do not have any attributes, reasoning through attributes is not possible for functional requirements. Rather, for functional requirements verbs are used to infer the type of the requirement. Classification of the Verb class is already explained in Chapter 2.Thus, the objective of this section is to illustrate the relation of the verbs and the requirement types. Unlike the attribute, the type of the verbs is user defined and only SWRL rules are required to reason the type of the requirement.

[Table 20] shows the connection of each subclass of verb with the subclasses of requirement type and the rules used for reasoning. For example, if a behavior of a functional requirement has a verb, which is a member of Transport class, the type of the corresponding functional requirement will be Transport. e.g. The requirement statement "The suspension traverse easily on slow difficult terrain" has a verb traverse, which is an instance of Transport class. The requirement type will be Transport Requirement. Similarly, another requirement "Doorjig can be simply attach to truck frame." has a verb attach, which is an instance of Link. According to the rules mentioned in [Table 20], if a requirement describes a behavior, which has a verb of class Link, the requirement will be Assembly type requirement. Thereby, the requirement "Doorjig can be simply attach to truck frame." will be an Assembly type requirement. Another requirement "Electric snow blower prevents snow blowback on operator" has a verb prevents of class prevent and according to the rules the type of the requirement will be Force. Likewise, type of all functional requirements can be determined by using the following rules.

Table 20: Connection of verb class with the requirement type and related SWRL rules

Verb Class	Requirement Type	SWRL rules
Branch		
Distribute	Operation	$Distribute(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Operation(?x)$
Separate		
Divide	Operation	$Divide(?z) \wedge FunctionalRequirement(?x) \wedge$

		$describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Operation(?x)$

Table 20-Continued: Connection of verb class with the requirement type and related SWRL rules

Verb Class	Requirement Type	SWRL rules
Extract	Operation	$Extract(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Operation(?x)$
Remove	Assembly	$Remove(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Assembly(?x)$
Channel		
Export	Operation	$Export(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Operation(?x)$
Guide		$Guide(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Kinematics(?x)$
Allow DOF	Kinematics	
Rotate	Kinematics	
Translate	Kinematics	
Import	Operation	$Import(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Operation(?x)$
Transfer		
Transmit	Operation	$Transmit(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Operation(?x)$
Transport	Transport	$Transport(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Transports(?x)$
Connect		
Couple		$Couple(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Assembly(?x)$
Join	Assembly	
Link	Assembly	
Mix	Operation	$Mix(?z) \wedge FunctionalRequirement(?x) \wedge$

		$describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Operation(?x)$

Table 20-Continued: Connection of verb class with the requirement type and related SWRL rules

Verb Class	Requirement Type	SWRL rules
Control Magnitude		
Actuate	Kinematics	$Actuate(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Kinematics(?x)$
Change		
Condition	Energy	$Condition(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Energy(?x)$
Increment	Operation	$Increment(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Operation(?x)$
Decrement	Operation	$Decrement \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Operation(?x)$
Shape	Ergonomics	$Shape(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Ergonomics(?x)$
Regulate		$Regulate(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Kinematics(?x)$
Decrease	Kinematic	
Increase	Kinematic	
Stop		$Stop(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Forces(?x)$
Inhibit	Force	
Prevent	Force	
Convert	Energy	$Convert(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Energy(?x)$
Provision		$Provision(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y)$
Store		

Collect	Operation	$\wedge hasVerb(?y, ?z) \rightarrow Operation(?x)$

Table 20-Continued: Connection of verb class with the requirement type and related SWRL rules

Verb Class	Requirement Type	SWRL rules
Contain	Operation	
Supply	Operation	
Signal		$Signal(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Signals(?x)$
Indicate	Signal	
Display	Signal	
Track	Signal	
Process	Signal	
Sense	Signal	
Detect	Signal	
Measure	Signal	
Support		
Position	Geometry	$Position(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Geometry(?x)$
Secure	Safety	$Secure(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Safety(?x)$
Stabilize	Quality Control	$Stabilize(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow QualityControl(?x)$

4.7 Conclusion:

The chapter describes ontological reasoning procedure for completeness and specificity measurement and also to classify requirements. The method described here

involves several steps to measure the completeness and specificity. A data property `hasCompletenessScore` with a range of 0 or 1 and requirement domain is introduced into the ontology. SWRL rules are applied to measure the completeness of each instance of requirement class. The reasoner analyzes the requirements and assigns completeness score of 0 or 1. Further, based on the requirement's score the reasoner also infers their class as either complete or incomplete. After measuring completeness, each complete requirement undergoes the specificity measurement process. A software tool is developed using JAVA and OWL API to measure the specificity and to classify requirements. Each time user runs the application; it calculates the specificity of requirements based on few criteria and assigns a specificity score in a scale of 0-1 and asserts specificity class of each requirement via ontology class assertion axioms. The tool also determines the type of all attribute instances. After attribute classification, the reasoner, depending on either attribute or verb, also infers the type of the requirements. The application is used for attribute classification and the application uses the concept of keyword matching to derive the type of each attribute instance. However, SWRL rules are used to reason the requirement class for both situations verb and attributes. Since, the rules used for requirement classification enforce partial constrains, it is possible to have requirements, which does not lie under any of the predefined category of requirements. In conclusion, the application and the reasoning technique together can accurately measure the completeness and specificity of requirements and also infers the appropriate type of the requirement statement.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

The objective of this research was to develop a form neutral model of engineering requirement representation and to use the model to measure the information content, analyze the quality indicators of requirements, and classify the requirements according to their types. To provide the answers for the research questions identified in Chapter 1, , different methods and metrics were used throughout this study. In this chapter, the findings related to these questions are summarized and the main contributions and the future works are discussed

5.1 Answers to Research Questions:

1. **What are the components of a formal ontology for requirements modeling?**

   As discussed in chapter 2, this research introduced ReqOn as an ontology for the representation of engineering design requirements. In ReqOn, requirements were classified into two distinct categories- functional and nonfunctional according to their purpose. The functional requirements describe the function that needs to be achieved by the product or any part of the product, while, the nonfunctional requirements explain the expected characteristics of the product or any part of the product.

   Further, functional and nonfunctional requirements were broken down to their

atomic elements.

A functional requirement is associated with one product and should exhibit a particular behavior. Behavior is not an atomic component and thereby, it is broken down into subject, verb, object and adjuncts. ReqOn also enforces cardinality restriction to the ontology modules. Behavior of the functional requirements must have exactly one primary subject, primary object, and one action verb. However, a behavior may have some secondary subjects, some secondary objects, and some adjuncts. In ReqOn, product and behavior classes are connected to the functional requirement class through object properties hasProduct, and describesBehavior respectively. Similarly, subject, object, verb, and adjuncts are connected to the behavior class through hasPrimarySubject, hasSecondarySubject, hasPrimaryObject, hasSecondaryObject, hasVerb, and hasAdjunct, respectively.

Nonfunctional requirements are also associated with a product and inherit all the properties of its parent class; Requirement. Nonfunctional requirements are composed of subject, attributes, and adjuncts. Similar to the functional requirements, nonfunctional requirements also have some necessary and some optional elements. Nonfunctional requirements have exactly one primary subject, at least one attribute, and some adjuncts. HasPrimarySubject, hasSecondarySubject, describesAttribute, and hasAdjunct are the connecting

object properties between nonfunctional requirement and subject, attribute, and adjunct, respectively. Further, attributes were classified into qualitative, quantitative, and boolean attributes. Qualitative attributes have exactly one qualitative value, boolean attributes have exactly one boolean value and quantitative attributes have at least one value or an upper limit or a lower limit or a range and a unit. Numerical Values are connected with the quantitative attributes with data properties hasValue, hasUpperValue ,and hasLowerValue. Only float is allowed as range for these data properties. Appropriate domain and range of the object and data properties were also identified in ReqOn.

2. **What is a good metric for measuring the information content of engineering requirements?**

   Chapter 3 demonstrates two different approaches for measuring the information content of requirements. Both approaches utilize Shannon's Information theory. One approach is based on the concept of uniform probability distribution, whereas, another approach is based on sequential selection.

   Uniform probability distribution approach assumes uniform probability distribution of the leaf classes on the ontology. In this approach, the entropy of each class is considered to be dependent on the structure of the class and the number of instances of the class that are available in the ontology at any given

timestamp. 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 total entropy of a class (Ec_i) is calculated as the summation of taxonomy (TEc_i) and size (SEc_i) entropy.

$$Ec_i = TEc_i + SEc_i$$

$$TEc_i = -log_2(Pc_i)$$

$$SEc_i = -log_2(\frac{1}{Nc_i})$$

Where, TEc_i is the Taxonomy Entropy of the i^{th} class c_i and Pc_i is the probability of occurrence of class c_i . SEc_i is the Size Entropy of the i^{th} class c_i and N_{ci} is the number of instances of class c_i.

In the sequential selection approach, it is assumed that a design decision is made through making a sequence of selections from a finite set of options. In this approach, the probability of occurrence of each class in the ontology depends on the number of direct and indirect instances of that class. Higher number of instances implies higher probability of occurrence. This approach involves traversing the tree from the top level to the bottom and calculating probability of occurrence of all classes along the path that leads to the final selection. According to this approach, total entropy of each element of the ontology will be the sum of the information contained in the class of the element and the information content of the decision associated in the selection of the element.

$$IC(Element) = IC(Class_c) + IC(D)$$

To validate these two approaches, information content of the requirement instances of ReqOn were measured using these methods individually. The results obtained were compared and it was concluded that the two methods provide almost similar values of IC. Therefore, any of these two approaches can be used to measure information content. Finally, the information content of a requirement statement can be calculated by using these two equations.

$$E_{\text{Functional Req}} = E_{\text{Behavior}} + E_{\text{Product}} + E_{\text{Subject}} + 1(\text{is Functional}) + 1(\text{isConstraint})$$

$$E_{\text{NonFunctional Req}} = E_{\text{Attribute}} + E_{\text{Product}} + E_{\text{Subject}} + E_{\text{Primary Subject}} + 1(\text{is Functional}) + 1(\text{isConstraint})$$

3. **How may the completeness and specificity of a requirement statement be measured using the formal requirement ontology?**

   Completeness analysis was conducted by checking the presence of the necessary elements of ReqOn. If a requirement contains all the required components, it was considered to be a complete requirement. If a functional requirement describes a behavior and the behavior contains a primary subject, a primary object and a verb, the requirement is considered as a complete requirement. Though, a nonfunctional requirement is considered to be a complete requirement if it contains a primary subject, describes at least one attribute and the attribute has a value.

The Specificity of complete requirements was measured quantitatively and a specificity score of 0-1 was assigned to each requirement statement of the ReqOn. For functional requirements existence of secondary subject, depth of verb, existence of secondary object, number of adjuncts, and existence of a measure adjunct were identified as main specificity enhancing factors. For nonfunctional requirement, existence of secondary subject, number of attributes, number of adjuncts, and type of Attributes were established as specificity criteria.

Each of these criteria was weighted in a 1-5 scale based on their impact on the specificity. The scores were also normalized and were represented in a 0-1 continuous scale.

After calculating the completeness and specificity score, ontology reasoning was used to determine and infer the specificity class. SWRL rules were also introduced in chapter 4 to execute the reasoning.

4. **How to use ontological reasoning techniques to classify requirements?**

Requirements pose restrictions on different aspects of a product such as the geometry, features, energy requirements, operation, manufacturing and assembly process, safety, maintenance, reliability, reusability, lifecycle, and overall quality of the product. Each of these categories is represented as a subclass of

Requirement Type in ReqOn. The type of the requirements was derived automatically using ontology reasoning method. Two types of reasoning technique were used in this research: reasoning based on *Attributes* for *Nonfunctional Requirement* and reasoning based on *Verb* for *Functional Requirement.* For Nonfunctional requirements attributes were classified into tangible and intangible attributes. Further these two attributes were subdivided into different categories and keyword-matching technique was used to assert the type of the attribute programmatically. A java tool was also developed to determine the type of the attribute based on keywords detection. Once the type of the attributes were assigned, SWRL rules were used to infer the type of the requirement based on the type of the attribute used. Since, functional requirement do not have any attributes, reasoning through attributes is not possible for functional requirements. Instead, for functional requirements, verb type was used to infer the type of the requirement. Unlike the attribute, the type of the verb is user defined and only SWRL rules are required to reason about the type of the requirement. Two examples of the rules that were used to determine the type of the requirement via ontology reasoning are presented below:

$$NonFunctionalRequirement(?x) \wedge Size(?y) \wedge describesAttribute(?x, ?y) \rightarrow Geometry(?x)$$

$$Guide(?z) \wedge FunctionalRequirement(?x) \wedge describesBehavior(?x, ?y) \wedge hasVerb(?y, ?z) \rightarrow Kineatics(?x)$$

5.2 Contribution:

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: 1) Developing the first comprehensive ontology for requirement representation based on OWL and 2) developing quantitative methods and metrics for requirement evaluation supported by automated ontological reasoning. The tools and methods developed in this work enable more intelligent decision making process in design. Also, they enable quantitative evaluation of the design process through monitoring information generation rate.

A java based automated tool was also built to translate the natural language requirement statement into OWL ontology. The tool is based on the linguistic structure of the requirement and it was developed in such a way that the user doesn't require any knowledge of owl ontology to use it. Further, necessary methods and metrics to measure the information content of a requirement statement were also established. A semi-automatic tool based on JAVA and OWL API was created to measure the information content of a single requirement statement or a whole requirement document for a product. 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 java tool was developed to measure and assert the completeness and specificity of a requirement. Also, requirements were classified into distinct categories using ontology reasoning.

It should be noted that the methodologies presented in this work is tailored for requirement statements that already follow a semi-structured syntax and grammar. For more unstructured texts and non-textual information, such as those found in technical standards or service guidelines, a more complete set of protocols should be developed. The metrics proposed in this work provide relative measures of the information content and should be treated as such. One explanation for the relative nature of the proposed measure for information content is that it varies with the size of the vocabulary captured. Therefore, the information content of the same requirement statement may change with time, depending on how many classes and instances exist in the ontology. For this reason, comparisons between two measure values of information content is meaningful only if they are calculated based on the same ontology.

5.3 Future Work:

There are multiple possibilities for extension of this work in the future. 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 automated information content measurement, it could be used for enabling knowledge management and reuse during 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 since it is based on the vocabulary of the Functional Basis (FB). But the non-functional 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.

APPENDIX SECTION

OWL API Guidelines

Installation and Getting Stated with OWL API

To configure your Java project download the owl api distribution jar file from http://sourceforge.net/projects/owlapi/. If you are using a Java IDE such as Eclipse, IntelliJ, or Netbenas then add all the owl api distribution jar files to your classpath.

Creating and Loading Ontology

To create an empty ontology or load an existing ontology from local file *OWLOntologyManager* should be created. The OWLOntologyManager provides a vital point for creating, loading, changing and saving ontologies. The instances of ontology are unique to a specific manager and the changes in ontology are incorporated through its manager. The following are some example of methods to create and load ontolology.
OwlOntologyManager creates a new (empty) ontology that has the specified ontology IRI (and no version IRI). It also requires an IRIMapper. The ontology document IRI of the created ontology will be set to the value returned by any installed OWLOntologyIRIMappers. If no mappers are installed or the ontology IRI was not mapped to a document IRI by any of the installed mappers, then the ontology document IRI will be set to the value of ontologyIRI.

Creating an Ontology

```
public static void createOntology () throws OWLOntologyCreationException {

    OWLOntologyManager manager =
OWLManager.createOWLOntologyManager();
    AutoIRIMapper mapper;
    mapper = new AutoIRIMapper (new File("myOntology"), true);
    manager.addIRIMapper(mapper);
    IRI myOntology_iri
=IRI.create("http://www.semanticweb.org/ontologies/ont.owl");
    OWLOntology ontology = manager.createOntology(myOntology_iri);
}
```

Loading an Ontology

```
public static void loadOntology(File file) throws OWLOntologyCreationException {

    OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
    OWLOntology  ontology = manager.loadOntologyFromOntologyDocument(file);

    }
```

OWLClass and OWLInstances

Retrieve All OWL Classes

```
Set<OWLClass> myClass = ontology.getClassesInSignature();
```

If you print the set then the output would be like :
<http://www.semanticweb.org/fa11/ontologies/2014/1/untitled-ontology-11#Part>
<http://www.semanticweb.org/fa11/ontologies/2014/1/untitled-ontology-11#ModificativeAdjunct>
<http://www.semanticweb.org/fa11/ontologies/2014/1/untitled-ontology-11#Flip>
<http://www.semanticweb.org/fa11/ontologies/2014/1/untitled-ontology-11#ConditionalAdjunct>
<http://www.semanticweb.org/fa11/ontologies/2014/1/untitled-ontology-11#Input>

To get the short form of classes, an instance of ShortFormProvider Class has to be declared and the method shortFormProvider.getShortForm(OWLEntity) will provide the short form of an owl entity in String format .

```
ShortFormProvider  shortFormProvider = new SimpleShortFormProvider();
Set<OWLClass> myClass = ontology.getClassesInSignature();
for (OWLClass  example : myClass){
                System.out.println(shortFormProvider.getShortForm(example));
        }
```

In this case the ouput would be like -
Part
ModificativeAdjunct
Flip
ConditionalAdjunct
Input

Retrieve All Instances:

```
Set< OWLNamedIndividual > myIndividuals = ontology.getIndividualsInSignature();
```

Parsing String into OWLClassExpression:

OWLDataFactory : OWLDataFactory is an interface and is bound to OWLManager.

To retrieve subclass, superclass or all instances of a user defined class; it's necessary to parse the user specified string into *OWLClassExpression*. To parse the string *OWLDataFactory*, *ManchesterOWLSyntaxEditorParser,* and *OWLEntityChecker* are needed. We also need a *BidirectionalShortFormProvider* to get the short form of all owl entities. The following code could be used for parsing a string into OWLClassExpression.

To provide an example a supporting java class MyMethods is been created and methods for parsing a String into OWLClassExpression is shown.

```
public class MyMethods {
   private static OWLReasoner reasoner;
   private static OWLOntology myOntology;
   private static BidirectionalShortFormProvider bidiShortFormProvider;
   private static ShortFormProvider shortFormProvider;

//create a constructor
public MyMethods (OWLReasoner reasoner, ShortFormProvider shortFormProvider) {
      myMethods.reasoner = reasoner;
      myMethods.myOntology = reasoner.getRootOntology();
      myMethods.shortFormProvider = shortFormProvider;

OWLOntologyManager manager = myOntology.getOWLOntologyManager();

// Gets the set of loaded ontologies that this ontology is related to (i.e. The set returned
includes all ontologies returned by the OWLOntology.getImports() method plus this
ontology.) If this ontology imports ontology B, and ontology B imports ontology C, then
this method will return the set consisting of this ontology, ontology B and ontology C.
 Set<OWLOntology> importsClosure = myOntology.getImportsClosure();
 bidiShortFormProvider = new BidirectionalShortFormProviderAdapter(manager,
importsClosure, shortFormProvider);

   }
```

```
public static OWLClassExpression  parseClassExpression(String classExpressionString)
throws ParserException {
        OWLOntologyManager manager = myOntology.getOWLOntologyManager();
        OWLDataFactory dataFactory = manager.getOWLDataFactory();
        ManchesterOWLSyntaxEditorParser myParser = new
ManchesterOWLSyntaxEditorParser(dataFactory, classExpressionString);
        myParser.setDefaultOntology(myOntology);
        OWLEntityChecker entityChecker = new
ShortFormEntityChecker(bidiShortFormProvider);
        myParser.setOWLEntityChecker(entityChecker);

        return myParser.parseClassExpression();
}
}
```

Retrieve SUPER/SUB/EQUIVALENT class or Instances of a class:

After parsing the string into *OWLClassExpression* we can get the sub class, super class, equivalent class, and instances of a class. Reasoning is a key part of working with OWL Ontologies and reasoners could be used to check the ontology consistency. OWLAPI has numerous interfaces to support the interaction with reasoners. But the main interface is the *OWLReasoner* that provides several methods to perform so many tasks. Likewise to get the subclass, super class or instances of a class we also need a reasoner.

Declare a reasoner –
OWLReasoner reasoner = new StructuralReasonerFactory().createReasoner(ontology);

SubClass : The following method will return the subclass of ClassName.

```
public static Set<OWLClass> getsubclasses (String ClassName , boolean direct){

   OWLClassExpression cls = parseClassExpression(ClassName);

   return  reasoner.getSubClasses(cls, direct).getFlattened();

   }
```

Note: For Boolean direct= true will return only direct subclasses of the String ClassName and for Boolean direct=false then it will return direct and indirect subclasses of ClassName.

Similarly, superclass and equivalent class of a class can be displayed.

For Super Class the return statement will be –
return reasoner.getSuperClasses(cls, direct).getFlattened();

For Equivalent Class the return statement will be –
return reasoner.getEquivalentClasses(cls).getEntities();

For Instances the above method could be used with a little change in the data type.

```
public static Set<OWLNamedIndividual> getInstances (String instanceName , boolean
direct){

   OWLClassExpression instance = parseClassExpression(instanceName);

   return reasoner.getInstances(instance, direct).getFlattened();

   }
```

OWL Properties:

OWLObjectProperties of ontology:

The method will return a set of all object properties of the ontology.

```
public static Set<OWLObjectProperty> getObjectProperties (OWLOntology
myOntology){

   return myOntology.getObjectPropertiesInSignature();

   }
```

Parsing a property expression (String) into an OWLObjectProperty: (Similar to parsing string into OWLClassExpression)

```
public static Set<OWLObjectPropertyExpression> parseObjectPropertyExpression
(String propertyExpressionString) throws ParserException {
      OWLDataFactory dataFactory =
myOntology.getOWLOntologyManager().getOWLDataFactory();
      ManchesterOWLSyntaxEditorParser myParser = new
ManchesterOWLSyntaxEditorParser(dataFactory, propertyExpressionString);
```

```
        myParser.setDefaultOntology(myOntology);
        OWLEntityChecker entityChecker = new
ShortFormEntityChecker(bidiShortFormProvider);
        myParser.setOWLEntityChecker(entityChecker);

        return myParser.parseObjectPropertyList();

    }
```

Range of Property: This code will return the range for each property in terms of Set<OWLClass> .

```
public static Set<OWLClass> getRange (String propertyExpressionString){
        Set<OWLClass> range = new HashSet<>();
        Set<OWLObjectPropertyExpression> property = parseObjectPropertyExpression
(propertyExpressionString);
        for (OWLObjectPropertyExpression p : property){
          range = reasoner.getObjectPropertyRanges(p, true).getFlattened();

        }
        return range;
    }
```

Domain of a Property: Domain of an object property is an owl class. The method will return the domain of object property.

```
public static Set<OWLClass> getDomain (String propertyExpressionString){
        Set<OWLClass> domain = new HashSet<>();
        Set<OWLObjectPropertyExpression> property = parseObjectPropertyExpression
(propertyExpressionString);
        for (OWLObjectPropertyExpression p : property){
         domain = reasoner.getObjectPropertyDomains(p, true).getFlattened();

        }
        return domain;
    }
```

Value of Property: Gets the object property values for the specified individual and object property expression.

```
public static Set<OWLNamedIndividual> getValueofProperty (OWLNamedIndividual
instance, String propertyExpressionString){
        Set<OWLNamedIndividual> value = new HashSet<>();
        Set<OWLObjectPropertyExpression> property = parseObjectPropertyExpression
(propertyExpressionString);
        for (OWLObjectPropertyExpression p : property){
         value = reasoner.getObjectPropertyValues(instance, p).getFlattened();

        }
        return value;
    }
```

OWLDataProperty

Here is an example of method, which could be used to print all the data properties in signature with the ontology in string format.

```
public static void printDataProperties (OWLOntology ontology, ShortFormProvider
shortFormProvider){

    Set<OWLDataProperty> dataProperty = ontology.getDataPropertiesInSignature();

        for (OWLDataProperty property : dataProperty){

        System.out.println(shortFormProvider.getShortForm(property));
    }

    }
```

The output of the method will be like

```
run:
hasLowerValue
hasBooleanValue
hasUpperValue
isFunctional
isConstrain
hasValue
```

OWL Data Property Values

Suppose we want the data property and their values associated with a particular instance. The following method could be used in this case. We have to provide the ontology and an

OWLNamedIndividual (instance). The method will return us a map containing the short form of data properties (String type) as key and property value of that data property for given instance as value.

```
public static Map<String, OWLLiteral> printDataProperties (OWLOntology ontology, OWLNamedIndividual instance){

    Set<OWLDataProperty> dataProperty = ontology.getDataPropertiesInSignature();
    Map<String, OWLLiteral> map = new TreeMap<>();
     for (OWLDataProperty property : dataProperty){
          Set<OWLLiteral>  literal  =  reasoner.getDataPropertyValues(instance, property);
          for (OWLLiteral value: literal){
             map.put(shortFormProvider.getShortForm(property), value);
                  }
    }
    return map;
    }
```

Method for Domain of Data Property

```
public static Set<OWLClass> getDomainofDataProperty (OWLDataProperty dataProperty){

      return reasoner.getDataPropertyDomains(dataProperty, true).getFlattened();
   }
```

REFERENCES

1. Ameri, F., & Summers, J. D. (2008). An ontology for representation of fixture design knowledge. *Computer-Aided Design and Applications, 5*(5), 601-611.
2. Ameri, F., Urbanovsky, C., & McArthur, C. (2012). A Systematic Approach to Developing Ontologies for Manufacturing Service Modeling. Paper presented at the *Proc. 7th International Conference on Formal Ontology in Information Systems (FOIS 2012), Graz, Austria,*
3. Carrillo de Gea, Juan M, Nicolás, J., Alemán, J. L. F., Toval, A., Ebert, C., & Vizcaíno, A. (2011). Requirements engineering tools. *Software, IEEE, 28*(4), 86-91.
4. 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.
5. Collopy, P. D., & Eames, D. J. (2001). Aerospace Manufacturing Cost Prediction from a Measure of Part Definition Information,
6. Darlington, M. J., & Culley, S. J. (2008). Investigating ontology development for engineering design support. *Advanced Engineering Informatics, 22*(1), 112-134.
7. Ducatel, G., Cui, Z., & Azvine, B. (2006). Hybrid ontology and keyword matching indexing system. Paper presented at the *Proc. of IntraWebs Workshop at WWW,*
8. El-Haik, B., & Yang, K. (1999). The components of complexity in engineering design. *IIE Transactions, 31*(10), 925-934.
9. Fabbrini, F., Fusani, M., Gnesi, S., & Lami, G. (2001). The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. Paper presented at the *Software Engineering Workshop, 2001. Proceedings. 26th Annual NASA Goddard,* 97-105.
10. Frey, D. D., Jahangir, E., & Engelhardt, F. (2000). Computing the information content of decoupled designs. *Research in Engineering Design, 12*(2), 90-102.
11. Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., & Schneider, L. (2002). Sweetening ontologies with DOLCE. *Knowledge engineering and knowledge management: Ontologies and the semantic Web* (pp. 166-181) Springer.
12. Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition, 5*(2), 199-220.
13. Halpin, T. (1996). Conceptual schema and relational database design Prentice-Hall, Inc.
14. Hauge, P. L., & Stauffer, L. A. (1993). ELK: A method for eliciting knowledge from customers. *ASME DES ENG DIV PUBL DE., ASME, NEW YORK, NY(USA), 1993, 53*, 73-81.

15. Hooks, I. (1994). Writing good requirements. Paper presented at the *INCOSE International Symposium, , 4*(1) 1247-1253.
16. Jiao, J. R., & Chen, C. (2006). Customer requirement management in product development: a review of research issues. *Concurrent Engineering, 14*(3), 173-185.
17. Joshi, S., & Summers, J. D. (2014a). Impact of Requirements Elicitation Activity on Idea Generation: A Designer Study. Paper presented at the *ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference,* V007T07A026-V007T07A026.
18. Joshi, S., & Summers, J. D. (2014b). Tracking Project Health Using Completeness and Specificity of Requirements: A Case Study. Paper presented at the *ASME 2014 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference,* V003T04A001-V003T04A001.
19. Kossmann, M., Wong, R., Odeh, M., & Gillies, A. (2008). Ontology-driven requirements engineering: building the OntorEM meta model. Paper presented at the *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference On,* 1-6.
20. Lamar, C. (2009). Linguistic analysis of natural language engineering requirements.
21. Lami, G. (2005). QuARS: A Tool for Analyzing Requirements,
22. Lefkoff-Hagius, R., & Mason, C. H. (1990). The role of tangible and intangible attributes in similarity and preference judgments. *Advances in Consumer Research, 17*(1), 135-143.
23. Lin, J., Fox, M. S., & Bilgic, T. (1996). A requirement ontology for engineering design. *Concurrent Engineering, 4*(3), 279-291.
24. Mir, M., Agarwal, N., & Iqbal, K.Applied ontology for Requirements Engineering: An approach to semantic integration of requirements model with system model. *International Conference on Software Engineering and Applications,* SEA. 214-221. doi:10.2316/P.2011.758-056
25. 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.
26. Motik, B., Sattler, U., & Studer, R. (2005). Query answering for OWL-DL with rules. *Web Semantics: Science, Services and Agents on the World Wide Web, 3*(1), 41-60.
27. Noy, N. F., & McGuinness, D. L. (2001). Ontology development 101: A guide to creating your first ontology.
28. Paul, G., & Beitz, W. (1984). Engineering design. *London, UK: Design Council,*
29. Qureshi, N. A., Jureta, I. J., & Perini, A. (2011). Requirements engineering for self-adaptive systems: Core ontology and problem statement. Paper presented at the *Advanced Information Systems Engineering,* 33-47.

30. Rolland, C., & Proix, C. (1992). A natural language approach for requirements engineering. Paper presented at the *Advanced Information Systems Engineering,* 257-277.
31. Sen, C., Caldwell, B. W., Summers, J. D., & Mocko, G. M. (2010a). Evaluation of the functional basis using an information theoretic approach. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 24*(01), 87-105.
32. Sen, C., Caldwell, B. W., Summers, J. D., & Mocko, G. M. (2010b). Evaluation of the functional basis using an information theoretic approach. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 24*(01), 87-105.
33. Sen, C., Summers, J. D., & Mocko, G. M. (2010). Topological information content and expressiveness of function models in mechanical design. *Journal of Computing and Information Science in Engineering, 10*(3), 031003.
34. Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review, 5*(1), 3-55.
35. Suh, N. P. (1990). *The principles of design*. New York ; Oxford: Oxford University Press.
36. Suh, N. P. (2001). *Axiomatic design: advances and applications*. New York ; Oxford: Oxford University Press.
37. Tseng, M. M., & Jiao, J. (1998a). Computer-aided requirement management for product definition: a methodology and implementation. *Concurrent Engineering, 6*(2), 145-160.
38. Tseng, M. M., & Jiao, J. (1998b). Computer-aided requirement management for product definition: a methodology and implementation. *Concurrent Engineering, 6*(2), 145-160.
39. Turk, W. (2006). Writing requirements for engineers [good requirement writing]. *Engineering Management, 16*(3), 20-23.
40. Ulrich, K. T. (2003). *Product design and development* Tata McGraw-Hill Education.
41. Uschold, M., & King, M. (1995). *Towards a methodology for building ontologies* Citeseer.
42. Wiegers, K. E. (1999). Writing quality requirements. *Software Development, 7*(5), 44-48.
43. Wilson, W. M., Rosenberg, L. H., & Hyatt, L. E. (1997). Automated analysis of requirement specifications. Paper presented at the *Proceedings of the 19th International Conference on Software Engineering,* 161-171.
44. Wilson, D. R., & Massachusetts Institute of Technology. Department of Mechanical Engineering. (1980). *An exploratory study of complexity in axiomatic design* Retrieved from http://hdl.handle.net/1721.1/15861

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