The Journal of Systems and Software xxx (2013) xxx–xxx

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Conceptual modeling of natural language functional requirements

Vidhu Bhala R. Vidya Sagar[a,*], S. Abirami[b]

[a] Mindtree Limited, India
[b] Department of Information Science and Technology, College of Engineering, Anna University, Guindy, Chennai 600025, India

ARTICLE INFO

Article history:
Received 10 March 2012
Received in revised form 27 July 2013
Accepted 11 August 2013
Available online xxx

Keywords:
Automated requirements analysis
Conceptual modeling
Natural language processing
Syntactic analysis

ABSTRACT

Requirements analysts consider a conceptual model to be an important artifact created during the requirements analysis phase of a software development life cycle (SDLC). A conceptual, or domain model is a visual model of the requirements domain in focus. Owing to its visual nature, the model serves as a platform for the deliberation of requirements by stakeholders and enables requirements analysts to further refine the functional requirements. Conceptual models may evolve into class diagrams during the design and execution phases of the software project. Even a partially automated conceptual model can save significant time during the requirements phase, by quickening the process of graphical communication and visualization.

This paper presents a system to create a conceptual model from functional specifications, written in natural language in an automated manner. Classes and relationships are automatically identified from the functional specifications. This identification is based on the analysis of the grammatical constructs of sentences, and on Object Oriented principles of design. Extended entity-relationship (EER) notations are incorporated into the class relationships. Optimizations are applied to the identified entities during a post-processing stage, and the final conceptual model is rendered.

The use of typed dependencies, combined with rules to derive class relationships offers a granular approach to the extraction of the design elements in the model. The paper illustrates the model creation process using a standard case study, and concludes with an evaluation of the usefulness of this approach for the requirements analysis. The analysis is conducted against both standard published models and conceptual models created by humans, for various evaluation parameters.

1. Introduction

The requirements gathering and analysis phase, or the requirements phase, in short, is the most critical phase in the software development life cycle (SDLC). During this phase, analysts collaborate with project stakeholders to gather the requirements for a project.

The requirements serve as inputs to further phases of the SDLC, in terms of work content, and for the planning of schedule and effort. They need to be as complete as possible. Changes to requirements during subsequent phases of the project can be more expensive than during the requirements phase. A major problem with functional requirements is that of unstated or implicit requirements, which stakeholders assume the analyst knows. Such issues in the requirements phase can cause disagreements among development teams and business teams much later, for instance, during acceptance testing, or after project launch, and can consume a lot of effort and time to correct.

To safeguard against changes made later by the requirements providers, most projects require a signoff of the requirements by all stakeholders, including customers, developers, business analysts and testers. Although this could be part of the process or a contractual requirement in the project, various issues exist that can reduce the signoff process to an 'in letter', rather than 'in spirit' activity.

The requirements analysis phase is highly dependent on personal opinions and is subjective in nature. Some stakeholders have trouble understanding the requirements presented in the form of text without the assistance of an analyst. It is important that the stakeholders are able to understand the requirements presented by the requirements analyst and comprehend the impact of the requirements in a uniform and efficient manner.

While many tools exist to render and visualize requirements models, not many tools exist to assist the requirements analyst during the requirements analysis phase. For the purpose of visualizing

Abbreviations: NLP, natural language processing; UML, unified modeling language; ER, entity relationship; EER, extended entity relationship; OOAD, object oriented analysis and design; SDLC, software development life cycle; OO, object oriented; NLTK, natural language tool kit; POS, part of speech.

* Corresponding author. Tel.: +91 442 266 3229.
E-mail addresses: vidhubhala.r.vidyasagar@gmail.com (V.B.R. Vidya Sagar), abirami_mr@yahoo.com (S. Abirami).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.08.036

the functional requirements, the conceptual or domain model is a valuable intermediate artifact. A conceptual model "*represents 'concepts' (entities) and relationships between ideas in a problem domain*" (CORDIS, 2013). The conceptual model, as defined by Larman (2002) contains only domain entities and attributes, and represents the static model of a system. On the other hand, the UML based OOAD model resembles a UML class diagram (Booch et al., 2010) through the inclusion of class operations and inter-class relationships, which also represent the dynamic nature of the system in study. A conceptual model is considered to be efficient in visual communication because it uses less space and fewer symbols to convey maximal information, as compared to natural language requirements. The use of conceptual models is widely recommended in newer software development models like Agile Modeling (Leffingwell, 2011), and has been adopted by objected-oriented development models using UML (Ambler, 2005).

A quality framework for conceptual model validation was first proposed by LindLand et al. (1994), who presented a framework for a conceptual model covering the syntactic, semantic and pragmatic qualities of the model. Syntactic quality addresses the correctness of the model through the use of a formal syntax. Semantic quality addresses the validity and the completeness of the conceptual model against the target domain, i.e., the domain from which the functional requirements are created. Pragmatic quality addresses the efficiency of the conceptual model in ensuring that the audience comprehends the information the model tries to convey. Each quality is elaborated with goals, and the recommended means to achieve the goals. The availability of a quality framework and the parameters to measure the quality of the conceptual model, make it a suitable choice for systematic requirements analysis.

This article focuses on the automated extraction of concepts and their relations to create a conceptual model. The constructed conceptual model is based only on stated, i.e., explicit requirements . Implicit requirements do not appear in the model, and must be provided in an explicit manner by the stakeholders during the requirements phase itself. The goal is to create a useful conceptual model that the analyst can use, to help the stakeholders understand the requirements before they sign off and conclude the requirements phase of the project.

By automating the creation of the conceptual model, the analyst can focus the attention of the stakeholders and herself/himself on the analysis and refinement of the models, rather than on the creation of the same. This allows the requirements to go through multiple iterations within the same timeframe, resulting in requirements that analysts and stakeholders are comfortable with.

The basis of this work lies in the linguistic aspects of the English language. Natural language sentences that constitute the functional specifications are parsed to understand their grammatical structures. Various design components such as classes, attributes, entities and relationships are then extracted from these parsed sentences, based on textual analysis.

Our work addresses the syntactic aspects of the conceptual model quality. As a logical extension, we discuss Pragmatic quality in Vidhu Bhala et al. (2012), which deals with the uniformity of audience perception, and is achieved through visualization techniques.

The structure of this paper is as follows. In Section 2, a survey of the related work in this field is offered. In Section 3, the architecture of the system is presented in the form of a visual layout of the functional blocks. The algorithms and functionality of each functional block and modules are detailed. Section 4 gives details of the implementation. A working example illustrates each of the functional blocks. In Section 5, an evaluation of the automated conceptual model is presented, and Section 6, gives the conclusion, along with the achievements of the current research work, the shortcomings and possible future work.

2. A survey of existing work

The trend in research in the area of automation of requirements analysis indicates, that although the need for automation was realized in the early 1990s (Kurt, 1995), the concept of the automation of this area has started gaining interest only recently, as is evident from the recent appearances of journal and conference articles in this area of research. A couple of reasons can be inferred for this renewed interest in this field. One is the recent surge in research and subsequent development in automatic text analysis, like parsers for natural language, tagged corpora for specific needs, anaphora resolvers etc., which have reduced the complexities of dealing with natural language. Another reason is the evolution of diagramming models like UML, and the ongoing attempts at standardizing the content of models. With the software industry converging to a few standard and well-known formats of requirements documentation, more focused research with wider application is possible.

Yue et al. (2011) refer to the conversion of requirements into models as a transformation process. They present an extensive survey of the transformation processes. They discovered that five types of pre-processing techniques, i.e., lexical, syntactic, semantic and pragmatic analyses, and categorization were found to be used in isolation or in combination. The resulting models from the transformation processes of the papers that were surveyed, were found to have low efficiency, and lacked evaluation mechanisms. Their survey included 20 works, categorized into 16 transformation approaches. They found that only 7 of the approaches were automated. Further, only 4 of the automated works could generate class diagrams, object diagrams or conceptual models, while the rest generated other types of analysis models like sequence diagrams or state charts. The survey also indicates underlying issues related to the evaluation of the completeness of rules used for transformation, due to the ambiguity of the English language. The four approaches that generated models similar to conceptual models were based on sentence pattern matching or shallow parsing. They required stringent rules on the pattern of the input sentences in the requirements text, for example, sentences of the form – sentence, verb, object, closely relating to the use case mode of writing requirements. Sanjay et al. (2010) explored the automatic creation of domain models from inputs that users provide in a spreadsheet. These inputs are processed as rules that represent individual events connecting entities in the requirements. A tool creates an intermediate representation of the requirements from the parsed rules, from which the domain model is then created. Ambriola and Gervasi (2006) present an extensive requirements modeling and analysis tool called CIRCE, that is based on fuzzy reasoning. A customized parser is used to parse sentences from the requirements text. User glossaries are provided manually to the tool. An expert system adds semantic information to the parsed information to create a final model. In both these approaches, the nature of the input text is constrained either through the use of templates, or through the use of a text that a parser can process.

Mu et al. (2009) parsed natural language requirements using typed dependencies, and extracted requirements from the functional requirements specifications. Their objective is to assess the non-functional requirements. While it might have been possible to use a free form input text, the authors require the input text to be in a standard format. They classify 10 types of semantic cases based on the sentence structure. The output is not a conceptual model, but a customized format for their specific needs.

Most of the publications discuss solutions that impose constraints on the language used for the requirements, with very few attempting to use natural language. Elbendak et al. (2011) represent the requirements model by semi-automatic processing of requirements presented in use cases, that are written in natural language. Their work, which motivated our research, attempts to analyze natural language requirements through shallow parsing. Manual intervention is required to aid the modeling process, with linguistic rules defined to guide the user in creating the model. The output is an extended entity relationship diagram. A limitation of their method is that, it is restricted to simple sentence structures, and it requires human interaction. However, their research presents an evaluation mechanism that we have used in our research as well.

As of writing this article, most research remains in the conceptual stage, as is evident from the pattern of published research work. While many conference papers have appeared in the past couple of years, only a handful of journal papers are seen. The trend clearly indicates that this is a very nascent research area that is yet to be established, and very few results are available.

The main contribution of this article is that, the methodology that we propose does not constrain the sentences in the input text, i.e., the requirements, to a strict pattern on structure. Another contribution in comparison with the surveyed work is that, the user interaction for the creation of the model is almost eliminated. Further, the automated conceptual model is evaluated using performance parameters, both against standard models and against models that humans are likely to create.

In line with the conceptual model quality framework, we present in this paper, a mechanism to generate a conceptual model automatically. We propose the usage of the Stanford Parser (2013) to parse the sentences from the requirements specifications text. The advantage of this Parser is that it can extract meaningful grammatical sentence constructs in the form of typed dependencies, on which we apply rules and generate appropriate constituents of the conceptual model, i.e., classes, attributes, operations and relations. Once the conceptual model is created to visualize the domain of the requirements, a visual transformation of the resulting model may be undertaken as described in Vidhu Bhala et al. (2012).

While the resulting conceptual model may not be in a format that can be used in further design phases of the project, because it lacks semantic information, it can certainly be used to fine-tune the requirements.

3. Implementation

The functional blocks involved in the automatic creation of the conceptual model are shown in the high-level functional block diagram shown in Fig. 1. The boxes shaded in gray on the left show the high level modules that represent the distinct and identifiable functional blocks that are executed in the sequence indicated by the direction of the arrows. The white boxes represent the individual modules within the functional blocks. The final output, i.e., the conceptual model is marked with a star.

The implementation was done in Python using NLTK (Bird et al., 2009). NLTK defines an infrastructure that can be used to build natural language programs in Python. It provides basic classes for representing data relevant to natural language processing, and standard interfaces for performing tasks to solve complex problems.

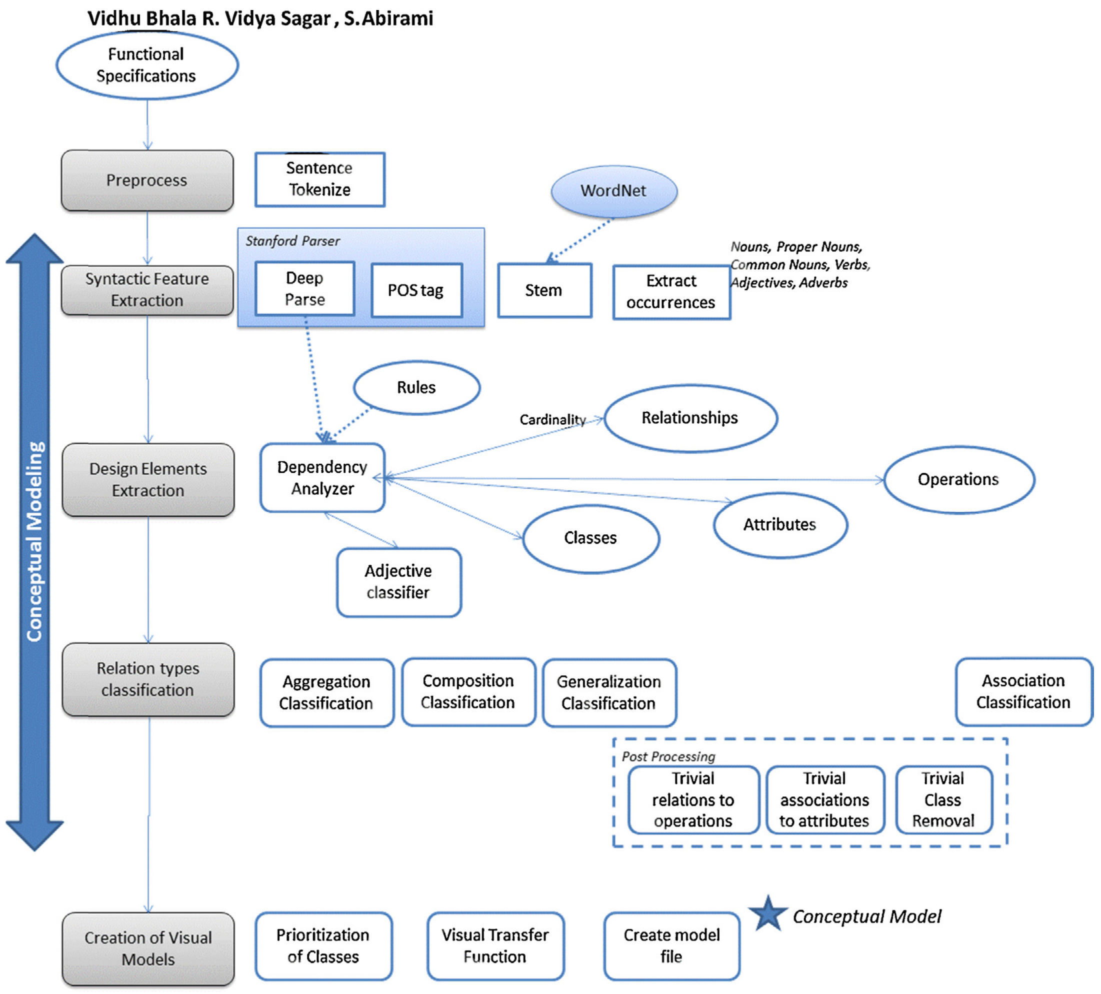

Vidhu Bhala R. Vidya Sagar , S.Abirami
Functional Specifications
Preprocess
Sentence Tokenize
WordNet
Stanford Parser
Syntactic Feature Extraction
Deep Parse
POS tag
Stem
Extract occurrences
Nouns, Proper Nouns, Common Nouns, Verbs, Adjectives, Adverbs
Conceptual Modeling
Rules
Cardinality
Relationships
Design Elements Extraction
Dependency Analyzer
Operations
Attributes
Classes
Adjective classifier
Relation types classification
Aggregation Classification
Composition Classification
Generalization Classification
Association Classification
Post Processing
Trivial relations to operations
Trivial associations to attributes
Trivial Class Removal
Conceptual Model
Creation of Visual Models
Prioritization of Classes
Visual Transfer Function
Create model file

Fig. 1. High level system diagram.

The Stanford Parser is a natural language Parser that works out the grammatical structure of sentences, i.e., which groups of words go together (as "phrases") and which words are subjects or objects of verbs. The Parser provides 'Typed Dependencies', otherwise known as grammatical relations. A typed dependency is of the form *dep_type (governor, dependent)*, where dep_type is the relation between the governor and the dependent word in a sentence. This representation provides a simple description of the grammatical relationships in a sentence. Along with the typed dependencies, the Parser can also return the part of speech of each word in a sentence, in the form of part-of-speech or POS tags.

3.1. Conceptual modeling

The base conceptual model is created as a result of the first four functional modules, i.e., text pre-processing, syntactic feature extraction, design elements extraction and relation type classification.

Although the objective of our research is to create models from natural language requirements, we assume that the sentences in the input text satisfy some basic requirements. Firstly, the sentences should be grammatically correct. Secondly, no negative statements are to be used in the input text. For example a sentence, '*The administrator cannot grant privileges to blocked accounts*' should not appear in the input text. Such statements do not contribute to conceptual models due to the nature of the model. Thirdly, each sentence is assumed to contain its own reference to any subject or object, i.e., no reference resolution is required on the input text. For example, a sentence, '*An administrator can send him an email*' should be written as '*An administrator can send the customer an email*'. This constraint includes all types of reference expressions: definite noun phrases, pronouns, demonstratives, one-anaphora, etc. (Jurafsky and Martin, 2000). Of the input constraints stated, with the exception of incorrect statements, it is possible to include the rest in future extensions without adding much complexity. This may be done by incorporating appropriate rules, reference resolution algorithms, and a semantic analysis corresponding to each of the other three constraints. The input text is assumed to be free from non-functional requirements, because conceptual modeling is meant for functional and not non-functional requirements.

3.1.1. Pre-processing

In typical natural language processing applications, functions like word tokenization, stemming, etc. are performed during pre-processing. However, due to the usage of the Stanford Parser, most of the typical pre-processing activities are eliminated. Preprocessing is limited to performing sentence tokenization to identify sentence boundaries.

3.1.2. Syntactic feature extraction

In syntactic feature extraction, the syntactic features of each sentence and the overall text are extracted. For each sentence, grammatical dependencies for each word in the sentence are retained in the form of typed dependencies, and the POS i.e., part-of-speech tags. The nouns are stemmed, i.e., converted to their base form. These results are combined to get a final set of words classified into their parts of speech. Each of these is examined further.

1. Dependency Parsing and POS tags
   The Parser (2013) is run against each sentence to generate typed dependencies. Typed dependencies give a simple representation of the grammatical relationships in a sentence that can be used effectively without too much linguistic experience or knowledge. It is an effective way to extract relations from sentences in natural language. For example, the sentence '*The employee's name is saved to the database by the program*' yields the dependencies *det(employee-2,The-1), poss(name-4,employee-2), nsubjpass(saved-6,name-4), auxpass(saved-6,is-5), det(database-9,the-8), prep_to(saved-6,database-9), det(program-12,the-11), agent(saved-6,program-12)*, where each grammatical dependency in the sentence is represented as a binary relation with a dependency name (abbreviated), a governor and a dependent word. For instance, nsubjpass (saved-6, name-4) means that 'name' is the subject of a passive sentence for the verb 'saved'. '*name*' occurs at position 4, while '*saved*' occurs at position 6 in the sentence, where position refers to the position of the word within the sentence. The Parser is capable of finding 52 types of grammatical relations that can be represented as typed dependencies (Parser, 2013). The Parser also returns the part-of-speech (POS) tags. The POS tags and the grammatical dependencies are retained for further processing.
2. Stemming
   Stemming is the process of reducing derived words to their base forms. For example, words like *creating*, and *created* are stemmed to *create*, and plural nouns are converted to singular forms. Stemming for words is done using the NLTK's WordNetlemmatizer module, which uses WordNet's inbuilt morphy function to find the inflected forms of the word (Morphy, 2013). WordNet is an online dictionary that includes many functions that semantically relate words, using functions. This is preferred over other morphological stemming algorithms like Porter or Lancaster, because the output of the lemmatizer should be a word that may be used as a class, which is not guaranteed in standard morphological stemmers (Jurafsky and Martin, 2000).
3. Extraction of grammatical occurrences
   Using POS tags, words are classified into nouns, verbs, adjectives or adverbs. Nouns are classified into proper and common nouns. Each word in the text is categorized into a single part of speech. Table 1 shows the mappings used for categorization.

The typed dependencies, stemmed words and the word categories are then used to further extract design elements that will participate in the design of the conceptual model.

3.1.3. Design elements extraction

The extraction of design elements that form the conceptual model is the core of the functional module. Here, we create the conceptual model from the building blocks of classes and relations. The syntactic information is used to determine the valid constituents of the conceptual model.

At this stage, typed dependencies are available for each sentence. We examine the typed dependencies for each sentence

Table 1
POS tag prefixes against Penn Treebank POS categories.

Prefix	Category	Actual types	
JJ	Adjective	JJ	Adjective
		JJR	Adjective, comparative
		JJS	Adjective, superlative
RB	Adverb	RB	Adverb
		RBR	Adverb, comparative
		RBS	Adverb, superlative
VB	Verb	VB	Verb, base form
		VBD	Verb, past tense
		VBG	Verb, gerund or present participle
		VBN	Verb, past participle
		VBP	Verb, non-3rd person singular present
		VBZ	Verb, 3rd person singular present
NN	Noun	NN	Noun, singular or mass
		NNS	Noun, plural
		NP	Proper noun, singular
		NPS	Proper noun, plural
Ignored		Others. . .	

against a set of rules. In these rules, we explore the grammatical relationships among the constituents of the sentence, and decide whether to create classes, attributes, objects or relationships.

While some rules are from established linguistic patterns from reference books, others are created afresh to take advantage of the deep parsing method that is used to extract grammatical structures. The rules are grouped into ***design rules***. The identification of classes, attributes, operations and relations is as a result of these rules. While design rules are generic statements about what constitutes design elements, implementation rules examine the typed dependencies to achieve the goals of the design rules. More than one ***implementation rule*** could map to a design rule. The purpose of layering the rules as design and implementation rules in this manner is to organize the rules better, with the design rules being the generic guidelines, and the implementation rules mapping into the detailed implementation specifics. For example, while a design rule could state that the subject of a sentence could be a class name, the implementation rules explore various types of subjects, like the subject of a passive sentence, active sentence, prepositional subject, etc.

Design rules are categorized, based on their syntactic contribution to the model, into subject-object transformation rules, and rules for classes, relations, operations and attributes. Each of the design rule categories is examined further.

3.1.3.1. Subject-object transformation rules. Some common rules, transformations and functions are applied along with other design rules. We call these subject-object transformation rules. There are two main categories of these rules, (1) change of subject and (2) creation of a compound subject using noun nominal or special adjectives.

***Subject Object Transformation Rule 1*:** *When a possession modifier type of grammatical relation occurs in a sentence, the subject is changed to refer to the owner of the possession.*

For example, "*The employee's name is saved to the database by the program*" yields the typed dependencies *det(employee-2,The-1)*, *poss(name-4,employee-2)*, *nsubjpass(saved-6,name-4)*, *auxpass(saved-6,is-5)*, *det(database-9,the-8)*, *prep_to(saved-6,database-9)*, *det(program-12,the-11)*, and *agent(saved-6,program-12)*. The identified conceptual class in this case becomes employee, and not name, although the subject of the sentence is name due to the occurrence of the poss typed dependency that denotes a possession modifier.

Although this rule does not make much sense at the linguistic level, it is appropriate and specific for conceptual modeling. This is because the object is likely to be an attribute or a constituent part of the main subject.

Subject Object Transformation Rule 2: *Noun compound modifiers are combined with the noun to generate compound words.*

This rule creates a compound noun from a typed dependency relation. For example, the sentence, '*An ATM accepts cash cards*' results in the dependencies *det(ATM-2,An-1), nsubj(accepts-3, ATM-2), nn(cards-5,cash-4), and dobj(accepts-3,cards-5)*. Although the object is 'cards', as specified by the dobj relation, the presence of nn, the noun compound modifier relation indicates that *cash card* is a compound noun, and must be the object in this relation.

Subject Object Transformation Rule 3: *An adjective that qualifies a noun, where the adjective cannot be classified combines with the noun subject to generate compound words.*

This rule decides whether a certain adjective qualifies the underlying noun as an attribute, in which case we call it classifiable, or whether it adds a meaning or categorization of the noun in a manner that the domain model demands, in which case, we call this an unclassifiable adjective. Depending on whether an adjective is classifiable or not, it can be an attribute of a class or the name of the class. For example:

save button	*- Becomes a domain conceptual class*
red button	*- Button has an attribute red, which is categorized into color as an attribute*

An adjective is a word that describes, identifies, modifies or quantifies something. In sentences, adjectives may be used in the predicative form (*The book was boring*), attributive form (*It was a boring book*), or appositive form (*The book, old and torn, was lying on the floor*) (Loberger and Shoup, 2009). Adjectives may be classified into categories such as quantity, shape, size, color etc. based on various criteria. In English, adjectives are examined for a category fit in a certain order, through an ordering process (Col, 2013). The first category of an adjective in an ordered list of categories determines the most likely category of that adjective.

The order of adjective categories is implemented using a dictionary data structure that included a corpus of known classifiable adjectives. When queried with a candidate adjective, the dictionary returns a classification type, in the order of the adjective categories, if it exists. If no classification is returned, we conclude that the adjective is unclassified. For instance, in the previous example, *red* would return a class '*colour*', while *save* would not return a class.

3.1.3.2. Identifying classes/entities. The identification of relevant conceptual classes is important for the creation of the Conceptual Model. Because the inherent assumption that our research makes is that the requirements text is potentially incomplete, the motivation behind the construction of the final conceptual model will be to visualize which concepts are explained in detail in the text. Based on this objective, a key change is made to the fundamental process of textual analysis to determine conceptual classes. The ***basic textual analysis*** (Elbendak et al., 2011) rules state that

All nouns are candidate classes
All verbs are either candidate operations or candidate relations

If all nouns get added to the list of classes, many unnecessary classes appear. These are usually irrelevant and may add to user confusion and visual clutter in the conceptual model. So, we altered *this basic rule* to state that '*A noun that appears as a subject is always a class, but a noun that appears as an object may not be a class*'. This rule is elaborated below.

Class Rule 1: Any Noun that appears as a subject is always a class

Nouns that appear as subjects are found in several types of sentence constructs, as can be seen from the indicative list in Table 2. Each of the rules in the table is implemented as an implementation rule.

Class Rule 2: Nouns occurring as objects participate in relations, but are not created as classes explicitly

This rule ensures that if an entity is described at least once directly, or indirectly, it will become a class. However, entities that always occur only as objects of discussion, may not appear as classes. In that situation, they will not participate in relations, but will appear as class operations led by the governing verb (action)

Table 2
Nouns as subjects appearing as classes.

Noun construct	Example	Identified class
Simple subject	A *bank* owns an ATM.	Bank
Subject of a sentence in passive form	An *ATM* is owned by a bank.	ATM
Subject of a prepositional clause that contains 'of'	The customer uses the ATM of the *bank*.	Bank
Agent of a sentence in passive form	An ATM is owned by a *bank*.	Bank

under the subject 'class', performing the action. An exception to this rule is when the object is an agent in a passive sentence, because the agent is responsible for performing the action.

In the sentence '*A bank owns an ATM*', class rule 1 ensures that *bank* is created as a class, while class rule 2 creates *ATM* as a potential class for the relation *own*. If ATM is described further in the text, then *ATM* will become a class using class rule 1; otherwise it will remain a candidate class, and will not occur as a separate class in the conceptual model.

Some standard class rules that have been in existence and have been well accepted in previous research works in this area are:

Class Rule 3: Gerunds are created as classes (Elbendak et al., 2011; Burg, 1997)

The Gerund forms of verbs are treated as class names.

e.g., Borrowing is processed by the staff → Borrowing is a class.

Class Rule 4: Nouns are always converted to their singular form Elbendak et al. (2011)

This is achieved by using the stemmed form of the word from the syntactic feature extraction phase. Because the word is a noun, stemming does not alter the basic morphology of the word.

3.1.3.3. Identifying attributes of a class. Attributes of classes can be broadly classified into two categories, i.e., nouns as attributes, and adjectives as attributes. When nouns are used as attributes of a class (e.g., A book has an author), further information is required to decide whether it must really be an attribute or stand-alone, in the context of the domain. Adjectives, when used as attributes must convey a certain property like color, size, shape, etc.

A key contribution of our work is the differentiation of adjectives that combine to form the class name, and adjectives that become attributes. We introduce a modified attribute rule for adjectives.

Attribute Rule 1: A classifiable adjective, either in the predicate or attributive form signifies an attribute

This rule uses the result of the adjective classifier. If the adjective classifier returns a category for the adjective, the category is used as the attribute of the subject that the adjective qualifies.

e.g., the red button glows when pressed. → Button has an attribute, color

Other rules that are used determine attributes of conceptual classes adopted from previous research works are:

Attribute Rule 2: A noun phrase which follows the phrase 'identified by', 'recognized by' indicates the presence of an attribute (Elbendak et al., 2011)

e.g., An employee is identified by the employee id. → Employee has attribute employee id

Attribute Rule 3: An intransitive verb with an adverb may signify an attribute (Elbendak et al., 2011)

e.g., The train arrives in the morning at 8 AM. → Train has an attribute, time

Attribute Rule 4: A possessive apostrophe signifies an attribute (Elbendak et al., 2011)

e.g., The employee's name is saved → Name is an attribute of the employee

Attribute Rule 5: The genitive case when referring to a relationship of the possessor using the 'of' construction signifies an attribute (Elbendak et al., 2011)

e.g., The name of the employee is printed. → Name is an attribute of the employee

These rules are suitable for conceptual modeling in most cases. However, there are some situations in which these can become inappropriate (Larman, 2002); for example, in the sentence phrases *employee's name*, and *employee's briefcase*, both name and briefcase will become attributes of the employee class. There are differing opinions by authors of object-oriented modeling techniques, on how to treat such situations. However, because our need is to model the underlying explicit information, it is not possible to determine if a given word should appear as an attribute or as a general association without further semantic information.

The approach adopted in our research is to model such attributes as relations, which are then converted to attributes during post-processing, if there are no further references, justifying that the ownership is contained within the class. This defers the decision as to whether a given word is an attribute until we are sure that it is not important enough to be modeled as a class. Further research into the semantics of the attribute can determine if the referred word will become an attribute or entity associated with the subject (Larman, 2002).

3.1.3.4. Identifying relation types. Once the conceptual classes, candidate conceptual classes (i.e., nouns as objects) and attributes have been identified, they are combined to form relations. A relation is represented as **R = (class a, card (class a), class b, card (class b), relation name)**, where *class a, and class b* are the participating classes in a binary relationship, and *card (class #)* denotes the multiplicity with which a class participates. By virtue of the earlier rules, *class a* is already denoted as a class, while *class b* may be a candidate class which does not exist as a class. The following rules identify the relations:

Relation rule 1: A transitive verb is a candidate for a relationship type (Elbendak et al., 2011)

A transitive verb links a subject and an object. If the object translates to a class (as per Class Rule 3), the verb becomes a relation.

e.g., *The banker issues a cheque → Relation (banker, 1, cheque, 1, issues)*

Relation rule 2: A verb with a preposition is a candidate for a relationship

A verb that contains a prepositional object linked to a transitive verb along with a preposition combines with the preposition to form a relationship.

e.g., *The cheque is sent to the bank.* → Relation (cheque, 1, bank, 1, sent_to)

Relation rule 3: A sentence of the form 'the r of a is b' (Elbendak et al., 2011)

Relation rule 4: A sentence of the form 'a is the r of b'

e.g., The bank of the customer is SBI → Relation (bank, 1, customer, 1, SBI)

e.g., SBI is the bank of the customer → Relation (bank, 1, customer, 1, SBI)

In the above sentences, r is the relation that relates class a to class b. The above two rules are examples of the relation being in the form of a noun, rather than a verb.

In the design elements extraction phase, associations, compositions, and inheritance are all extracted in a common format, and the classification is done at a later stage.

3.1.3.5. Identifying operations. Operations or class behaviors are used as dynamic aspects of a model. Although this is not the main focus of our research, operations are picked up anyway as a result of sentence structure processing.

Operation rule 1: An intransitive verb is an operation (Elbendak et al., 2011)

An intransitive verb is a verb that has no direct object.

e.g., The laptop hibernates. → hibernate is an operation of the laptop.

Operation rule 2: A verb that relates an entity to a candidate class that is not created as an entity (fails Class Rule 2 after completion of class identification) becomes an operation

Operation rule 2, as above is implemented by creating all subject, verb, object combinations as generic class relations during the

Table 3
Design rule categories and statements.

Rule category	S. no.	Design rule statement
Subject object transformation	1	When a possession modifier grammatical relation occurs in a sentence, the subject is changed to refer to the owner of the possession.
	2	noun compound modifiers are combined with the noun to generate compound words
	3	An adjective that qualifies a noun, where the adjective cannot be classified combines with the noun subject to generate compound words.
Class rule	1	Any Noun that appears as a subject is always a class
	2	Nouns occurring as objects participate in relations, but are not created as classes explicitly
	3	Nouns are always converted to their singular form
	4	Gerunds are created as classes
Attribute rules	1	A classifiable adjective, either in the predicate or attributive form signifies an attribute
	2	An intransitive verb with an adverb may signify an attribute
	3	A noun phrase which follows the phrase 'identified by', 'recognized by' indicates the presence of an attribute
	4	possessive apostrophe signifies an attribute
	5	The genitive case when referring to a relationship of the possessor using the 'of' construction
	6	Use of has, have in sentences denotes attributes
Relation rules	1	A transitive verb is a candidate for a relationship type
	2	A verb with a preposition is a candidate for a relationship
	3	A sentence of the form 'the r of a is b'
	4	A sentence of the form 'a is the r of b'
Operation rules	1	An intransitive verb is an operation
	2	A verb that relates an entity to a candidate class that is not created as an entity (fails Class Rule 2 after completion of class identification) becomes an operation

design elements identification stage. When the objects become trivial, i.e., are not referenced as a subject elsewhere, the class becomes a trivial class and the relation decomposes to a trivial relationship, which is modeled as a class operation. For instance, in the sentence, *an ATM accepts cash cards*, according to class rules 1 and 2, *ATM* and *cash card* become classes and candidate classes respectively. If the entity *cash card* were to be further described, e.g., *A cash card is made of plastic*, then the cash card becomes a class; otherwise, the *ATM* class will contain the operation accepts_cashcard().

3.1.3.6. Sample implementation rule and results. Design rules are categorized as classes, attributes, operations, relations and subject-object transformation rules. Table 3 shows the design rule categories and rule statements. Each design rule may be implemented through one or more implementation rules; i.e., specific implementations of a generic class of design rules.

The application of the implementation rule for a given design rule is illustrated in Fig. 2.

The design rule '*The r of a is b*' is a relation rule, expressed as an implementation rule

$$\forall n(subj(n,\alpha) \curlywedge prep_{of}(a,b) \curlywedge cop(n,v) \curlywedge n \in setCommonNouns \\ \curlywedge v \in setVerbs \vdash class(a), class(b), relation(a,b,n)) \quad (1)$$

The sentence '*The father of Tom is Jerry*', yields the dependencies det(father-2,The-1), nsubj(Tom-6,father-2), prep_of(father-2,Jerry-4), cop(Tom-6,is-5). The results of this are Class(Tom), Class(Jerry), relation(Tom, Jerry, father).

Similarly, Class Rule 1, states 'Any Noun that appears as a subject is always a class' and is expressed as an implementation rule:

$$\forall n(subjpass(n,\alpha) \curlywedge agent(n,v) \curlywedge n \in setCommonNouns \\ \vdash class(b), class(a), relation(a,b,n+'by')) \quad (2)$$

The sentence 'The evaluation is conducted by a technician', yields the dependencies det(evaluation-2,The-1), nsubjpass(conducted-4,evaluation-2), auxpass(conducted-4,is-3), det(technician-7,a-6), agent(conducted-4,technician-7). The results are Class(evaluation), Class(technician), Relation (evaluation, conducted_by, technician).

3.1.4. Relation types classification

The relationships that have been created after, the extraction of design features, are further classified as Associations, Aggregation, Composition and Inheritance or Generalization.

3.1.4.1. Aggregation classification. Aggregation can be found by considering clause patterns like 'is made up of', 'is part of', 'contains', 'consists', 'comprises', etc.

3.1.4.2. Composition classification. Composition is a special type of aggregation where there is a strong relationship between the whole and the part. Due to lack of sufficient information in the specification text, it may not always be possible to differentiate between composition and aggregation. Larman recommends that when deciding, if you need to use composition over aggregation, '*if in doubt, leave it out*' (Larman, 2002). The implementation correctly identifies one unambiguous type of sentence construct for composition, where a mass noun is used, for example, a consortium of banks, or a fleet of ships.

3.1.4.3. Generalization classification. Identification of generalization relations is based on the presence of a copula verb, which relates two nouns, (proper or common nouns) in a sentence. Generalization is also based on the detection of preposition types, which contain words like type of, categories of, kinds of, etc.

3.1.4.4. Post processing relations. After identifying aggregation, composition and generalization, the remaining relations can be categorized as associations. However, we recommend a post-processing phase to eliminate a few more relations, by creating attributes and operations before the final categorization.

Post processing of relations refers to actions that are only possible after all the relations and classes have been firmed up. The main intention of this phase is to remove and compress unnecessary detail; i.e., to ***coarse-grain*** the information that is extracted. This coarse-graining is based only on the implicit information in

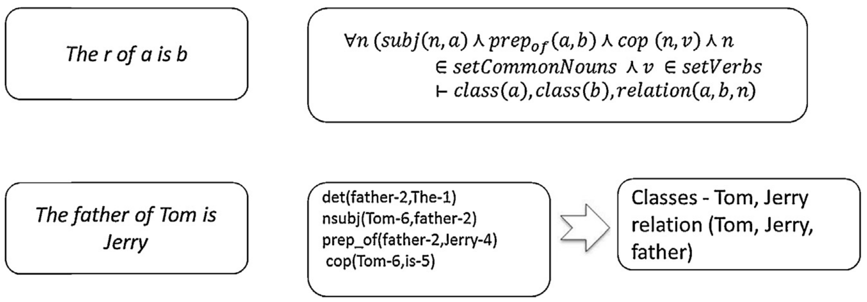

The r of a is b
$\forall n\ (subj(n,a) \wedge prep_{of}(a,b) \wedge cop\ (n,v) \wedge n \in setCommonNouns \wedge v \in setVerbs \vdash class(a), class(b), relation(a,b,n)$
The father of Tom is Jerry
det(father-2,The-1)
nsubj(Tom-6,father-2)
prep_of(father-2,Jerry-4)
cop(Tom-6,is-5)
Classes - Tom, Jerry
relation (Tom, Jerry, father)

Fig. 2. Illustration of an implementation rule under a design rule.

the text of the requirements, and not on human intelligence. We propose three types of Post Processing actions:

3.1.4.4.1. Trivial relations to operations. Relationships that do not refer to an existing class are created as class operations of the governing class. Due to Class Rule 1, the governing class will always exist.

$$\forall r(e \in R \wedge \exists \quad class(a) \wedge \quad class(b) \vdash ClassOpn(c, re\ln)) \tag{3}$$

where R is the set of relations $R = \{\langle class \quad a, card(class \quad a), class \quad b, card(class \quad b) re\ln\rangle\}$.

3.1.4.4.2. Trivial association to attributes. Relationships that are named ‘has’ or ‘have’ where there are no further references to the referred entity are moved into the governing class as attributes. To enable this post-processing, during attribute identification, all attributes found through the attribute rules are added to relations using the ‘has’/’have’ relation, rather than as direct attributes as illustrated in Table 4.

3.1.4.4.3. Trivial class removal. In some cases functional specifications include non-functional specifications or actor actions. Human intelligence can easily remove such classes from consideration. However, the implementation does not consider the semantics of each word. To deal with this problem, we define trivial classes, and remove all relations that contain this trivial class. Trivial classes could include words like ‘system’, ‘software’, ‘interface’, etc. This list, however, cannot be considered as globally applicable, because the domain and scope of each requirement defines what is trivial and what is not, and syntactic or semantic identification cannot replace human thinking.

3.1.4.5. Association classification. A relationship, that cannot be categorized into aggregation, composition or generalization, or pre-processed to trivial relations, is an association relation.

4. Case study

This section gives an implementation walkthrough of the conceptual model creation process with intermediate results. The input requirements specification is taken from the ATM problem statement (Rumbaugh et al., 1991). The passage is first manually modified to remove types of references, pronouns and wh-pronouns (who, whom, whose, whichever, whatever, etc.). It is also modified to ensure consistent usage of a word for a stated sense, i.e., one sense per discourse.

Modified ATM problem statement from Rumbaugh’s ATM model (Rumbaugh et al., 1991).

The system must support a computerized banking network that includes both human cashiers and ATMs. The computerized banking network will be shared by a consortium of banks. Each bank provides a computer that maintains the bank's accounts and processes transactions against the accounts. Cashier stations are owned by individual banks and communicate directly with the bank's computers. Human cashiers enter the account data and transaction data. An ATM communicates with a central computer. The central computer clears transactions with the banks. An ATM accepts a cash card and interacts with the user. An ATM communicates with the central computer to carry out transactions. An ATM dispenses cash, and prints receipts. The system requires appropriate record-keeping and security provisions. The system must handle concurrent access to the same account correctly. The banks will provide the bank's own software for the bank's own computers.

Table 4
Trivial relationship moved to attribute.

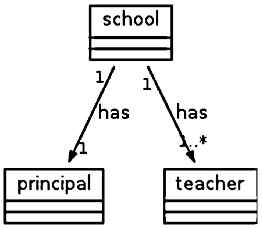
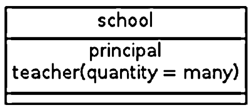
Statement	Relationships after extracting design features	Trivial relationship processed
A school has a principal and many teachers	 school 1 has 1 principal 1 has 1..* teacher	

Table 5
Trace plan of implementation rules for sample sentence.

Typed dependencies	Rules checked	Rule results and further actions	Trace of low level actions in rule results processing.
det ATM An	Ignored		
nsubj communicates ATM	Check **'Basic Rule'**	obj does not exist for same verb Fails	
	Check **'Intransitive Verb Rule'**	Fails, due to presence of prepositional object	
det computer a	Ignored.		
amod computer central	Ignored.		
prep_with communicates computer	Check **'Prepositional subject and object'**	Pass	
		class(ATM)	Stem (atm) = atm No compound nouns No adjective modifiers
		Create atm class	
		class(computer)	
		class(central_computer)	Stem (computer) = computer No compound nouns Adjective Modifier found No classification for 'central' Append to class name computer->central_computer
		Not created, as this is an object	
		Relation (ATM, central_computer, communicates_with)	
			cardinality for atm = 1 cardinality for computer = 1
		Create Relation (ATM,1, central_computer,1, communicates_with)	

4.1. Conceptual modeling

In the preprocessing phase, the passage is broken down into sentences. The result of the syntactic feature extraction gives the list of adjectives, adverbs, verbs, proper and common nouns. The final list that categorizes the words into parts of speech in the passage is:

Adjectives	'computerized', 'appropriate', 'individual', 'central', 'human', 'concurrent', 'own', 'same'
Adverbs	'directly', 'correctly'
Nouns	'record-keeping', 'receipt', 'cashier', 'consortium', 'computer', 'user', 'data', 'card', 'account', 'transaction', 'network', 'provision', 'atm', 'system', 'access', 'banking', 'station', 'bank', 'cash', 'security', 'software'
Proper Nouns	None
Common Nouns	'record-keeping', 'receipt', 'cashier', 'consortium', 'computer', 'user', 'data', 'card', 'account', 'transaction', 'network', 'provision', 'atm', 'system', 'access', 'banking', 'station', 'bank', 'cash', 'security', 'software'

In the final list, *ATM* has assumed a common noun form. A noun that occurs as a proper noun will assume a common noun form, if used in that manner even once. In this case, the usage of '*ATM*' in the first sentence '*The system must support a computerized banking network that includes both human cashiers and ATMs*' – has caused ATM to assume a common noun format.

Next, the design elements are extracted. The typed dependencies of each sentence are passed through each implementation rule. The category of the resulting implementation rule determines what classes, attributes and relations are obtained for a phrase. For instance, for the sentence, ***An ATM communicates with a central computer***, the sequence used to check for rules for this statement is shown in Table 5.

Each sentence is processed by different rules depending on the sentence structure, presence of conjunctions, prepositional subjects, gerunds, etc. The final results of the design elements extraction for the given text, after all the rules have been executed, are:

Classes:
set(['atm', 'cashier', 'system', 'consortium', 'computerized_banking_network', 'computer', 'central_computer', 'individual_bank', 'bank', 'cashier_station'])

Class relations:
The list of relations and relation names generated are shown in Table 6. The specific rule that was used to extract the design elements is mentioned in the last column. The advantage of the use of dependency parsing is evident from the fact, that the sentence is broken down into fragments, and the implementation rules are based on fragmented sentence structures, rather than on whole sentences.

The generic relationships are then classified according to their type, and trivial classes, and relationships are processed to eliminate them by moving them to attributes and operations respectively.

Table 6
Sample implementation – relations after design element extraction.

S. no.	Sentence No.	Class 1	Class 2 (may not exist)	Name of relation	Rule used
1.	0	computerized_ banking_network	atm	Includes	• Compound noun generator • Elementary subject object rule
2.	0	system	computerized_ banking_network	support	• Compound noun generator • Elementary subject object rule
3.	0	computerized_ banking_network	Cashier	Includes	• Compound noun generator • Elementary subject object rule • Adjective classifier–human denotes quality • cashier added as class, with attribute quality = human
4.	1	computerized_ banking_network	consortium	shared_by	• Compound noun generator • Passive subject with agent
5.	1	consortium	bank	of	• Secondary 'of' preposition following agent in passive subject (composition)
6.	2	bank	computer	provides	• Elementary subject object rule
7.	2	bank	account	Has	• Possession modifier for bank, converted to association likely to become attribute
8.	2	computer	bank	maintains_ accounts	• Elementary subject object rule, with object conversion due to possessional relation with another object
9.	2	computer	bank	processes_ transaction	• Elementary subject object rule, with object conversion due to possessional relation with another object
10.	2	computer	account	processes_ against	• Subject and object of prepositional clause
11.	3	cashier_station	individual_ bank	owned_by	• Compound noun generators for both subject and object • Passive subject with agent
12.	3	cashier_station	computer	communicate_ with	• Subject and object of prepositional clause
13.	4	cashier	account_data	enter	• Compound noun generator • Elementary subject object rule • Adjective classifier–human denotes quality • cashier added as class, with attribute quality = human
14.	4	cashier	transaction_ data	enter	• Compound noun generator • Elementary subject object rule • Adjective classifier–human denotes quality • cashier added as class, with attribute quality = human
15.	5	atm	central_computer	communicates_with	• Compound noun generator • Subject and object of prepositional clause
16.	6	central_computer	transaction	Clears	• Compound noun generator • Elementary subject object rule
17.	6	central_computer	bank	clears_with	• Compound noun generator • Subject and object of prepositional clause
18.	7	Atm	cash_card	accepts	• Compound noun generator • Elementary subject object rule
19.	7	atm	user	interacts_with	• Subject and object of prepositional clause
20.	8	atm	central_ computer	communicates_with	• Compound noun generator • Subject and object of prepositional clause
21.	9	atm	receipt	prints	• Elementary subject object rule
22.	9	atm	cash	dispenses	• Elementary subject object rule
23.	10	system	record-keeping	requires	• Elementary subject object rule • Adjective classifier–appropriate denotes opinion • Record-keeping added as class, with attribute opinion = appropriate
24.	10	system	security_provision	Requires	• Compound noun generator • Elementary subject object rule
25.	10	system	concurrent_access	handle	• Elementary subject object rule • Adjective classifier–concurrent not classified • concurrent_access added as class name
26.	10	system	account	handle_to	• Subject and object of prepositional clause
27.	11	bank	bank	provide_ software	• Possession modifier bank's software changes the object and name of relation
28.	11	bank	computer	Has	• Possession modifier for bank, converted to association likely to become attribute
29.	11	bank	software	has	• Possession modifier for bank, converted to association likely to become attribute
30.	11	bank	computer	provide_software_for	• Preposition 'for' after the object of the sentence.

Aggregation classification

```
computerized_banking_network> cashier (Aggregation)
computerized_banking_network>atm(Aggregation)
```

The above two relations, from Table 6 become aggregations.

Composition classification

```
consortium > bank (composition)
```

Generalization classification

There are no instances of generalization in the given text. A sentence '*A computer is an electronic device*', would have yielded a generalization of the *computer* class.

Trivial associations to attributes

The trace for the determination of trivial classes is as below, along with the reasons:

```
Evaluating relation : bank-> computer ( has )
        Association : non-trivial (has/have) association retained bank : computer
```

In the relation *bank has a computer*, *bank* is retained as a separate class, because of further description of the *bank* class.

```
Evaluating relation : bank -> account ( has )
        Attribute : (has/have) association moved to attribute bank : account
Evaluating relation : bank -> software ( has )
        Attribute : (has/have) association moved to attribute bank : software
```

Neither *account* nor *software* is further described, and does not exist as classes; so they are created as attributes of the class *bank*.

Trivial relations to operations

Relationships that refer to non-existing classes become class operations, because the classes that they refer to are trivial references. In the trace below, the second entity does not exist as a class.

```
Evaluating relation : computer -> account ( processes_against )
        Trivial Association -> Class Opern : computer : processes_against_account
Evaluating relation : system ->concurrent_access( handle )
        Trivial Association -> Class Opern : system : handle_concurrent_access
Evaluating relation :atm ->cash_card ( accepts )
        Trivial Association -> Class Opern :atm : accepts_cash_card
Evaluating relation :atm -> cash ( dispenses )
        Trivial Association -> Class Opern :atm : dispenses_cash
Evaluating relation :atm -> cash ( prints )
        Trivial Association -> Class Opern :atm : prints_cash
Evaluating relation :central_computer -> transaction ( clears )
        Trivial Association -> Class Opern :central_computer : clears_transaction
Evaluating relation : system ->security_provision ( requires )
        Trivial Association -> Class Opern : system : requires_security_provision
Evaluating relation : system -> account ( handle_to )
        Trivial Association -> Class Opern : system : handle_to_account
```

```
Evaluating relation : cashier ->account_data ( enter )
        Trivial Association -> Class Opern : cashier : enter_account_data
Evaluating relation :atm -> user ( interacts_with )
        Trivial Association -> Class Opern :atm : interacts_with_user
Evaluating relation : system -> record-keeping ( requires )
        Trivial Association -> Class Opern : system : requires_record-keeping
Evaluating relation : cashier ->transaction_data ( enter )
        Trivial Association -> Class Opern : cashier : enter_transaction_data
```

Association classification

The remaining relationships after going through all the above steps are retained as associations.

```
Evaluating relation : bank -> bank ( provide_software )
        Association : bank : provide_software_bank
Evaluating relation :computerized_banking_network -> consortium ( shared_by )
        Association :computerized_banking_network : shared_by_consortium
Evaluating relation : computer -> bank ( maintains_accounts )
        Association : computer : maintains_accounts_bank. . .
```

Trivial class removal

In this step, a pre-defined list of words to be removed is included. These words are taken away from the list of classes and the relations. In the ATM example, the word '*system*' may be denoted as a trivial class to be removed. The final results after the execution of all the above steps are marked in Table 7. The table shows the actions done on Table 6.

The conceptual model may be rendered using concepts from Vidhu Bhala et al. (2012) to achieve pragmatic efficiency. The conceptual model is created by programmatically creating a code in the DOT programming language, which is part of Graphviz graphic visualization software (Ellson, 2002). The DOT layout offers a hierarchical layout, with the edges and nodes laid out in a top to bottom, left to right manner.

The priority assigned to each class is determined by the normalized frequency of occurrence. The normalized frequency is determined by the frequency of reference to the class name in the process of the creation of the design elements, and may not be a direct count of occurrences in the text. The resulting output is shown in Figs. 3 and 4. In the displayed images, *atm* and *bank* are the top two conceptual classes that stand out.

5. Performance evaluation

The Performance evaluation of this work was a challenge, because there is no definition of a 'correct' conceptual model. The conceptual models that are traditionally created were found to contain implicitly added knowledge by the analyst (which may be incorrect, because of assumptions). Implicit information in the conceptual model could be related to classes or relations or both, that are not mentioned in the requirements text. For instance, in the ATM example, *Bank owns Bank Computer* represents an implicit assumption on a relation, while the class *Remote Transaction* is an implicit assumption that a certain class exists, although it is not stated in the requirements explicitly.

The evaluation is thus, biased against our implementation, because our implementation does not consider implicit information, but rather only models what is explicitly stated in the requirements. There are very few standard models which demonstrate conceptual models against given texts, with the model aligning exactly with the text, rather than with an analyst's opinion of the system.

For the purpose of comparison, the standard models will constitute, where appropriate, concepts from standard books or author results, and implicit information, but will consciously, but subjectively, avoid any assumptions made on the end system that are not stated in the requirements. A separate evaluation is done for implicit information added to facilitate comparison.

Because some conceptual classes are converted to attributes or operations based on how much they have been elaborated on, the

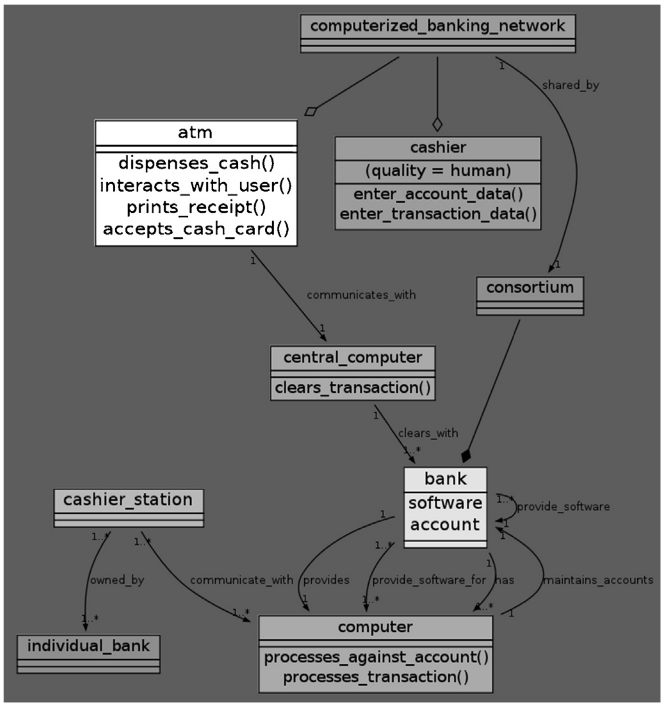


Fig. 3. Sample implementation – conceptual model in UML format.

Table 7
Results of relation type classification.

S No	Class 1	Class 2 (may not exist)	Relation name
1.	computerized_banking_network	atm	Includes
~~2.~~	~~system~~	~~computerized_banking_network~~	~~Support~~
3.	computerized_banking_network	cashier	Includes
4.	computerized_banking_network	consortium	shared_by
5.	consortium	bank	Of
6.	bank	computer	Provides
7.	bank	account	Has
8.	computer	bank	maintains_accounts
9.	computer	bank	processes_transaction
10.	computer	account	processes_against
11.	cashier_station	individual_bank	owned_by
12.	cashier_station	computer	communicate_with
13.	cashier	account_data	Enter
14.	cashier	transaction_data	Enter
15.	atm	central_computer	communicates_with
16.	central_computer	transaction	Clears
17.	central_computer	bank	clears_with
18.	atm	cash_card	Accepts
19.	atm	user	interacts_with
20.	atm	central_computer	communicates_with
21.	atm	receipt	Prints
22.	atm	cash	Dispenses
~~23.~~	~~system~~	~~record-keeping~~	~~Requires~~
~~24.~~	~~system~~	~~security_provision~~	~~Requires~~
~~25.~~	~~system~~	~~concurrent_access~~	~~Handle~~
~~26.~~	~~system~~	~~account~~	~~handle_to~~
27.	bank	bank	provide_software
28.	bank	computer	Has
29.	bank	software	Has
30.	bank	computer	provide_software_for
	Legend		
~~Strk~~	Trivial classes removed	🟧	Aggregation
🟩	Trivial relations moved to operations	🟦	Composition
🟠	Trivial associations moved to attributes	🟨	Remaining association

evaluation, where appropriate, also allows for such trivialized entities to be counted as conceptual classes (making the assumption, that if the text is further elaborated, they will become classes). Two modes of evaluation are considered:

1. Evaluation against a sample from books or from experts
2. Evaluation against human performance

5.1. Performance metrics used

We use performance metrics to compare our automatically generated conceptual model against a standard published model or a human model. The chosen metrics are precision, recall and over-specification, as recommended by Elbendak et al. (2011). Precision indicates the correctness or the relevance of the classes identified in the conceptual model. The recall percentage indicates the ability of the automation to generate all classes. The over-specification percentage indicates the number of unnecessary, but correct classes that the automation process includes in the generated conceptual model. The formulae for these measures are given below:

$$Recall = \frac{N_{correct}}{N_{correct} + N_{missing}} \quad (7)$$

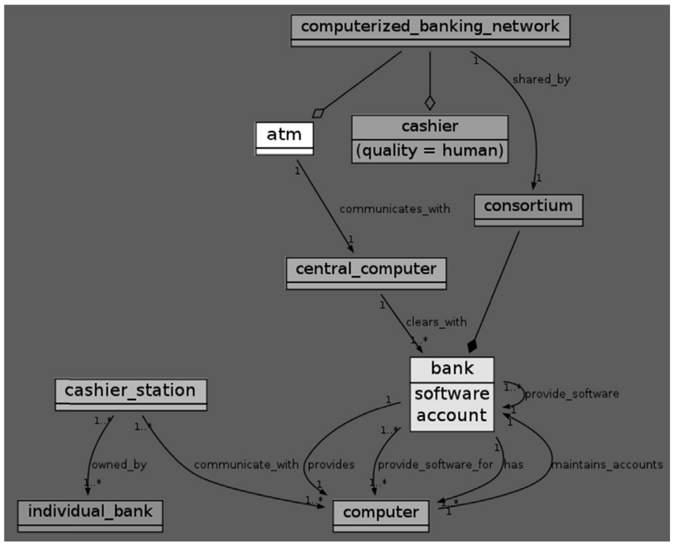


Fig. 4. Conceptual model with only classes, attributes and relations.

$$Pescision = \frac{N_{correct}}{N_{correct} + N_{incorrect}} \tag{8}$$

$$Over\text{-}specification = \frac{N_{extra(valid)}}{N_{correct} + N_{missing}} \tag{9}$$

$N_{correct}$ is the number of correct classes identified; $N_{incorrect}$ is the number of correct classes identified as wrong; $N_{missing}$ is the number of classes extracted by the human expert and missed by the system; $N_{extra(valid)}$ is the number of valid extra classes retrieved (different from spurious).

Although the model is not meant for understanding implicit knowledge, we evaluate it to compare it with human created models. To calculate this, we introduce another variable, $N_{implicit}$, which denotes the number of classes that are added that are correct and valid, but not stated in the requirements.

The formulae used to separate implicit knowledge are

$$Recall_{implicit} = \frac{N_{correct}}{N_{correct} + N_{implicit} + N_{missing}} \tag{10}$$

$$Precision_{implicit} = \frac{N_{correct}}{N_{correct} + N_{incorrect}} \tag{11}$$

$$Over\text{-}specification_{implicit} = \frac{N_{extra(valid)}}{N_{correct} + N_{implicit} + N_{missing}} \tag{12}$$

While some reference values are available for measures that do not include $N_{implicit}$, published target values do not exist for any of the measures. So, for the purpose of our evaluation, we define favorable conditions to evaluate the performance. Recall and Precision should be as high as possible, i.e., close to 100%, to accurately represent the target model. Over-specification should be low, i.e., close to 0%, to avoid adding too many additional details.

5.2. Results of conceptual class modeling against standard models

Table 8 shows the results of the performance evaluation against some case studies. The conceptual models that were generated for the EFP and the course registration case studies are shown in Figs. 6 and 7 respectively, while the ATM conceptual model is shown in Fig. 3. These case studies were used by various authors to demonstrate the creation of class diagrams, and the corresponding results are found in the respective references, i.e., ATM (Rumbaugh et al., 1991), EFP (Kurt, 1995) and Course registration (IBM Corp, 2004). It must be noted, however, that these standard

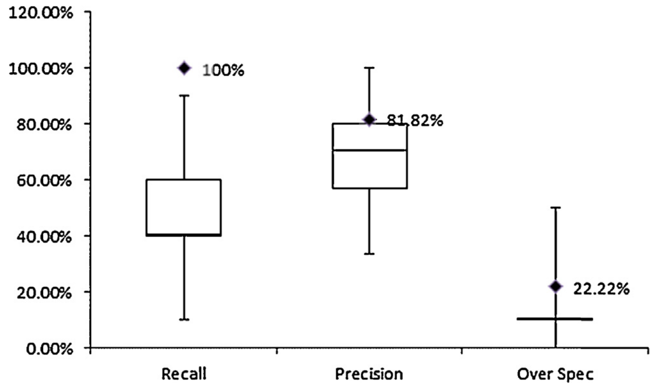


Fig. 5. Manual results – statistical analysis.

models were neither built for the requirement analysis, nor created automatically, and thus incorporate an element of human intelligence, which our implementation lacks.The application of human judgment to add relevant classes is demonstrated by the drop in precision. Our implementation added some classes that would be considered incorrect. The recall rate is very high because all classes are always considered candidate classes. However, the over-specification measure, which should be low, shows high values.Although high over-specification values cause visual clutter in the produced models, OO principles encourage over-specification over under-specification (Larman, 2002). Where there is no difference between the results for the two columns, the reason lies in the comparison model, rather than the proposed model's ability to model implicit information.

5.3. Results against human subjects

Due to lack of a standard reference text and domain models, testing was conducted on human subjects. The subjects were 37 final year engineering students majoring in Computer Science and Engineering. They had undergone a course as part of their curriculum, which trained them theoretically and practically, in creating class diagrams from natural language specifications. The results of their final CASE tools lab examination were taken for reference. The students were given the requirements of the Course Registration case study, and they spent about 30–45 min creating the model. Fig. 5 shows the statistical results for the manual results, indicating median values of approximately 40%, 70% and 10% for recall, precision and over-specification respectively. The results from Table 8 are also shown on the graph.

We see that our implementation is better than the overall students' performance. It compares favorably against human work and against expert judgment for precision and recall. Two main reasons for degraded over-specification are lack of semantic information, and lack of judgment of words that cannot be classes. For instance, in EFP, a class name *mean* was created for a sentence, '*a text document is created by author, word processor and some other means*'.

5.4. Sample results of the evaluation of the performance of relations

The same metrics, i.e., recall, precision and over-specification, are used to evaluate the performance of our implementation for the relationships among the classes. The results are shown in Table 9.

The results against human subjects also show similar results for associations, i.e., a very high over-specification rate (95–140%) and a lower recall rate (50–70%) where implicit information is added. For the relations that are associations, the results indicate that the

Table 8
Evaluation results.

Case study	Without implicit information or assumptions			With implicit information/assumptions		
	Recall (%)	Precision (%)	Over spec (%)	Recall (%)	Precision (%)	Over spec (%)
ATM (Rumbaugh et al., 1991)	100	91.67	9.09	91.67	91.67	8.33
EFP (Kurt, 1995)	100	94.44	41.18	85	94.44	35
Course registration (IBM Corp, 2004)	100	81.82	22.22	100	81.82	22.22

Table 9
Performance evaluation – relationships.

Case study	Relation	Without considering implicit information or assumptions (%)			Considering implicit information or assumptions (%)		
		Recall	Precision	Over spec	Recall	Precision	Over spec.
ATM	**Overall**	**83.33**	**100**	**100**	**50**	**100**	**60**
	Association	80.00	100	120	44	100	67
	Composition	100	100	0	100	100	0
VSS	**Overall**	**100**	**80**	**100**	**57**	**80**	**57**
	Generalization	100	100	0	100	100	0
	Association	100	75	133	50	75	67
EFP	**Overall**	**93**	**88**	**93**	**61**	**88**	**61**
	Aggregation	100	100	0	67	100	0
	Generalization	100	100	100	100	100	100
	Association	86	75	171	50	75	100

over-specification is very high when compared to manual results. This refers to valid relations that are not considered important or relevant for inclusion in a conceptual model. Our implementation uses all forms of verbs to interconnect classes, because it is limited to a textual analysis. Further coarse-graining may be required to retain only the most important relations among classes, thereby imitating the fuzzy nature of human thinking.

When we consider cases where implicit information is used to apply assumptions on the relations based on human judgment, the recall rate also drops for associations. There are two reasons for this. Semantic relations are used to automatically link conceptual classes, e.g., *a bank – has – accounts*, or *cashier transaction – entered on – cashier station*, even when not stated in the requirements. Another reason is that, two or more 'verb' relations among classes are compressed into one single 'verb', e.g., *bank owns computer*, replaces other relationships like *bank provides computer*, or *bank provides software* for computer, etc.

The results show a fairly good performance for relationships like generalizations, compositions and aggregations, in comparison to the associations. These are special cases, and do not have much ambiguity or semantic compression.

6. Conclusion and future work

This paper has presented an approach to automate the generation of a conceptual model from stated requirements. The results of the work so far indicate that a syntactic analysis on explicitly stated requirements is a primary step toward creating satisfactory conceptual models, which may be used within the requirements phase. The results compare favorably with human novice modeling.

Our implementation has been able to successfully create conceptual models with reasonable precision and recall in a fully automated manner. We have been able to successfully apply various performance measures to evaluate the quality of the automatically generated models against human judgment, both expert and novice.

The performance analysis shows that over-specification of the conceptual classes and relations are a problem that might occlude human comprehension for larger requirements models. Rather than attempting to coarse-grain, the concepts that translate to classes in the model are assigned scores, and rendered as per these to provide the context visually.

During the course of our research, we realized that there cannot be a comprehensive coverage of all kinds of natural language texts, for several reasons. Firstly, there exists no way to guarantee that a given sentence is syntactically correct, and that the Parser chosen (Stanford) will yield the expected parse for it. Secondly, even if the results are correct, the sentence structure may not have been coded in the rules, to extract the classes and relationships. Thirdly, there exists no corpus of standard sentences that might be used in the language of requirements. This is evident from the fact that starting from 12 rules in an established work, our implementation has devised a total of 38 rules that may still not be sufficient to cover all kinds of sentence structures.

A further constraint that was encountered was the inherent ambiguity in the English language, especially in the area of determining attributes (or aggregations) and in generalization. For example, whether *book* is an attribute of *author*, or *author* is an attribute of *book* depends on the availability of further information. Although our implementation overcomes such problems to some extent through deferring the decision of attribute vs. association or relation vs. operation, problems inherent in English linguistics cannot be resolved.

A few other limitations of the implementation are listed below

Relation attribute modeling – modeling relation attributes as separate classes requires a deep NLP analysis of the prepositional phrases related to adverbs. This slightly advanced feature of the conceptual model was not considered. Some researchers do advocate that relationship attributes should not be modeled.
Implicit or non-stated modeling based on human intelligence – unstated or implicit information may be generated through a semantic analysis of the relationship between the classes. Because this information is not yet completely available, an implicit i.e., Semantic processing of the conceptual model is not done.
Sentence structure considerations – as in all NLP projects, the set of rules identified is intricately associated with the set of sample case studies and examples from various sources. This does not indicate the completeness of the rule set, because there may be peculiar sentence constructions for which rules may not be framed, failing to identify appropriate subjects, objects and

relations. This can also happen when the construction of the sentence is 'considered' to be inappropriate in the context of the Parser. Sentences without subjects like predicative sentences or passive sentences with no agent are considered inappropriate inputs for requirements. No decisions can be made out of them.

6.1. *Future enhancements*

The work presented in our research can be extended in many interesting ways. We consider two directions of enhancements.

With the objective of improving the efficiency of the requirements analysis activity, the work may be enhanced to offer an analysis of the model. This could mean the incorporation of semantic information and/or existing knowledge into the model, to add implicit knowledge, and the ability to judge the completeness of the conceptual model and prompt the analyst in the direction of better formed conceptual models. This activity involves an assessment of what constitutes 'sound' conceptual classes, and what information is necessary for 'completeness'. Another way to improve the requirements process could be the ability to quantify the contribution of each relationship or class to the overall specification of the conceptual model, i.e., to determine what relationships or operations carry more significance to the specification of the concept. This can help the participants in the requirements phase, to be aware of the completion percentage of the requirements.

Yet another objective of enhancements could be to improvise on the conceptual model itself, with the objective of reusing the model during the design phase. This objective also requires semantic information of the identified concepts, to identify and retain only the necessary classes, attributes and relations, as suggested by the OO design principles (Larman, 2002).

In summary, the next big step to take in either direction would be the incorporation of semantic information in the conceptual model, to make sense of the concepts to either improve the conceptual model for the requirements analysis, or for conversion to a design model like a class diagram.

Acknowledgement

Further work on this research is getting supported by UGC, New Delhi, India under Major Research project scheme of Engineering Sciences– F.no. 42-129/2013(SR).

Appendix. Conceptual models generated for EFP and course registration

See Figs. 6 and 7.

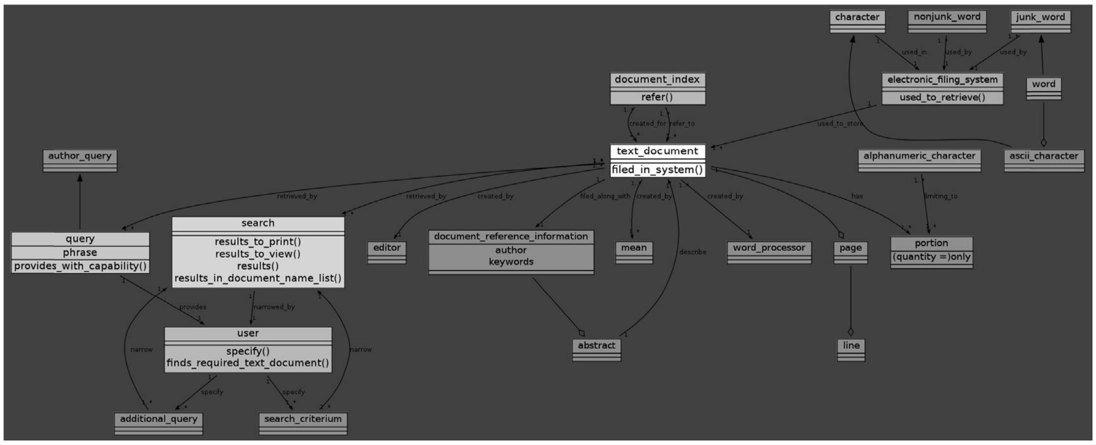


Fig. 6. UML style conceptual model for EFP standard case study (Kurt, 1995).

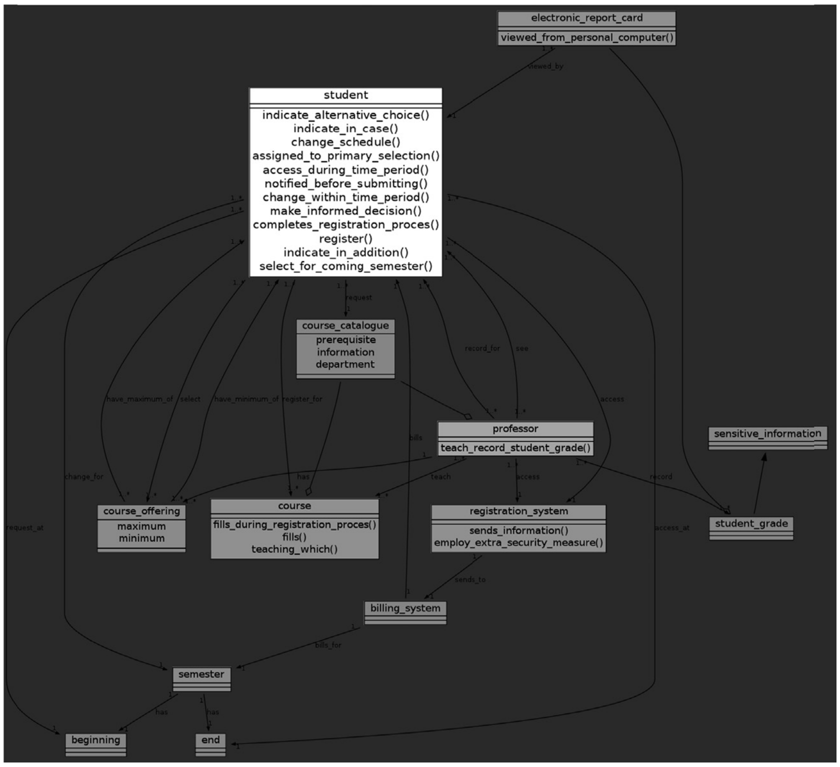


Fig. 7. UML style conceptual model for Course Registration standard case study (IBM Corp, 2004).

References

Ambler, S., 2005. The elements of UML 2. 0 style. Cambridge University Press.
Ambriola, V., Gervasi, V., 2006. On the systematic analysis of natural language requirements with CIRCE. Journal of Automated Software Engineering 13 (January (1)), 107–167.
Bird, S., Klein, E., Loper, E., 2009. Natural Language Processing with Python. O'Reilly Media.
Booch, G., Maksimchuk, R., Michael, E., Young, J., Conallen, J., Houston, A., 2010. Object – Oriented Analysis and Design with Applications. Pearson Education.
Burg, J., 1997. Linguistic Instruments in Requirements Engineering. IOS Press.
Col, J., 2013. Adjective and a list of Adjectives. In: Enchanted Learning, Available at: http://www.enchantedlearning.com/grammar/partsofspeech/adjectives/
News and Events, 2013. European Commission – CORDIS, Available at: http://cordis.europa.eu/fetch?CALLER=EN_NEWS_EVENT&ACTION=D&DOC=39&CAT=NEWS&QUERY=0132d8bc3aca:de9c:21c94748&RCN=33343
Elbendak, M., Vickers, P., Rossiter, N., 2011. Parsed use case descriptions as a basis for object-oriented class model generation. Journal of Systems and Software 84 (July (7)), 1209–1223.
Ellson, J., Gansner, E., Koutsofios, L., North, S., Woodhull, G., 2002. Graphviz—Open Source Graph Drawing Tools. Graph Drawing: Lecture Notes in Computer Science, vol. 2265. Springer, Berlin/Heidelberg, pp. 594–597.
IBM Corp: IBM Rational Software, 2004. Section 1: Course Registration Requirements.
Jurafsky, D., Martin, H., 2000. Speech and Language Processing, 1st ed. Pearson Education.
Kurt, W.D., 1995. Applying OMT: A Practical Step-by-Step Guide to Using the Object Modeling Technique. SIGS Publications, NY, USA.
Larman, C., 2002. Applying UML and Patterns – An introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd ed. Pearson Education India.
Leffingwell, D., 2011. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise, 1st ed. Addison-Wesley Professional.
LindLand, O., Sindre, G., Solvberg, A., 1994. Understanding quality in conceptual modeling. IEEE Software 11 (March (2)), 42–49.
Loberger, G., Shoup, K., 2009. Webster's New World English Grammar Handbook.
2013. Morphy (7WN) manual page. In: WordNet: A lexical database for English, wordnet@princeton.edu, Available at: http://wordnet.princeton.edu/man/morphy.7WN.html
Mu, Y., Wang, Y., Guo, J., 2009. Extracting software functional requirements from free text documents. In: International Conference on Information and Multimedia Technology, Jeju Island, South Korea.
2013. Stanford Parser – Spatial Language, Available at: http://projects.csail.mit.edu/spatial/Stanford_Parser
Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., 1991. Object-Oriented Modeling and Design. Pearson Education.
Sanjay, A., Sankar Basu, S., Choudhury, S., 2010. A requirement framework for enablement of automatic generation of domain model. In: Computer Information Systems and Industrial Management Applications (CISIM), Krackow, vol. 10.1109/CISIM. 2010.5643633, pp. 359–364.
Vidhu Bhala, R.V., Mala, T., Abirami, S., 2012. Effective visualization of conceptual class diagrams. In: 2012 International Conference on Recent Advances in Computing and Software Systems, Chennai, India, pp. 1–6.
Yue, T., Briand, L., Labiche, Y., 2011. A systematic review of transformation approaches between user requirements and analysis models, 16. Springer: Requirements Engineering, pp. 75–99.

Vidhu Bhala R. Vidya Sagar works with Mindtree, India. She pursued her Masters degree in the department of Information Science and Technology at Anna University, CEG. She has more than 13 years of experience in the software industry and her interests are in the area of requirements elicitation and analysis and the ability to validate and enhance software requirements elicited from customers. Her current research work includes processing of natural language text to extract artifacts useful for requirements analysis and validation.

Dr S. Abirami is an Assistant professor in the department of Information Science and Technology at Anna University, CEG for past 6 years. Her areas of interest include natural language processing, image processing, multimedia technologies and data mining.