Object Oriented Visualization of Natural Language Requirement Specification and NFR Preference Elicitation

G.S. Anandha Mala and Dr.G.V.Uma
malamanosuke@yahoo.co.in gvuma@annauniv.edu
Department of Computer Science and Engineering, Anna University, Chennai, Tamil Nadu, India-600025

Summary
Requirements engineering is where the formal meets the informal. Application of natural language understanding to requirements gathering remains a field that has only limited explorations so far. Further, automation of requirement gathering is still in its infancy. There are three module proposed in this paper. In the first module, an approach for automatic requirements capture from natural language requirements specification is proposed. This approach starts by subjecting the natural language text for identification of parts of speech of the words in each sentence by applying sentence tagging techniques. The text thus tagged is normalized to simple sentences. Further, to resolve the ambiguity posed by the pronoun, the pronoun resolution is performed on the simplified text. Then, the elements of the object oriented system namely the classes, the attribute, methods and relationships between the classes, sequence of actions, the usecases and actors are identified by mapping the 'part of speech tagged' words of the natural language text onto the object oriented modeling language element, using some mapping rules based on the classical noun-verb analysis, thus eliciting the system requirements. In the second module, from the elicited object oriented elements, a semi-automatic approach of design and development of ontology for the requirement specification is proposed. Ontologies are especially useful for the development of high-level reusable software, like domain models and frameworks. They provide an unambiguous terminology that can be shared by all involved in the development process. Modeling of software requirements as ontology is done by, converting the object oriented elements elicited from the requirement specification as Resource Description Framework. By storing the ontology in database, the user can query and acquire the domain knowledge. This way, the same ontology can be used to guide the development of several applications, diluting the cost of the initial stage and allowing knowledge sharing and reuse. Requirement engineering plays a vital role in the development of the software. The quality of the software being developed depends on the Non-Functional Requirements(NFR), which are still not derived effectively due to the conflicts between NFRs. In the third module of this paper, a new approach is proposed to identify the NFRs for a given usecase description from the domain model such as Unified Modeling Language (UML) class diagram, and goal based questionnaires. This approach makes use of the domain model to find out the behavior of the system and possible constraints for actors in the system. The NFR taxonomy and the user preference are used to analyze the conflicts, which is resolved based on trade-off analysis by prioritizing the preference. The prioritization depends on the dominating NFRs from the inference engine.

Key words:
Natural Language Processing, Object Oriented Analysis, Object Oriented Design Models, Ontology, Non-Functional Requirements, Requirement Specification.

Introduction

Software development invariably begins with some human need or desire - to explore or solve a problem. We use natural language to describe our needs and problems, but it's often complex, vague and ambiguous. Sentences are vague when they contain generalizations, or they are missing important information, especially the subject or objects needed by a verb for completeness, or containing pronouns. Sentences are ambiguous when they are open to multiple interpretations. All these troubles arise when we discuss our needs and problems using natural language. On the other hand, software requires more precision, formality and simplicity than that commonly found in natural language. Also, the translation from requirements to code often reduces, eliminates or distorts much of the original meaning intended by the requirements. Given these factors, it's not surprising that the translation of natural language descriptions into usable software poses quite a challenge. We need ways to reduce ambiguity and complexity without sacrificing the richness and meaning of natural language. This paper describes technique for transforming natural language requirement specification into Object Oriented Modeling Language elements, which help fill the gap between the informal natural language used to describe problems and the formal modeling languages used to specify software solutions. Also a semi automatic approach of design and development of ontology for the requirement specification, for providing the facilities of sharing and reuse of the domain information among the developers and an efficient way of eliciting Non Function Requirements preference for actor's of use case from domain model are

presented in this paper. The paper begins with a review of advances in the field of requirements engineering and various definitions given for ontology and their related works in section 2. The proposed methodology is explained in section 3. Our implementation results are discussed in section 4. The conclusions and future work is contained in Section 5.

2. Related Works

Although it has been proven that Natural Language processing with holistic objectives is very complex, it is possible to extract sufficient meaning from NL sentences to produce reliable models. Advances in the field of requirements elicitation is discussed in this section, which is followed by the various definitions given for ontology.

The first relevant published technique attempting to produce a systematic procedure to produce design models from NL requirements was Abbot [1]. Abbot suggested a non-automatic methodology that only produces static analysis and design products obtained by an informal technique requiring high participation with that of users for decisions. Methods to bring out a justified relationship between the natural- language structures and OO concept are proposed by Sylvain [19] who shows that computational linguistic tools are appropriate for preliminary computer assisted OO analysis. Sawyer in their REVERE [16] makes use of a lexicon to disambiguate the word senses thus obtaining a summary of requirements from a natural language text but do not attempt to model the system. Liwu Li [10] also presents a semi-automatic approach to translate a use case to a sequence diagram. It needs to normalize a use case manually. Overmyer [14], also present only a complete interactive methodology and prototype. However, the text analysis remains in good part a manual process. Liu [4] present an approach, which uses formalized use cases to capture and record requirements. Ke Li [9] also semi-automate the process of requirement elicitation where the text is matched with predefined statements. If there is no match then get help from user to clarify incomplete/ambiguous data. Participation of domain experts, customer are needed in class identification process in contrast to our fully automatic methodology which is named here as “Domain Knowledge Elicitor”.

This paper also aims at providing the facilities of sharing and reuse of the domain information among the developers in the form of ontology. It is possible to find in the literature several definitions for ontology. One of the most cited is the one proposed by Gruber, “An ontology is a formal, explicit specification of a shared conceptualization” [8]. The definition proposed by Gruber is general; however, ontology can be defined in specific contexts. For example, taking the paradigm of agents into account, [15] establish that ontology is a formal description of the concepts and relations, which can exist in a community of agents. The importance of the terms of ontology can be perceived in the next definition: “An ontology is a hierarchically structured set of terms to describe a domain that can be used as a skeletal foundation for a knowledge base [18]”. More recent definitions of ontologies are the following ones: “An ontology is a common, shared and formal description of important concepts in an specific domain [6]”; “An ontology is a formal explicit representation of concepts in a domain, properties of each concept describes characteristics and attributes of the concept known as slots and constrains on these slots [13]”. Sometimes concepts are termed classes, properties are also known as roles while facets are used rather than slots. “An ontology is a theory which uses a specific vocabulary to describe entities, classes, properties and related functions with certain point of view [7]”. Ontology necessarily includes a specification of the terms used, ("terminology") and agreements to determine the meaning of these terms, along with the relationships between them [17]. Like Fonseca’s [7] definition of ontology, the method proposed in this paper for the construction of ontology consists of classes, subclasses, attributes, methods, and the relationships among the classes. The ontology can be constructed for any domain with the help of requirement specification. The pre-processor module of “Domain Knowledge Elicitor” in the system architecture shown in figure 1 identifies the key concepts required for the construction of ontology.The quality of the software being developed depends on the non-functional requirements. Haruhiko Kaiya [27] discusses the elicitation of the non-functional requirement performed by comparing the existing usecase to derive the invariants related to the non-functional requirements and the stakeholders involved. GQM approach is used for the trade-off analysis based on the stakeholder’s goal preferences. NFR taxonomy is used for the conflicting NFR identification. The system does not focus on the internal description of the usecase as well as the quality requirements are not prioritized. But our system considers the usecase description to identify the system interaction by comparing it with the domain model; also trade-off analysis is used to prioritize the quality requirements.

3. Proposed System Architecture

In all the earlier works mentioned about requirements elicitation, the process is not fully automatic. User assistance is required in several places example pronoun resolution, class identification etc., Also domain modeling and extracting information from use case models have done separately. We present i) a new methodology for domain knowledge (functional requirements) elicitation and developing ontology for them, which includes automatic reference resolution and eliminates the user intervention as in the previous works, ii) and a new approach for NFR preference elicitation for actors of usecase by comparing the usecase with domain model in our case high level class diagram. The method of visualizing the requirements specification as ontology by applying natural language techniques is explained in figure 1. The figure 4 tells, how to capture the NFR preference for actor's of usecase from domain model.

3.1 Domain Knowledge Elicitaion and Ontology Construction

Ontologies are key elements required to enable knowledge exploitation and information retrieval of systems. Essentially, domain ontologies are made of sets of concepts and the relationships that can be expressed among those concepts. Since the building blocks of domain ontologies are concepts and relations among concepts that describe an application domain, NLP techniques are used to retrieve the object-oriented concepts namely classes, attributes, methods and relationship among the classes. The concepts identified are represented in the form of ontology and effectively stored in the database for easy information retrieval. The architecture for functional requirements elicitation and ontology representation has the following modules, 1) Domain Knowledge Elicitor, 2) RDF Generator 3) RDF Parse Engine 4) Data Storage and Query Engine

3.1.1 Domain Knowledge Elicitor

The proposed methodology for domain knowledge elicitation includes the automatic reference resolution. It takes an input a natural language text describing the requirements for a system and identifies the object-oriented system elements namely classes, attributes, methods, and relationships among the classes. "Domain Knowledge Elicitor" does this job with the help of pre-processor, Normalizer and NL-OOML mapper.

Pre-Processor The given input problem statement is split into sentences to ease the further processing. Each sentence of the input text is subjected to tagging in order to get the parts of speech marker for every word in a sentence. Tagging of the words is necessary to chunk the words that form a noun or verb phrase. Also the words that are candidates for classes, attributes, methods, use cases and actors have to be chosen depending upon their tags. We make use of the Brill tagger for this purpose. The noun and the verb phrases are identified based on simple phrasal grammars. In the sentence, the subjects and objects sometimes happen to be pronouns. In that case, they have to be resolved to their respective noun phrases. We make use of the Mitkov's 'Pronoun resolution with limited knowledge' algorithm for this task and get all pronouns resolved.

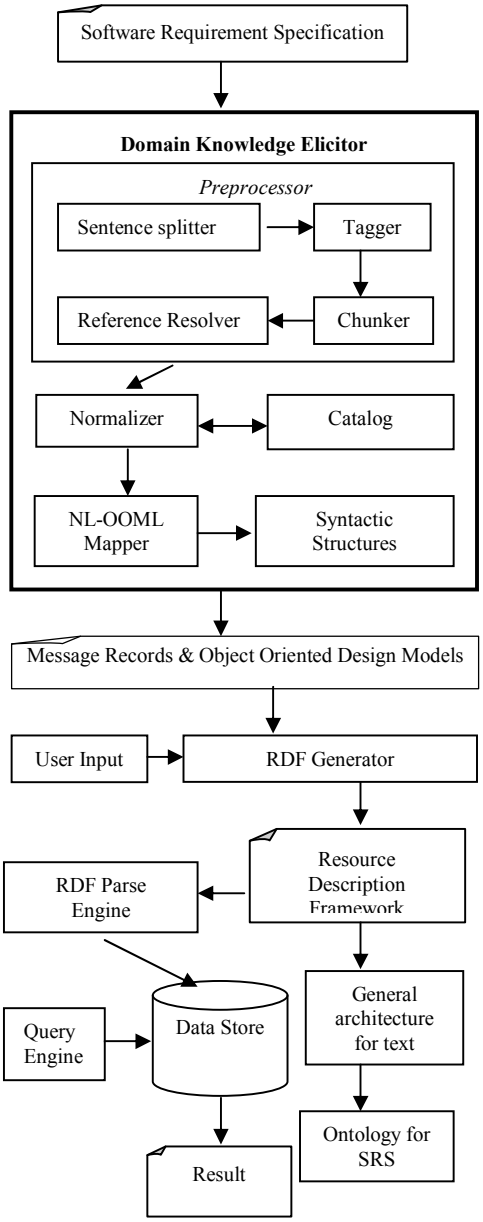

Fig. 1 Visualizing the Requirements Specification as Ontology

Normalizer The text has to be simplified into the following constructs to ease the task of mapping the words onto the Object Oriented system constituents.

Conditional: Conditional syntax is
If Condition transaction else otherTranstions
The else clause is optional.
Iteration: Iteration syntax is
While condition transactions endwhile.
Concurrency: Concurrency syntax is
Start concurrency transactionl
concurrent transaction2
...
concurrent transactionk
end concurrency
which executes transactionl, transaction2, . . . , transactionk concurrently.
All the transaction statements are simple. To break compound sentence into simple first the sentence is checked whether it contains any conjunction or not by tracing the POS tags appended to each word. If so, the number of conjunctions present is identified and split into sentences that are between the conjunctions. A number of patterns using conjunctions and their corresponding splits in the sentences are stored in a file. Each sentence is checked against the stored patterns and the corresponding split up is made. For example, the statement, "If the source and the destination of the request fall on the same route, the receptionist checks the seat that are available and issues the ticket to the passenger and blocks the seat", is identified as control statement using the keywords 'if'. The transactions of the control statements are simplified by breaking the conjunction and transferred to following conditional after reference resolution,

If the source and the destination of the request fall on the same route
The receptionist checks the seat.
The receptionist issues the ticket to the passenger
The receptionist blocks the seat
End if

NLOOML Mapper The NL-OOML mapper accepts a normalized problem description as input. It recognizes nouns, verbs, adjectives, and prepositions in a sentence from the preprocessor. It uses catalog to store user's instructions on how to translate some sentences. It translates each normalized sentence into a message record, which records a message sender, a receiver, and a message according to the table 1. Simple rule based approach is followed for identifying actors, usecases, class attributes etc., List of rules are given below:

1: Translating Nouns to Classes. A noun, which does not have any attributes, need not be proposed as a class.
2: Translating Noun-Noun to Class-Property according to position. When two nouns appear in sequence in the text, the first Noun is translated to Class and the following Noun is translated to properties of this Class.
3: A simple heuristic is used to decide which nouns are classes, and which form the attribute. In Noun-Noun, if the first noun is already been chosen as the class then the second noun is taken as the attribute. The attributes are decided based on the verb phrase.
4: Translating the lexical Verb of a non-personal noun to a Method of this noun. Decide the sender, receiver classes and argument to this method based on the Table 1.
5: Translating the lexical Verb of a personal noun to a use case (or part of a use case) linked with an actor defined by this noun.
6: Matching a Noun to a Personal Pronoun as the nouns of previous sentence.

Table 1: Syntactic Structures of Simple Sentences

No.	Syntactic Structure	Sender	Receiver	Action	Argument
1	subject verb object	subject	object	verb	-
2	subject verb object (to) verb1 (object1)	subject	object	verb1 (+object1)	(object1)
3	subject verb object participle (object1)	subject	object	participle verb (+object1)	(object1)
4	subject verb object adjective	subject	object	be + adjective	-
5	subject verb object conjunctive to verb1 (object1)		subject	verb	object, verb1 (+object1)
6	subject verb gerund (object)		subject	verb	gerund verb

					(+object)
7	subject verb object preposition object1	subject	object1	verb + object	(object)
8	subject verb object object1	subject	object	verb	object1
9	subject verb (for) complement		subject	verb	complement
10	subject verb		subject	verb	-
11	subject be predicative		subject	be + predicative	-

3.1.2 RDF Generator

After successful completion of domain knowledge extraction, the RDF Generator generates the RDF (Resource Description Framework) file using the domain knowledge elicited from the software requirement specification. The Generator concentrates on the class, subclass and properties of the RDF. The sample RDF code generated for the requirements specification of hospital management system is shown in figure 2. The RDF generated is fed as input to the GATE tool to visualize the RDF as ontology. The ontology for the hospital management system is shown in figure 3.

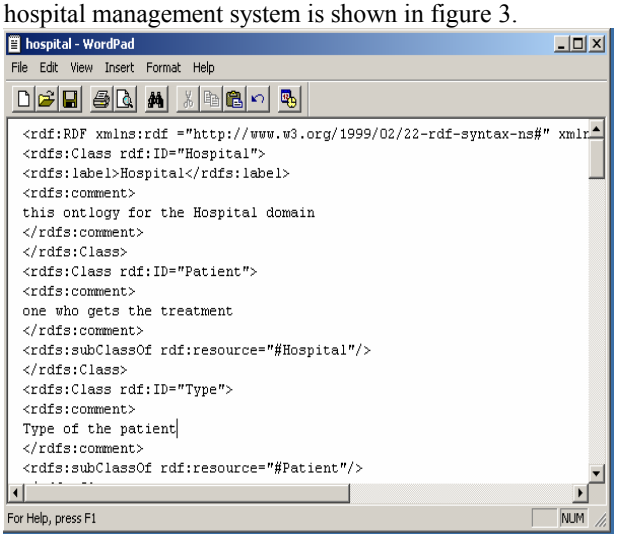

Figure 2. Sample RDF Code

3.1.3 RDF Parse Engine

The RDF generated is parsed to extract the information and stored in the dynamic database. The Parsing Algorithm is a generalized one, which takes care of the classes, subclasses and properties. The parsing algorithm of RDF is given below;

1. The complete RDF file of the domain is read and stored in a string.
2. The String Tokenizer is used for splitting the RDF file into tokens
3. Finding out if there exist more tokens after tokenizing.
4. Check the each token.
5. If the token is rdfs :class then the rdf:ID is extracted by identifying the index position of the “ and stored in the database.
6. If the token is rdfs:subclassof then the resource is extracted by identifying the index position of # and stored in the database.
7. If the token is a comment then the next token is extracted and stored in the database.
8. The Process is done until the end of class is identified and all the extracted information is stored in the database.
9. The Properties is extracted and the domain and the range value are stored in the database.

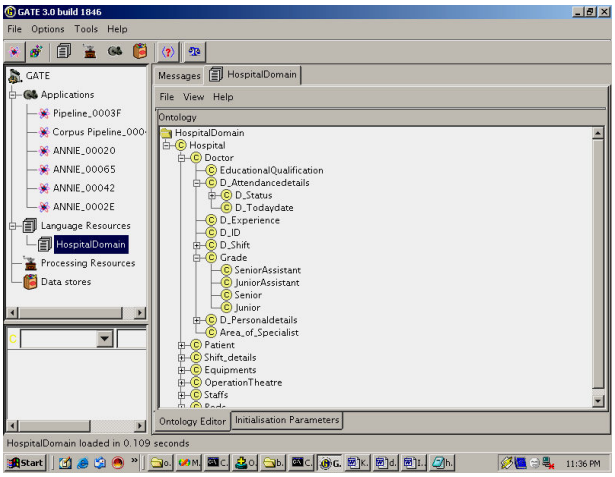

Figure 3. Ontology

3.1.4 Data Storage and Querying

The database is designed effectively to store domain information extracted and maintain the data in three tables. First table stores each class along with its subclasses and descriptions. The second one stores the properties along with its domain. The third one stores the range of the properties along with domain for which the properties belongs. The database can be queried to get the details like what kind of relationships holds between the classes and what can be the attribute value etc.,

3.2 Elicitation of Non-Functional Requirement Preference for Actors of Usecase from Domain Model

In the earlier works [22][23][26][28][29], domain modeling and extracting information from usecase models have done separately. We have suggested a method of combining both, and extracting the variants in usecase and combining with domain model and non-functional taxonomy to derive the actor's preference. The system architecture is shown in Figure 4.

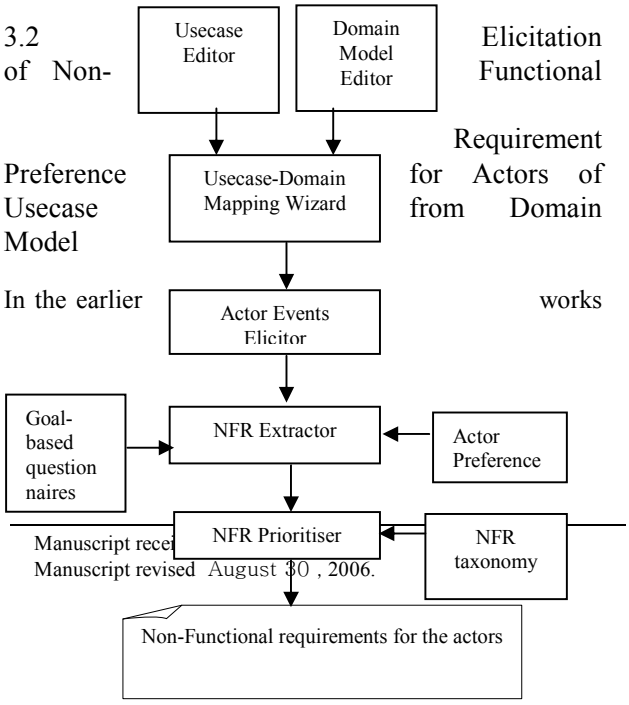

Fig. 4 System Architecture to capture the NFR preference.

Usecase and Domain model are structured using the XML editor to the Data Type Definition (DTD) format. The editor checks the syntax of both the usecase and domain model with the specified structure. They are represented with specified notations for easy traceability of the usecase description with the domain model. The syntactic structure of the usecase description is validated using natural language processing. The various syntactic structures used in usecase and domain model are listed in the Table 4. The Usecase-Domain Mapping Wizard extracts the entities, which are not present in the domain model from the usecase description by mapping the usecase with domain model. The usecase follows the below structure.

Title: a label that uniquely identifies the use case within the usecase model.
Primary Actor: the actor that initiates the use case.
Participants: other actors participating in the use case.
Goal: primary actor expectation at the successful completion of the use case.
Precondition: condition that must hold before an instance of usecase can be executed.

Manuscript received August 5, 2006.
Manuscript revised August 30 , 2006.

Postcondition: condition that must be true at the end of a 'successful' execution of an instance of the usecase.

Steps: Sequence of steps involved in the usecase along with extension.
Extensions: a set of step extensions that applies to all the steps in the use case.

Also it captures the non-existing actors, operations and conditional statements. Then updates the domain model using the reverse engineering wizard supported by the UML plug-in. The wizard uses the structured usecase and domain concept. Usecase editor and domain editor are used for this purpose. 'Actor Event Elicitor' maps the Usecase with the domain model based on the pre-conditions of the usecase. On successful completion the precondition states are withdrawn. The state chart for the entities, which are mapped with the domain model, is generated then. The state chart contains entries like "*1 --- [insert card]--> 2*", meaning that from state 1 it goes to state 2 on performing the event 'insert card'. From the state chart the events related to the actors alone are identified. 'NFR Extractor' generates the non-functional requirements for the actor events with the help of the actor preference and goal-based questionnaires. Actor preference is a matrix of actor versus event. The matrix entries tell whether the actor can perform the specified event or not. In goal-based questionnaires all the events are embedded with possible questions, which helps to identify the quality requirements. Sample actor preference matrix and goal-based questionnaires are shown in the Table 2. and Table 3. respectively. 'NFR Prioritizer' prioritizes the identified non-functional requirements based on the trade-off analysis. The results are shown in Fig. 5. This is done by the inference engine, which in turn makes use of the NFR taxonomy. In NFR taxonomy, all the NFRs are associated with other conflicting and dependable NFRs.

Table 2: Sample Actor Preference Matrix

Events	Patient	User	Doctor	Nurse
Triggers alarm	Yes	No	Yes	Yes
Press logout button	Yes	Yes	Yes	Yes
Enter vital signs	Yes	No	Yes	Yes
Insert Card	Yes	Yes	Yes	Yes
Connect cables	No	No	Yes	Yes

Table 3: Sample Goal-Based Questionnaires

Events	Preference	NFR
Insert card	Card Expired	Security
Enter pin	Invalid pin number	Security
Enter pin	Provide proper human computer interface	Usability

Table 4: Syntactic Structures used in Usecases and Domain Model

Statements	Syntax	Sample	Description
simple	[Determinant] entity verb value	User identification is valid	The value of the entity 'user identification' is 'valid'
complex	NO/NOT simple	Not User identification is valid	The value of the entity 'user identification' is 'not valid'
	[NO/NOT] simple AND/OR [NO/NOT] simple	User identification is invalid AND User number of attempts is equal to 4	The value of the entity 'user identification' is 'invalid' and the value of the entity 'user number of attempts' is 'equal to 4'
ANY statement	"ANY" "ON" entity [*]	ANY ON user*	This refers to all the conditions with "User" as the entity (e.g. "User is logged in"), but does not include conditions like "User Card" or "User identification" which are different entities associated with User
	"ANY" "ON" entity	ANY ON user	This refers to all the conditions with "User" as the entity (e.g. "User is logged in"), also includes conditions like "User Card" or "User identification" if present.
Operation declaration	action_verb [action_object]	Validate user identification	"validate" is the action verb and the action object is "user identification" which is an attribute of concept "User".
	[delay_specification] [condition_statement] [determinant] entity operation_reference	BEFORE 60 sec, USER enters pin	'Before 60 sec' is the delay specification,'User' is the entity ,'enters pin' is the operation_reference
		ATM asks user validation to the Bank	ATM is an entity. 'asks user validation to the bank' is operation_ reference
operation_reference	conjugated_action_verb [(binding_word)+] action_object [action_participant]	asks user validation to the bank	'Validation' is the conjugated action verb, 'to' is the binding word, 'bank' is the action_object
condition_statement	"IF" simple/complex "THEN"	IF User Identification is valid THEN, ATM displays operation menu	The simple statement 'User identification is valid' is taken as the condition
branch	[delay_specification] [condition_statement] "GOTO" ["STEP"] step_reference	Go to Step 2	Control transferred to step number 2

The NFR taxonomy looks like,
Usability#Simplicity+#Accessibility+#Installability+#Operability+#Maintainability
It states that simplicity, accessibility, installability and operability are directly proportional and maintainability is indirectly proportional with usability.

4. Results & Discussion

The entire system was implemented using JAVA and the 'Domain Knowledge Elicitor' was validated using 100 problem samples each of around 500 lines. The result produced by the system was compared with that of the human output. The human outputs were the results that were obtained by conducting the noun-verb analysis on the text. It was considered as the baseline and taken as expert judgment. The system does not miss to identify any of the classes and methods. But approximately 12.4% of additional classes and 7.4% of additional methods are identified in the entire sample taken, those that are removed by human by intuition that they may not be classes. Since system lacks that knowledge, they are listed as classes. The missed out methods occur only if the tagger assigns a wrong tag to the word. Also the system perfectly identifies all the attributes, and actors with out any additional, missed or miss assignments.

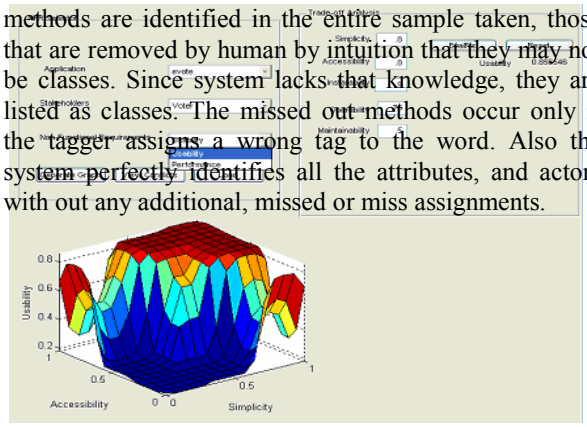

Fig. 5 Trade-off Analysis.

The acquired domain knowledge is visualized as ontology and stored in a database, which makes the retrieval of information quick and easy. We have tried to bring out a generic system in which the RDF is generated for all the SRS irrespective of the domain.

The non-functional requirements elicitation part also has been implemented using JAVA and a plug in to eclipse for reverse modelling. UML interface 'nsuml' is used to generate the state machines. This is used to synthesize the user behaviour of the system. The editor is created for generating both use case description and domain model description. The system is tested for the following domains ATM, Retailing system, Patient Monitoring System and E-voting system. The system will act according to the user's preference given in the non-functional requirement taxonomy.

The usecase and domain model makes use of XML editor to create data type definition (DTD) to store the model. The XML reader and writer are used to import and export the files for processing. The use case and domain model make use of structured text, which has been checked for the syntax. The wordnet is used as an interface to check for valid parts of speech.

Domain model makes use of UML diagram descriptions. The reverse engineering wizard is used to check usecase and domain maps. Workbench.jar, jface.jar, jdom.jar, runtime.jar are all used to implement the domain mapping and extracting. The prioritization of the non-functional requirement is mapped to goal questions, which are stored in Microsoft Access database. The conflicts are stored in a separate data file. The conflicts are resolved using inference engine created in matlab. The inference engine calculates the preference from the given user weights for each non-functional requirement.

The inference engine is designed to perform trade-off analysis for the non-functional requirements such as usability, performance, maintainability, security, correctness, authorization, reliability and availability.

5. Conclusions and Future Work

The paper presents an approach to develop ontology for the natural language requirement specification text by acquiring domain knowledge and an approach for deriving non-functional requirements by comparing usecase description with the domain model . Natural language processing techniques and set of rules are used to elicit domain knowledge from the requirement specification. The deficiencies in the tagger and the reference resolver present in the preprocessor can be overcome by building a knowledge base which can also improve the effectiveness of generation of the system elements. For non-functional requirements elicitation, the system takes a usecase written in a restricted form of natural language and generates a state model that integrates the behaviour specified by the usecase. The invariants and its initiators are captured from the usecase and domain model. The conflicting non-functional requirements are derived from the NFR taxonomy and they are prioritized using trade-off analysis. The system can be extended to automate the trade-off analysis by using an intelligent system.

1. (Abbot.R.J., 1983), "Program Design by informal English descriptions". *Communications of the ACM*, vol.26, 882 – 894.
2. (Akira Osada, Daigo Ozawa, Haruhiko Kaiya, Kenji Kaijiri, 2005), "Modelling Software Characteristics and Their Correlations in A Specific Domain by Comparing Existing Similar Systems," QSIC, pp. 215-222, Fifth International Conference on Quality Software (QSIC'05), 2005.
3. (Brill E., 1992): "A simple rule-based part-of-speech tagger". *Proceedings of Third ACL Conference on Applied Natural Language Processing*, Trento, Italy, 152-155
4. (Dong Liu, Kalaivani Subramaniam, Behrouz H. Far, Armin Eberlein, 2003), "Automating transition from use cases to class model", *MSc Thesis*, University of Calgary.
5. (Fensel D., 2000), "The semantic web and its languages", *IEEE Computer Society 15,* 6 (November /December), pp.67-73.
6. (Fensel D., Horrocks I., Harmelen F., Decker S., Erdmann M., Klein M. 2000), "OIL in a nutshell", *Proceedings of the European Knowledge Acquisition Conference,* (EKAW-2000), R. Dieng et al. (eds), Lecture Notes in Artificial Intelligenc, LNAI, Springer-Verlag, October.
7. (Fonseca, F. Egenhofer M., Agouris, P., Camara G. 2002), "Using Ontologies for Integrated Geographic Information Systems", Transactions in GIS, -(6):3
8. (T. R. Gruber, 1993), "A Translation Approach to Portable Ontology Specifications", Knowledge Acquisition, 5(2): 199--220, 1993.
9. (Ke Li, 2005), "Towards Semi-automation in Requirements Elicitation: mapping natural language and object-oriented concepts", *13th IEEE*

International Requirements Engineering Conference.

10. (Liwu Li, 1999), "A semi-automatic approach to translating use cases to sequence diagrams", *Proceedings of Technology of Object-Oriented Languages and Systems*, July, IEEE CS Press, 184 – 193
11. (María Auxilio Medina Nieto, 2003), "An Overview of Ontologies", Technical report, Center for Research in Information and Automation Technologies Interactive and Cooperative Technologies Lab Universidad De Las Américas Puebla – México.
12. (Mitkov. R, 1998), "Robust pronoun resolution with limited knowledge", Proceedings of the 18.th International Conference on Computational Linguistics (COLING'98)/ACL'98",Montreal, Canada, 869-875.
13. (Noy F. N., McGuinness D. L. 2001), Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March.
14. (Overmyer ScottP., Lavoie.B.Rambow, 2001), "Conceptual modelling through linguistic analysis using LIDA", *Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001*, Toronto.
15. (Russell S., Norvig P. 1995). *Artificial Intelligence: A Modern Approach.* Prentice Hall, Englewood Cliffs, NJ.
16. (Sawyer P., P Rayson, and R Garside, 2002), "REVERE: support for requirements synthesis from documents", *Information Systems Frontiers Journal*. Vol.4, 343 - 353.
17. (Starlab 2003), "Systems Technology and Applications Research Laboratory home page", Faculty of Sciences, Department of Computer Science, Vrije Universiteit Brussel. Available at: http://www.starlab.vub.ac.be/default.html
18. (Swartout B., Patil R., Knight K., Russ T. 1996), "Toward distributed use of large-scale ontologies", *Proceedings of the Tenth Knowledge Acquisition for Knowledge- Based Systems Workshop, Nove*mber 9-14, Banff, Alberta, Canada.
19. (Sylvain Delisle, Ken Barker, Ismaïl Biskri, 1999), "Object-Oriented Analysis: Getting Help from Robust Computational Linguistic Tools, in G. Friedl, H.C. Mayr (eds) *Application of Natural Language to Information Systems*, Oesterreichische Computer Gesellschaft, 167-172.
20. 1 Markus Nick, Klaus-Dieter Althoff, Carsten Tautz: Facilitating the Practical Evaluation of Organizational Memories Using the Goal-Question-Metric Technique. KAW'99 – Twelfth Workshop on Knowledge Acquisition, Modeling and Management 1999.
21. 2 Jane Cleland-Huang, Raffaella Settimi, Oussama BenKhadra, Eugenia Berezhanskaya, Selvia Christina: Goal-Centric Traceability for Managing Non-Functional Requirements. ACM ICSE'05 May 15–21, 2005
22. 3 Annie I. Anton, Colin Potts: The Use of Goals to Surface Requirements for Evolving Systems. 20th International Conference on Software Engineering (ICSE98), pages 157-166, April. 1998
23. 4 Luiz Marcio Cysneiros, and Julio Cesar Sampaio do Prado Leite: Nonfunctional Requirements: From Elicitation to Conceptual Models. IEEE Transactions On Software Engineering, Vol. 30, No. 5, May 2004
24. 5 Xiaoqing Frank Liu, John Yen: An Analytic Framework for Specifying and Analyzing Imprecise Requirements. Proceedings of 18th International Conference on Software Engineering (ICSE-18), Berlin, Germany, pp 60-69, March 25-30, 1996
25. 6 Martin Glinz: Rethinking the Notion of Non-Functional Requirements. Proceedings of the Third World Congress for Software Quality (3WCSQ 2005), Munich, Germany, Vol. II, 55-64
26. 7. Vittorio Cortellessa, Katerina Goseva-Popstojanova, Ajith R. Guedem, Ahmed Hassan, Rania Elnaggar, Walid Abdelmoez, Hany H. Ammar: Model-Based Performance Risk Analysis. IEEE Transactions On Software Engineering, Vol.31, No.1, January 2005
27. 8. Haruhiko Kaiya, Akira Osada, Kenji Kaijiri: Identifying Stakeholders and Their Preferences about NFR by Comparing Use Case Diagrams of Several Existing Systems. Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04) IEEE 2004
28. 9. John Yen, W. Amos Tiao, and Jianwen Yin: STAR: A Tool for Analyzing Imprecise Requirements. *Proceedings of 1998 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE '98)*, Anchorage, Alaska, pp. 863-868, May 4-9, 1998
29. 10. Andreas Gregoriades and Alistair Sutcliffe: Scenario-Based Assessment of Nonfunctional Requirements. IEEE Transactions On Software Engineering, Vol. 31, No. 5, May 2005.