ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)
Volume 2, Issue 4, April 2013

Model Based Test Case Generation From Natural Language Requirements And Inconsistency, Incompleteness Detection in Natural Language Using Model-Checking Approach

NEETHU GEORGE[#1], J.SELVAKUMAR[*2]

[#1]*PG Scholar ME-Software Engineering, SrRamakrishna Engineering College, Coimbatore Tamil Nadu, India*

[*2] *Associate Professor*
Sri Ramakrishna Engineering College Coimbatore, Tamil Nadu, India.

Abstract **-- Natural language (NL) is any language that arises in an unpremeditated fashion as the result of the innate facility for language possessed by the human intellect. A natural language is typically used for communication, and may be spoken, signed/written. Natural language (NL) is still widely used for developing software requirements specifications or other artifacts created for documenting requirements. However, natural language deliverables suffer from ambiguity, inconsistency and incompleteness. This work presents a methodology that produces model based test cases considering natural language requirements. Natural language requirements are converted in to state chart models and test cases are generated from state chart models. Inconsistency is a major problem that permeates all aspects of software development. Inconsistency occurs when a specification contains conflicting, contradictory description of the expected behavior of the system to be built or of its domain. Incompleteness contributes to one of the very serious problems that are present in software specifications. Existence of defects such as incompleteness certainly generates a source code that doesn't meet the undisclosed goals of the customers resulting in the generation of incoherent system and acceptance test cases. This paper proposes a methodology for dealing with defects such as incompleteness and inconsistency in natural language requirements deliverables. Model checking combined with k-permutations of n values of variables and specification patterns were used to detect incompleteness in software specifications. A method using both theorem-proving and model-checking techniques were used for automatically discovering inconsistencies in the requirements.**

Index Terms—**Inconsistency, Incompleteness, Model based testing, Model-checking, Natural language.**

I. INTRODUCTION

Various reports of serious problems in systems that have occurred due to defects in software have drawn attention, for some time already, from academicians and industry professionals.These researchers and practitioners have thus insisted on the use of standards, methodologies, techniques for developing software to try to ensure that, in the end, high-quality software is produced. Verification and validation is one of the pillars to ensure that software products have high quality. Verification and validation encompass a wide array of activities including formal technical reviews, inspection and all sorts of testing. Thus, testing a software product is only a facet to get quality. Model based testing has drawn attention in both industrial and academic areas in which several different types of software models have been used in order to guide test case generation. Model based testing has different interpretations. According to Mathur ," Model based testing occurs when the requirements are formally specified,using one/more mathematical/graphical notations such as Z, state charts and tests are generated using the formal specifications". Model based testing can also be considered as a test strategy in which test cases are derived entirely/partly from a model that describes functionality, safety and performance of software. Formal methods commonly used for system and acceptance model based test case generation are: FSM and Z .Complex software products present parallelism and hierarchy that are difficult to be represented using FSMs. This necessitates the need for considering higher level techniques as state charts. State charts have been introduced as a visual formalism for specifying the behaviour of complex reactive systems by extending the classical state transition diagram in several ways, while retaining and enhancing, their visual appeal.

Natural language processing (NLP) is a field of computer science dealing with the problem of computers to process and understand human languages.NLP may be applied in spelling correction, grammar checking, search engines, information retrieval, and speech recognition and so on. Field of NLP is vastly supported by several concepts like finite automata, part of speech (POS) tagging, word sense disambiguation (WSD), and first order predicate calculus and so on. Lexical category or POS is a linguistic category of words, which is generally defined by the syntactic or morphological behaviour of the lexical item in question. Common linguistic categories include noun and verb, among others. Word Sense Disambiguation is the ability to identify the meaning of words in context in a computational manner. A word can have different meanings even in the same lexical category. For instance, consider the two examples of the distinct senses that exist for the word "bass". The word bass could be (i) a type of fish, (ii) tones of low frequency and the sentences (i) I went fishing for some sea bass and (ii) the bass line of the song is too weak. To a human it is obvious that the first sentence is using the word "bass", as in the former sense above and in the second sentence, the word "bass" is being used as in the latter sense below. Developing algorithm to replicate this human ability can often be a difficult task ,as is further exemplified by the implicit equivocation between " bass"(sound) and " bass" (musical instrument).Thus, WSD is used to identify the correct sense in certain context. WordNet is the most commonly used lexicon of English for WSD. WordNet is an electronic lexical database that groups English words into sets of synonyms called synsets that provide short, general definitions and records the various semantic relations between these synonym sets. WordNet distinguishes between nouns, verbs, adjectives and adverbs as they follow different grammatical rules. Every synset contain a group of synonymous words or collocations. Different senses of a word are in different synsets. Most synonym sets are connected to other synsets via a number of semantic relations. WordNet also provides the polysemy count of a word i.e.; the number of synsets that contain the word. Inspite of all these, natural language requirements suffer from inconsistency, incompleteness. Model – Checking approach has been proposed for dealing with these issues.

In software and hardware design of systems, more time and effort are spent on verification than on construction. Techniques are sought to reduce and ease the verification efforts while increasing their coverage. Formal methods offer a large potential to obtain an early integration of verification in the design process, to provide more effective verification technique, and to reduce the verification time. Formal methods are actually treated as applied mathematics for modeling and analyzing systems. They concentrate on establishing system correctness with mathematical rigor. Research in formal methods has led to the development of some very promising verification techniques that facilitate the early detection of defects.

Model based verification techniques are based on models describing the possible system behavior in a mathematically precise and unambiguous manner. It turns out that prior to any form of verification; the accurate modeling of systems often leads to the discovery of incompleteness, ambiguities, and inconsistencies in informal system specifications.

Model-Checking is one of the prominent formal verification techniques for accessing the functional properties of systems. For this, a model of the system and a desired property is taken into consideration for systematically checking whether or not the given model satisfies this property. Properties like deadlock, invariants and request-responses can be checked. Model-checking is an automated technique. Given a finite-state model of a system and a formal property, a model-checker systematically checks whether this property holds for (a given state in) that model. For this, a model checker is first initialized by appropriately setting the various options and directives that may be used to carry out the exhaustive verification. Subsequently, the actual model-checking takes place. This is basically an algorithmic approach in which the validity of the property under consideration is checked in all states of the system model.

Model-checking can be considered as an automated technique to check the absence of errors. This technique explores all possible system states in a brute-force manner. Even the subtle errors that remain undiscovered can potentially be revealed using model-checking. It is a general technique applied in areas of hardware verification and software engineering.

Model-checking process evolves through the following three phases: modeling phase (system under consideration is modeled using the model description language), running phase (validity of the property in the system model is checked by running the model checker) and analysis phase (checks whether this property is satisfied/not).

Model-checking approach has a number of benefits. It is a general verification approach and is applicable to a wide range of applications. Using this approach, properties can be checked individually. One of the unique features of model-checking principle is its sound mathematical underpinning. The use of model-checking requires neither a high degree of user interaction nor a high degree of expertise. It can be easily integrated in existing development life cycles. Model-checking provides diagnostic information in case a property is invalidated thus helping in debugging process. Model-checking is an effective technique to expose errors.

One of the main classes of defects in requirement specifications is inconsistency. Inconsistency occurs when a specification contains conflicting, contradictory descriptions of the expected behavior of the system to be built or of its domain. Conflicting descriptions may arise as a result of conflicting goals or as a consequence of un-coordinated changes introduced in the specification during the usual evolution of the requirements.

Inconsistency permeates all aspects of software development making it possible to design and implement a system that respects its specification. Inconsistency in software specifications can be treated in two ways. First way ensures that it is not present in the software at all times i.e., it can be treated as an error, that needs to be corrected before further activities take place. Second way is to consider that inconsistency may be tolerated and resolved at later stages. It can be effectively managed by introducing a certain degree of formalism. Explicit inconsistencies in requirements can be easily identified while the identification of implicit inconsistencies requires the usage of formal specifications.

Incompleteness represents one of the serious problems in software specifications. Software specifications are created early within the software development lifecycles, thus causing their defects to affect the next software artifacts, including the source code to be developed. For instance, test designers take in to account software requirements specifications and related artifacts to create software design documents. Existence of defects such as incompleteness almost certainly generates source code that doesn’t meet the undisclosed goals of the customer resulting in the generation of incoherent system and acceptance test cases. A software specification is complete if it possesses the following qualities: (i) Inclusion of all requirements.

(ii) Definition of the software to all realizable classes of input data in all realizable classes of situations.

If a software specification doesn't satisfy any aspect of these two qualities, then the specification is considered to be incomplete.

II. RELATED WORK

A lot of research has been carried out in this field. Gervazi and Zowghi (2005) proposed a formal framework for identifying, analyzing and managing inconsistency in natural language requirements derived from multiple stakeholders. In this article, a particular inconsistency namely logical contradiction (any situation in which some fact α and its negation ¬α can be simultaneously derived from the same specification) was concentrated. Christel Baier and Joost–Pieter Katoen published a book on "Principles of Model Checking". Gervazi also proposed a methodology for the lightweight validation of natural language requirements. This article treats validation as a decision problem. In the field of requirement engineering, measuring inconsistency is crucial to effective inconsistency management. A practical measure considers both the degree and significance of inconsistency in specifications. This was realized by Kedian Mu, Zhi Jin, Ruqian Lu and Weiru Liu using the priority -based scoring vector ,which integrates the measure of the degree of inconsistency with the measure of the significance of inconsistency.
Lot of methodologies using dynamic ontologies was proposed for checking the inconsistencies in natural language requirements.

III. METHODOLOGY

The first activity in the proposed methodology for model based test case generation is the definition of a dictionary by the user/test designer. Dictionary defines the application domain. Dictionary contains the names defining mainly the names of states of the model, a function from input event set to output event set and a semantic translation model describing the hierarchy in the state chart model. After the definition of the dictionary, scenarios are identified. A scenario basically describes the interaction between the user and the implementation under test (IUT).Scenarios are identified and defined according to the define scenarios activity of the methodology. The first task serves to obtain the basic elements that enable interaction with the IUT. Simple scenarios are identified based on the core elements that enable interaction with the IUT. Factors and levels are assigned to these scenarios. Strength is assigned to each scenario using the formula:

Strength=#factors-1 (1)

Normal scenarios are identified based on the strength defined. For each normal scenario, new factors and levels are identified and a priority factor is assigned to each scenario. Once the scenarios are identified, the next major step deals with the generation of the appropriate model. The generate model activity is composed of two sub-activities. These sub-activities include: (i) generation of BSAO tuples and (ii) translation from BSAO tuples into behavioural model. The first task in the generation of model refers to the generation of behavior- subject-action-object (BSAO) 4-tuples.Actually, after the user has selected and inserted the natural language requirements that characterize a scenario; the tool combines all such requirements into a file. This file is input to the Stanford POS tagger which assigns the pos tag of each word. Besides the dictionary, the methodology takes as input all the words in the file that contains the set of natural language requirement. Some additional sets were predefined in order to help the process of BSAO tuples derivation. They are:

(i) useful words: This set contains POS tags that aid in the process of creating a key for the words within the requirements. This set defines the lexical categories of the running/useful words for the WSD algorithm. Common nouns (singular and plural), verbs, adjectives, adverbs were the chosen lexical categories.
(ii)candSubObj: This set contains POS tags that define the candidate words to be subjects and objects. The selected categories were common nouns (singular and plural), proper nouns and adjectives.
(iii) confirmSub: This set contains POS tags that confirm that a previous word identified as a subject is indeed a subject. The lexical categories included were modal verbs and verbs.
(iv)confirmAct: Pos tags that characterize an action were included under this. The selected lexical categories were verbs, adverbs and coordinating conjunctions

BSAO tuples generated are translated in to behavioural model. The generated model is a set of ordered quadruples where each quadruple represents a transition in the model. Each ordered quadruple is composed of: a source state, an input event set, an output event set and a destination state. The generated model is subjected to manual refinement to eliminate unnecessary states and transitions of the model and to fulfil the destination states of the transitions. After this, WSD refinement was made. This was achieved by manually searching synsets related to verbs in word net to find verb's senses. Abstract test cases are generated after the construction of the required model. Abstract test cases are translated into executable test cases to enable the effective execution of test cases. Having created the test cases for a single scenario, the test designer starts again selecting and inserting the NL requirements for the next scenario. This process is repeated until there are no more scenarios. Figure 1 below shows the methodology followed for detecting incompleteness in natural language deliverables.

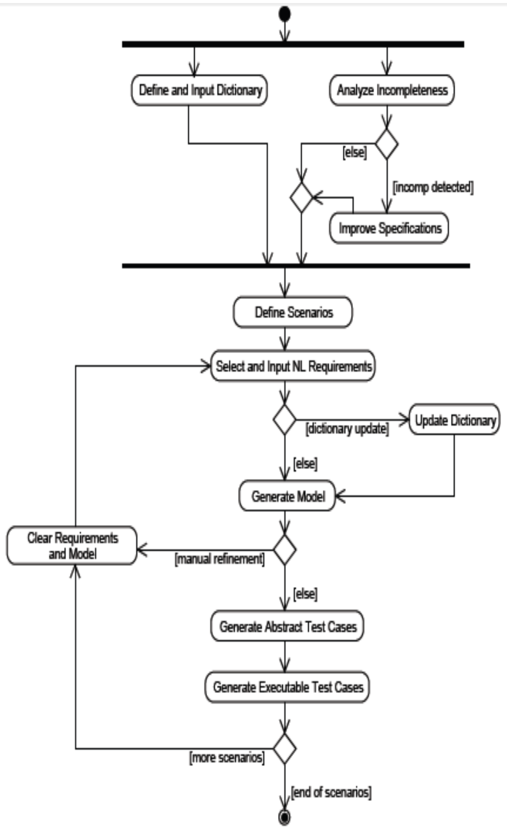


Fig1: Methodology for dealing with incompleteness

The most important activity to deal with the problem of incompleteness is analyzing incompleteness. It is by means of this activity that incompleteness defects are truly detected. Once incompleteness defects are detected, the quality of the assessed software specifications can be improved by completing the documents when necessary. Analyze incompleteness activity of the proposed methodology consists of studying software specifications, modeling the system, simulating the model and then applying model checking thus generating a report of detected incompleteness defects. An incompleteness defect is detected if there is a discrepancy between the satisfaction and non-satisfaction of the property by the model and the expected result. This methodology assumes that the formalization of properties and the generation/simulation of the model are done in parallel. First step is to choose a primary characteristic (attribute) obtained from software specifications that identify the main states of the software product. Some valprim is selected in order to generate finite state model. For each valprim, secondary characteristic (other attributes) of software specifications are selected. A finite model for each selected valprim is then generated and simulated to determine and convert possible defects which arise by translating the software specifications in to the model.Figure 2 below shows the incompleteness activity in detail.

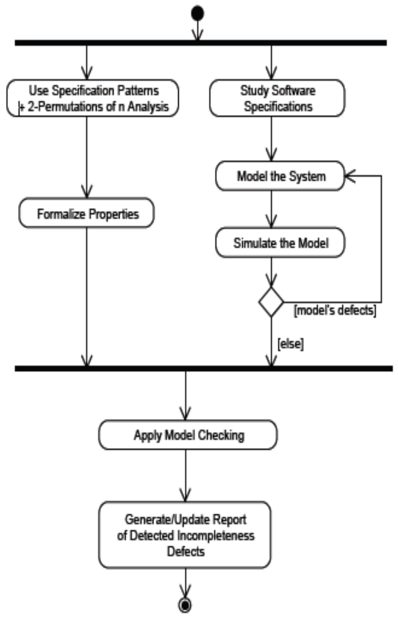

Use Specification Patterns + 2-Permutations of n Analysis
Study Software Specifications
Model the System
Formalize Properties
Simulate the Model
[model's defects]
[else]
Apply Model Checking
Generate/Update Report of Detected Incompleteness Defects

Fig 2: Analyze Incompleteness Activity in detail

Eliminating simple modeling defects before any form of thorough checking occurs may reduce the time -consuming verification effort. When there is no more remaining defect in the model and all properties are created, model checking is applied. Detected incompleteness defects are then reported in a certain type of document. Requirements introduced by a number of stakeholders are expressed as natural language sentences, and each stakeholder can state the requirement that is significant from his/her particular viewpoint. These natural language requirements are subjected to a series of transformations:
(i) Typographical adjustments [tokenization and synonym substitution],
(ii)Parts-of-speech tagging,
(iii)Parsing and transferring into a set of parse trees, and
(iv)Translation into a set of logic formulae. These logic formulae are then added to the specification, either as requirement (a property that is required to hold) or as constraints (a property that is desired not to hold) depending on the type of operation initially requested by the stakeholder. The various requirements operations are modeled as belief revision operations on the theory that represents the specification. As a result of these changes, inconsistencies may arise in the specification and inconsistencies arising in the requirement are checked by using model-checkers. If any inconsistency is found, a detailed report and the various alternative interpretations are presented to the stakeholders. A number of domains and functions are used to define all the transformations and operations in this approach. Figure 3 below shows the inconsistency detection activity.

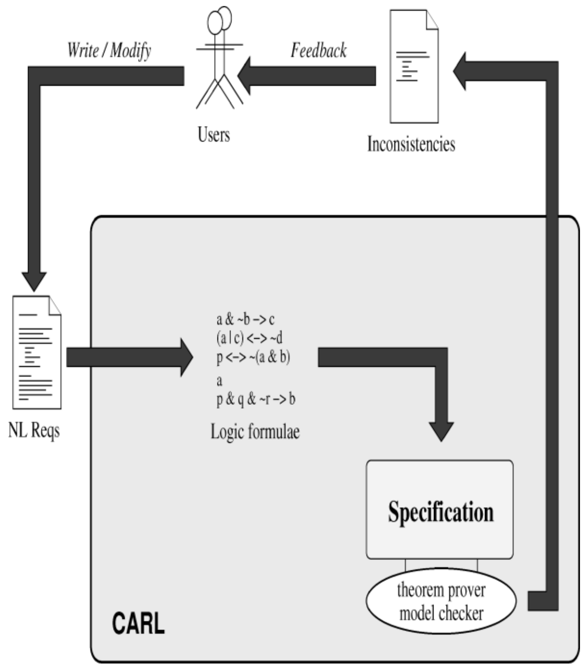


Fig 3: Inconsistency Detection Activity

A parsing process translates natural language sentences into logic formulae, and an unparsing process translates back those formulae into natural language sentences. The first step of the parsing step consists of a pre-processing stage devoted to typographical adjustments, tokenization and morphological analysis. During this step, multiple-word terms and domain-specific terms defined in a user glossary are converted into single tokens. All the words that do not appear in the user glossary are at this stage marked as potentially erroneous and the user is prompted to confirm their correctness. The preprocessed text is then parsed, according to small set of fuzzy parsing rules, to produce a parse tree corresponding to the original statement. Figure 4 shows how the requirements are preprocessed.

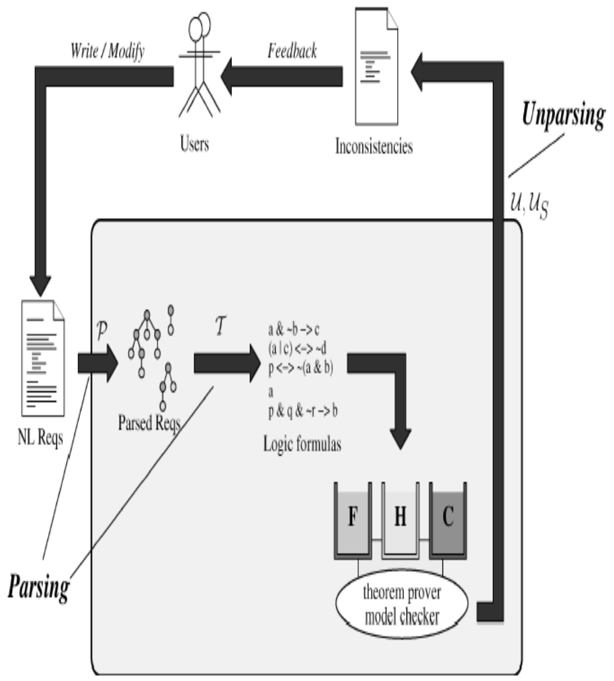
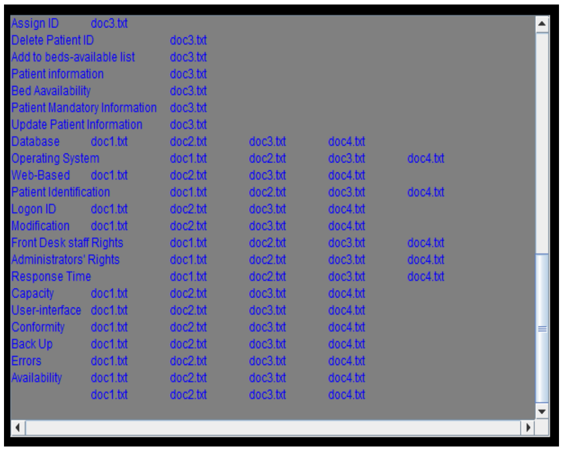


Fig 4: Detailed Inconsistency Detection

Finally, the parse tree is translated into predicate logic forms, and submitted for consistency checking and adding to the specification.

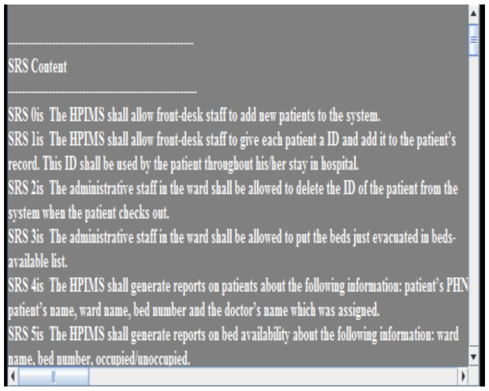
IV. RESULTS

This section deals with an analysis of the results obtained as a part of the implementation of the proposed methodology for model based test case generation.



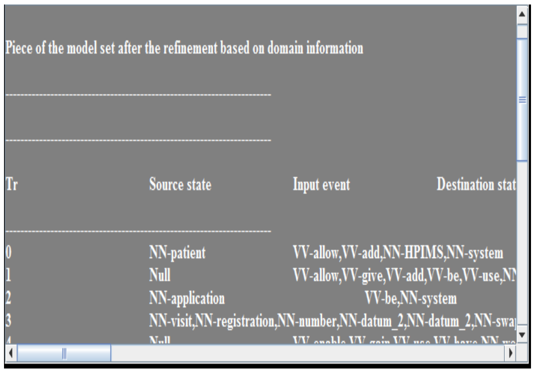
```
Assign ID       doc3.txt
Delete Patient ID               doc3.txt
Add to beds-available list      doc3.txt
Patient information             doc3.txt
Bed Aavailability               doc3.txt
Patient Mandatory Information   doc3.txt
Update Patient Information      doc3.txt
Database        doc1.txt        doc2.txt        doc3.txt        doc4.txt
Operating System                doc1.txt        doc2.txt        doc3.txt        doc4.txt
Web-Based       doc1.txt        doc2.txt        doc3.txt        doc4.txt
Patient Identification          doc1.txt        doc2.txt        doc3.txt        doc4.txt
Logon ID        doc1.txt        doc2.txt        doc3.txt        doc4.txt
Modification    doc1.txt        doc2.txt        doc3.txt        doc4.txt
Front Desk staff Rights         doc1.txt        doc2.txt        doc3.txt        doc4.txt
Administrators' Rights          doc1.txt        doc2.txt        doc3.txt        doc4.txt
Response Time                   doc1.txt        doc2.txt        doc3.txt        doc4.txt
Capacity        doc1.txt        doc2.txt        doc3.txt        doc4.txt
User-interface  doc1.txt        doc2.txt        doc3.txt        doc4.txt
Conformity      doc1.txt        doc2.txt        doc3.txt        doc4.txt
Back Up         doc1.txt        doc2.txt        doc3.txt        doc4.txt
Errors          doc1.txt        doc2.txt        doc3.txt        doc4.txt
Availability    doc1.txt        doc2.txt        doc3.txt        doc4.txt
                doc1.txt        doc2.txt        doc3.txt        doc4.txt
```

Fig 5: Preprocessing



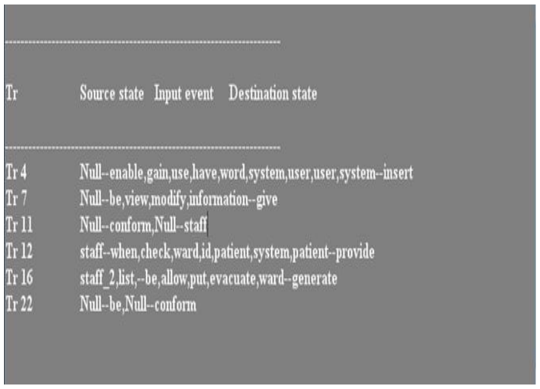
```
------------------------------------------------------
SRS Content
------------------------------------------------------
SRS 0is  The HPIMS shall allow front-desk staff to add new patients to the system.
SRS 1is  The HPIMS shall allow front-desk staff to give each patient a ID and add it to the patient's
record. This ID shall be used by the patient throughout his/her stay in hospital.
SRS 2is  The administrative staff in the ward shall be allowed to delete the ID of the patient from the
system when the patient checks out.
SRS 3is  The administrative staff in the ward shall be allowed to put the beds just evacuated in beds-
available list.
SRS 4is  The HPIMS shall generate reports on patients about the following information: patient's PHN
patient's name, ward name, bed number and the doctor's name which was assigned.
SRS 5is  The HPIMS shall generate reports on bed availability about the following information: ward
name, bed number, occupied/unoccupied.
```

Fig 6: BSAO Tuple result



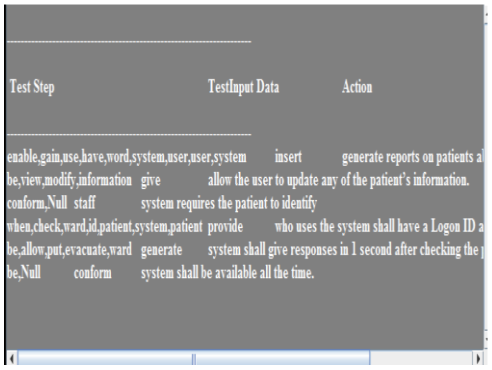
```
Piece of the model set after the refinement based on domain information

--------------------------------------------------------------------

--------------------------------------------------------------------

Tr                    Source state          Input event               Destination stat

--------------------------------------------------------------------
0                     NN-patient            VV-allow,VV-add,NN-HPIMS,NN-system
1                     Null                  VV-allow,VV-give,VV-add,VV-be,VV-use,NN
2                     NN-application                   VV-be,NN-system
3                     NN-visit,NN-registration,NN-number,NN-datum_2,NN-datum_2,NN-swa
```

Fig 7: Refined Tuple result



```
------------------------------------------------------------------

Tr        Source state  Input event  Destination state

------------------------------------------------------------------
Tr 4      Null--enable,gain,use,have,word,system,user,user,system--insert
Tr 7      Null--be,view,modify,information--give
Tr 11     Null--conform,Null--staff
Tr 12     staff--when,check,ward,id,patient,system,patient--provide
Tr 16     staff_2,list,--be,allow,put,evacuate,ward--generate
Tr 22     Null--be,Null--conform
```

Fig 8: Similarity Tuple reduction



```
----------------------------------------------------------------
Test Step                               TestInput Data          Action
----------------------------------------------------------------
enable,gain,use,have,word,system,user,user,system       insert          generate reports on patients a
be,view,modify,information   give              allow the user to update any of the patient's information.
conform,Null   staff              system requires the patient to identify
when,check,ward,id,patient,system,patient  provide          who uses the system shall have a Logon ID a
be,allow,put,evacuate,ward   generate          system shall give responses in 1 second after checking the
be,Null          conform          system shall be available all the time.
```

Fig 9: Test case generation

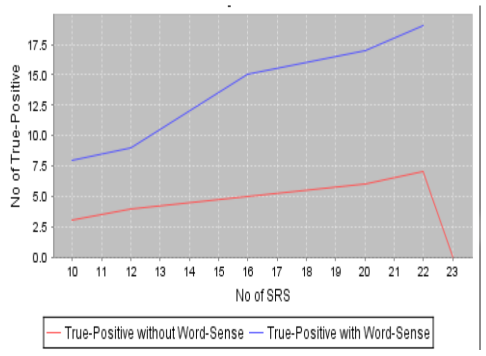


Fig 10: Test cases generated with WSD and without WSD: A comparison chart

REFERENCES

[1]Ambriola, V., & Gervasi, V: "On the systematic analysis of natural language requirements with CIRCE", Automated Software Engineering, 13(1), 107–167, 2006.

[2] Cristia, M., & Monetti, P:" Implementing and applying the Stocks–Carrington framework for model based testing", In K. Breitman, & A. Cavalcanti (Eds.), Formal methods and software engineering (LNCS, Vol. 5885, pp. 167–185),2009, Berlin/Heidelberg, Germany: Springer Applications, 38, pp.7277-7290.

[3]Gervasi, V:"The cico domain based parser", tech.rep.TR-01-25.Dipartimento di Informatica, University of Pisa, Pisa, Italy. (2001)

[4]Gervasi, V. And Nuseibeh, B:" Lightweight validation of natural language requirement Softw: Pract. Exper.32, 2 (Feb.), 113-133, 2002.

[5]Gervasi, V., & Zowghi, D:" Reasoning about inconsistencies in natural language requirements", ACM Transactions on Software Engineering and Methodology, 14(3), 277–330, 2005.

[6]Harel, D., Pnueli, A., Schmidt, J. P., & Sherman, R:" On the formal semantics of State charts (extended abstract)", In Proceedings of the 2nd IEEE symposium on logic in computer science (LICS), (pp. 54–64), 1987, Washington, DC: IEEE Computer Society

[7]Hunter,A. And Nuseibh, B:"Managing Inconsistent Specifications –Reasoning, Analysis and Action", Trans.Softw.Engg.Method.7.4 (oct.), 335-367, 1998.

[8]Santiago Junior ,V.A; Vijaykumar, N.L :"Generating model- based test case generation from natural language requirements for space application software", Software Quality Journal, 20:77-143,2012.

[9] Santiago Junior, V. A., Cristia, M., & Vijaykumar, N. L.:" Model-based test case generation using Statecharts and Z: A comparison and a combined approach", INPE, Sao Jose´dos Campos, http://www.urlib.net/sid.inpe.br/mtc-m19@80/2010/02.26.14.05,(INPE-16677-RPQ/850),2010.
system state transitions", Information and Software Technology, 51(2), 418–432, 2009.

[10]Sarma, M. & Mall, R.:"Automatic generation of test specifications for coverage of system state transitions", Information and Software Technology, 51(2), 418–432, 2009.

[11]Vincenzo Ambriola, Vincenzo Gervasi:"Processing natural language requirements", in proceedings of the Eighth Conference on Software Engineering Environments. IEEE Computer Society Press, 1997.

[12]Toutanova, K., Klein, D., Manning, C. D., & Singer, Y:" Feature-rich part-of-speech tagging with a cyclic dependency network", In Proceedings of the conference of the North American chapter of the association for computational linguistics on human language, 2003.

Neethu George obtained her B.Tech degree in Computer science from Cochin University of Science and Technology in 2010. She is currently pursuing her M.E degree in software engineering. Area of interest includes software engineering, natural language processing.

Dr.J.Selvakumar completed his Ph.D in Anna University Chennai in 2013, He completed his M.E(CSE) in Bharathiyar University in 2002 and B.E(CSE) in Madras University in 2001. He has around 40 publications in International Journals, International Conference & National Conference. His area of interest is Software Engineering.