SANAYOJAN: A Framework for Traceability Link Recovery between Use-Cases in Software Requirement Specification and Regulatory Documents

Student Name: RITIKA JAIN

IIIT-D-MTech-CS-DE-12-047
Jan 18,2014

Indraprastha Institute of Information Technology
New Delhi

Submitted in partial fulfillment of the requirements
for the Degree of M.Tech. in Computer Science,
with specialization in Data Engineering

Keywords: Mining Software Repositories, Requirements Engineering, Traceability Link Recovery, Text Mining, Natural Language Processing, Regulatory Compliance

Certificate

This is to certify that the thesis titled **"SANAYOJAN: A Framework for Traceability Link Recovery between Use-Cases in Software Requirement Specification and Regulatory Documents"** submitted by **Ritika Jain** for the partial fulfillment of the requirements for the degree of *Master of Technology* in *Computer Science & Engineering* is a record of the bonafide work carried out by her / him under my / our guidance and supervision in the Data Engineering group at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Prof. Ashish Sureka
Indraprastha Institute of Information Technology, New Delhi

Abstract

User requirement specification (URS) documents written in the form of free-form natural language text contain system use-case descriptions as one of the elements in the URS. For few application domains, some of the system use-cases in URS define services and functionality which needs to comply with regulations pertaining to the application domain. In this paper, we present a multi-step approach to automatically extract system use-cases from URS and construct traceability links between system-uses and appropriate regulations in the regulatory documents. We define lexicon-based, syntactic and semantic features to discriminate system use-cases from other elements in the URS. We investigate the application of five semantic similarity methods implemented in the SEMILAR semantic similarity toolkit to compute similarity between a given system use-case with regulations in a regulatory document. We conduct a series of experiments on real-world data obtained from software projects of a large global Information Technology (IT) services company to validate the proposed approach. Experimental results demonstrate effectiveness (accuracy of 83.3% for system use-case extraction and 72% for constructing traceability links) and limitations of the proposed approach.

Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Prof. Ashish Sureka for the continuous support in my M.tech Thesis and Research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Rahul Purandare, Vivek Balaraman, for their encouragement, insightful comments, and hard questions.

My sincere thanks also goes to Dr. Smita Ghaisas for offering me the summer internship opportunity in her group and leading me working on exciting project in Software Engineering.

I thank my fellow mates in IIIT, Delhi: Megha Mittal, Abhishek Bhola, for the stimulating discussions and moral boost up.

Last but not the least, I would like to thank my family: My parents for giving birth to me at the first place and supporting me spiritually throughout my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Motivation

Software applications and information systems providing services to the users and supporting business processes need to comply with the regulations related to the services and business processes supported by them [2] [6] [12] [10] [7] [8] [9] [11] [13]. For example, information systems in the health-care domain need to comply with Health Insurance Portability and Accountability Act (HIPAA[1]) and applications in certain financial domain need to comply with the Sarbanes-Oxley Act[2]. The need of software application compliance to laws and regulations requires eliciting and addressing law related functional and non-functional requirements and also maintaining traceability of specific laws with specific elements in the software artifact due to regulatory changes [2] [6] [12] [10] [7] [8] [9] [11] [13]. Identification of elements within a software to specific legal regulations and maintaining the traceability links (focus of the work presented in this paper) as the system evolves is a non-trivial problem in the context of large and complex software systems. Manual process of uncovering traceability links between software artifacts and regulatory documents is not scalable, is tedious and error-prone due to the large size and complexity of the software as well as the regulations. Automatic traceability link recovery (compliance checking between software artifacts and regulatory documents) also poses several technical challenges due to factors such as natural language text, terminology mismatches between software domain and legal domain and ensuring adaptability to regular amendments and revisions in regulatory documents. Compliance checking and verification and traceability link recovery between software artifacts and regulatory documents is an area that has attracted several researchers' attention (refer to Section 1.2 on related work). However, the problem is not fully solved and the work presented in his paper is motivated by the need to provide alternate solutions and a fresh perspective to the stated problem. The specific research aims of the work presented in this paper are the following:

1. To investigate lexicon-based, syntactic and special features for the purpose of extracting system use-cases (for the next-step of compliance checking) from a software requirement and design document.

2. To examine semantic-based textual similarity analysis techniques for linking elements in the software artifact with specific laws in the regulatory documents.

[1] http://www.hhs.gov/ocr/privacy/
[2] http://www.soxlaw.com/

3. To conduct an empirical study on real-world dataset obtained from an Information Technology (IT) service organization and validate the proposed model.

1.2 Related Work and Research Contributions

Extracting regulations and policies from free-form software artifacts and natural language text in software domain is an area that has attracted several researchers' attention. Automating the process of linking sentences in software documents with legal requirements and documents is an open research problem and we find few papers addressing the given problem area.

Bobkowska et al. analyze both lawyers' and software engineers' perspective and propose use of common information space between software engineers and lawyers during requirements engineering. The common information space will include basic knowledge of software engineering for Lawyers, basic knowledge of law for software engineers and basic knowledge of difference between these two. They propose collaboration of both stakeholders at the time of requirements elicitation [2].

Breaux et al. illustrate use of semantic parameterization combined with extended methodology to derive rights and obligations and to restate regulations text into restricted natural language statements (RNLS). They employ manual extraction techniques using RNLS patterns to separate the right or obligation phrase(s) from relevant constraint phrase(s). They identify conflicts between rights and obligations [3].

Huang et al. present use of probabilistic network models in computing similarity score regulations and requirements based upon the frequency and distribution of terms in the two text segments. Every term in requirements is assigned a probabilistic weight with respect to every term in regulations. They also propose use of machine learning approach - classification for improving traceability links regulations and requirements [4].

Massey et al. propose a methodology to evaluate legal compliance of security requirements using direct terminology mapping, requirements elaboration on the basis of terminology and then finding traceability links between regulations and legal text requirements. Terminology mapping includes finding associations between software terms and legal terms which provides a basic level of traceability links between them. During requirements disambiguation the existence of each term is analyzed in terms of its role and meaning. The focus of their study lies in security and privacy of software requirements and their compliance with HIPAA regulations [6].

Maxwell et al presents an approach to analyze existing requirements for regulatory compliance using queries to a production rule model through use of Prolog to validate existing requirements and identify new potential requirements. Terminology mappings are found between requirements and laws using these queries to production rule model and then preconditions are identified and grouped into sets. Potential areas of non-compliance are then identified querying the production rule model [7].

May et al describe sets of rules and language suitable to model the HIPAA privacy rules and a method to translate natural language text to these models. During translation, the difference between the terminologies is analyzed [8].

Seresth et al present use of Recursive Object Model (ROM) use to represent technical texts in software engineering documents. They suggest use of Expert-Comparable Contextual model to form the dictionary of words to be used in writing the requirements and use homogenized terminology to avoid inconsistencies [10].

Xiao et al. present use of NLP techniques and linguistic analysis including semantic and syntactic analyses which can be used to automatically extract instances of access control policies from NL documents. Their experiments and study is limited to access control policies only [13].

Ghaisas et al. present use of NLP techniques like syntactic and semantic analysis to extract rules from SRS documents using Ronald Ross' definition of rules and find rule intent patterns in sentences. These intent patterns are then matched in business process steps to check for the matching business process steps [5].

In context to existing work, the study presented in this paper makes the following novel contributions:

1. We propose a solution to extract system-use cases from free-form URS documents using lexicon-based, syntactic and spatial patterns or features.

2. We investigate the application of several state-of-the-art lexical and semantic based sentence similarity computation algorithms for traceability link recovery between system use-cases and regulations.

3. We conduct a series of experiments on real-world URS and regulatory documents and present performance evaluation analysis to demonstrate the effectiveness of the proposed approach.

Chapter 2

Solution Approach

Figure 2.1 presents a high level architecture of our proposed solution approach. The framework consists of three phases - (1) Manual analysis and visual inspection of URS documents to identify syntactic, spatial and lexical features and possible combination of features for system use case extraction. (2) Implementation of patterns identified in previous step to automate the task of system use case extraction. (3) Automated traceability links recovery between system use cases and the regulations using different textual similarity techniques and comparing the results of each.

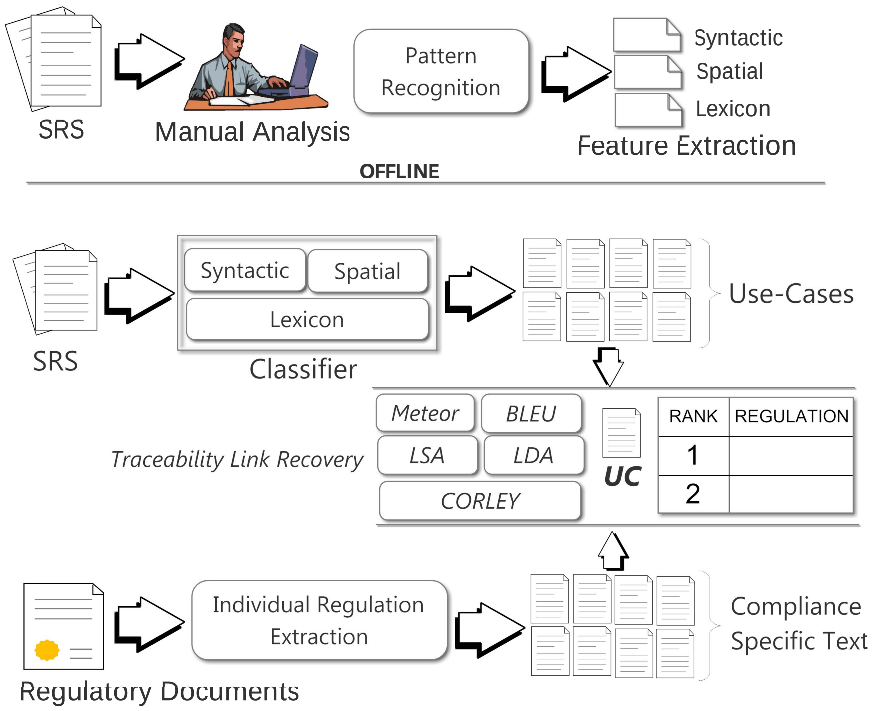


Figure 2.1: Research framework

2.1 Features Identification Using Manual Analysis

A user requirement specification (URS) document is an unstructured description of a software system written in natural language. Apart from system use cases, URS documents may contain various other elements of text such as title page, revision history, table of contents, definitions of terms, diagrams, purpose and scope of document. The URS document may also have various structural forms like tables, diagrams and enumerated text. A System Use-Case [1] is defined as a high level description of system processes where an actor performs an action to achieve a system's goal and ensures that the behavior of the system is what the user requires. System use cases are not labeled explicitly and are placed arbitrarily in a URS document surrounded with other elements of text hence making it difficult to extract system use cases from a URS document. The process of extracting system use cases from URS documents manually is time consuming and monotonous. To avoid this monotonicity and speed up the process of system use case extraction, we scan the URS documents manually to look for certain features which can be automated to extract system use cases.

During manual analysis, we visually inspect the URS documents and identify certain features which can distinguish a system use case from a non system use case like syntactic, spatial and lexical features. For this purpose, we investigate a sampled set of 15 URS documents using stratified sampling technique which are representative of the complete dataset. This sampled dataset serves as the training set for our features' selection process. After multiple iterations of this document scanning, we come up with a list of features (shown in Table 2.1). These proposed features are not not domain depepndent hence generic enough to be implemented against URS of any domain and are efficient discriminators of system vs non-system use cases. These features are then tested against test dataset consisting of 54 URS documents for checking their individual accuracy and performance in system use case extraction.

2.2 Automated Extraction of System Use Cases

The second phase of our solution approach consists of implementation of features identified in previous phase to automate the task of extraction of system use cases. We implement each feature mentioned in Table 2.1 which are as follows:

2.2.1 Spatial Features

Based on our inspection of SRS documents, we notice that various elements of text can be organized in different structural representations in a URS document in order of space such as diagrammatic representation, tabular representation and enumerated text. Diagrammatically represented use cases are accompanied with the textual description as well and hence scannning of diagrams in SRS documents can be ignored without significant loss in system use case ex-

[1] http://alistair.cockburn.us/get/2465

Table 2.1: List of discriminatory features: syntactic, spatial and lexicon-based features

#	Abbr.	Feature Title	Feature Type	Remarks
F1	UCTI	Use Case Table Identification	Spatial	Table with 2-4 columns where one column has 3-5 terms followed by another having more than 9 terms.
F2	SUCI	Process Step Use Case Identification	Spatial-Syntactic	Identify descriptive text using level - 'normal' not heading & length is more than 9 terms + syntactic check using F8,9,10
F3	UCPI	Use Case Parts Identification	Spatial-Syntactic	Identify text in bullets using level of text and club sub-points + syntactic check using F8,9,10.
F4	SETD	Software Engineering Term Dictionary	Lexicon	Identify software engineering terms like "Actor", "User"
F5	IDND	Insurance Domain Noun Verb Dictionary	Lexicon	Identify insurance domain noun action keywords like "Insurer", "Claim"
F6	DAAD	Domain Agnostic Action Verb Dictionary	Lexicon	Identify domain agnostic action verbs like "Submit", "Indicate"
F7	UIUI	User Interface Use Case Identification	Lexicon	Identify user interface specific keywords like "Click", "Display"
F8	CUCI	Conditional Use-Case Identification	Syntactic	(If/When/Where) Pre-condition (Then/There) Post-condition
F9	ETUI	Event-Triggered Use Case Identification	Syntactic	(Once/Before/After) Pre-Activity Post-Activity
F10	NAUI	Noun Action Use Case Identification	Spatial-Syntactic	Identify noun-verb patterns having more than 9 terms.

traction accuracy. Enumerated representation of text alone did not prove to be an efficient discriminator of SUC from a non-SUC text as not all numerated texts are system use cases and hence we propose use of enumerated text patterns in combination with other features rather than using it as an independent feature.

UCTI

A URS document may contain several Tables inserted at various places in the document. A System Use-Case (SUC) can be described both in the form of Tables or text which are not Tables such as enumerated list of bullet points. The Apache POI toolkit to extract information from Microsoft Word documents can be used to identify Tables from a word document. We define UCTI feature to classify if a given Table is a system use-case table or not. We study several Tables in the sample dataset and observe that SUC Tables have a certain structure which can be exploited to distinguish it from non-SUC Tables. For example, we observe that a SUC Table consists of $2 - 4$ columns in which $1 - 2$ column contains less than 5 terms of text (the System Use-Case ID and label) and another column containing more than 9 terms containing the system use-case description. We count the number of columns and the average number of term in the rows and label it as SUC or non-SUC. Figure 2.2 shows examples of 3 Tables. Table (a) in Figure 2.2 is a SUC Table whereas Table (b) and (c) is not a SUC Table. Table (b) of Figure 2.2 contains more than 4 columns and Table (c) contains 4 columns but does not contain a column with more than 9 terms whereas Table (a) contains 3 columns with 2 columns containing 2 terms and third containing more than 9 terms. UCTI uses apache POI toolkit to scan these tables and marks Table (a) as system use case and Table (b) & (c) as non system use case.

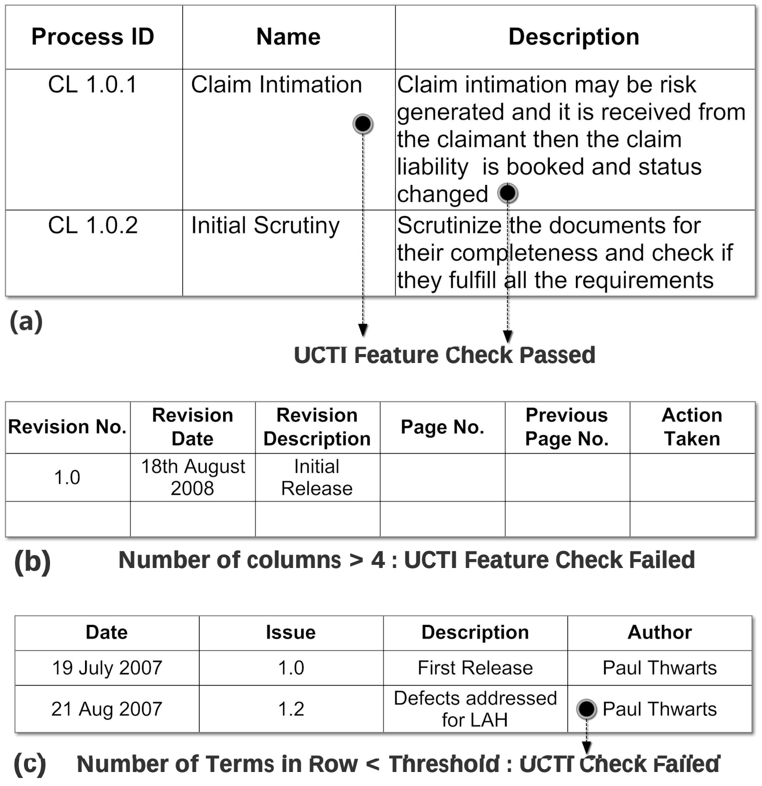
As shown in Algorithm 1, using Apache POI toolkit we examine every segment of text as individual paragraph and check if the segment is a tabular text or not. If text is tabular then we check for the number of columns in the table. If number of columns is between $2 - 4$ then we check for columns' content length and mark a table as system or non-system use case accordingly.

Algorithm 1 Extract system use-case tables

```
For every text segment
if segment is tabular then
    $ColCount \leftarrow count\ number\ of\ columns\ in\ table$
    if $ColCount \geq 2$ and $ColCount \leq 4$ then
        $i \leftarrow 0$
        while $i \leq ColCount$ do
            if $Col_i\ has\ 2 - 4\ terms\ and\ Col_{i+1}\ has\ more\ than\ 9\ terms$ then
                "$Table\ is\ a\ System\ Use\ Case\ Table$"
                $i \leftarrow i + 1$
            end if
        end while
    end if
else
    "$Table\ is\ not\ a\ System\ Use\ Case\ Table$"
end if
```

2.2.2 Lexicon-Based Features

Based on our manual analysis of the URS documents, we observe correlation between certain terms and insurance domain specific system use-case description. We believe presence of certain pre-defined terms in free-form text can be exploited as discriminatory features for the purpose of identifying and extracting system use-cases. We create four categories of lexicon: Use Case Specific Terms (USCT), Insurance-Domain Noun Terms (IDNT), Domain Agnostic Action Terms (DAAT) and User Interface Specific Terms (UIST). Table 2.2 shows sample terms from each of the four categories. As shown in Table 2.2, we categorize terms such as Process, Flow and Actor as USCT and terms such as Actuary, Agent and Broker as IDNT. Terms like screen, diplay, click let us identify user interface specific system use cases. We scanned the URS documents and learnt that lexicon-based feature alone cannot discriminate between system use-case and non-system use-case and needs to be used in conjunction with syntactic and spatial features to perform the classification task.



Process ID	Name	Description
CL 1.0.1	Claim Intimation	Claim intimation may be risk generated and it is received from the claimant then the claim liability is booked and status changed
CL 1.0.2	Initial Scrutiny	Scrutinize the documents for their completeness and check if they fulfill all the requirements

Revision No.	Revision Date	Revision Description	Page No.	Previous Page No.	Action Taken
1.0	18th August 2008	Initial Release			

Date	Issue	Description	Author
19 July 2007	1.0	First Release	Paul Thwarts
21 Aug 2007	1.2	Defects addressed for LAH	Paul Thwarts

(a)
UCTI Feature Check Passed
(b)
Number of columns > 4 : UCTI Feature Check Failed
(c)
Number of Terms in Row < Threshold : UCTI Check Failed

Figure 2.2: Example illustrating tabular form of system use case

Table 2.2: Sample terms from all the lexicons

#	USCT	IDNT	DAAT	UIST
1	Process	Actuary	Apply	Screen
2	Flow	Adjuster	Collect	Display
3	Define	Approver	Conduct	Enter
4	Actor	Agent	Examine	Select
5	Design	Broker	Generate	Choose
6	System	Cedant	Maintain	Option
7	Entity	Claimant	Modify	Show
8	Goal	Deceased	Reject	Monitor
9	Module	Handler	Resume	Interface
10	Scenario	Officer	Update	Button
11	Scope	Viator	Validate	Scroll
12	User	Member	Indicate	Icon
13	Event	Company	Establish	Cursor
14	Step	Beneficiary	Explain	Control
15	Case	Insured	Permit	Edit

2.2.3 Syntactic Features

Based on our manual inspection of the URS documents, we notice that the syntactic structure of the sentence can be exploited and encoded into a pattern for automatic system use-case extraction. A system use-case is an action performed by an actor and we believe that general regular-expression based patterns over part-of-speech tags can be written to automate the extraction of system use cases. We annotate each term in the textual document under analysis using a part-of-speech tagger and match the tagged output with a pre-defined syntactic pattern. Table 2.3 shows our proposed syntactic patterns along with annotated example from the dataset. In Table 2.3, VB denotes forms of verb, NN denotes forms of noun (actor), IN denotes use of prepositions, WRB and RB denotes family of adverbs, MD denotes modal verbs and "*" denotes any number of terms (0 or more) occurring in between the mentioned POS tags.

To check for POS patterns, we first remove stopwords from the text segment under observation and then use OpenNLP API for tagging the terms of sentence. This tagged sentence is then checked for predefined syntactic patterns for possible matches. Syntactic patterns defined are as follows:

CUCI

Conditional statements are statements where an action is performed on the basis of a condition and are represented by keywords such as *if-else, check whether, if and only if, unless, provided.* There exists a precondition on the basis of which a postcondition is to be done. The precondition and postcondition need not be necessarily in this order. The regular expressions proposed by us reflect this order interchange. Table 2.3 shows an example on the application of $CUCI$. As shown in Table 2.3 pattern $P2$ is applied to extract the sentence consisting of *if [IN]*, *is [VB]*, *then [RB]* and *sign [VB].*

Table 2.3: Illustrations of syntactic patterns in use cases

Feature	Patterns	Example
Conditional	P1: VB*RB*IN	***If*** *[IN] the policy is [VB] in trust* ***then*** *[RB] all the trustees must* ***sign*** *[VB] the death claim application.*
	P2: IN*VB*RB*VB	
	P3: VB*MD*VB	*Joint life policies are* ***accepted [VB] only [RB] if [IN]*** *both the signatures are available on the form.*
Event-Triggered	P4: MD*VB*IN	*Case must be referred at the early stage of the claim,* ***after [MD] notification [VB]*** *of death and* ***on [IN]*** *receipt of death certificate.*
	P5: WRB*NN*VBN	*Death claim is settled only* ***when [WRB]*** *direct* ***debit [NN]*** *is* ***suspended [VBN]*** *to avoid overpayment.*

ETUI

Event triggered statements are statements where multiple actions are performed in chronological order, one after other, and are represented by terms such as *once, after, before.* The patterns for identifying event triggered statements are as shown in Table 2.3. Table 2.3 shows an example sentence consisting of the term in bold annotated with the part-of-speech tag to illustrate the application of the pattern.

2.2.4 Feature Combination

This section comprises of certain features' combination which could not discriminate between a system use case and a non-system use case when used alone. The combination of features help in identification of system use cases written in bulleted text form, discriminating a heading from a descriptive text and applying regular expression patterns over descriptive texts. In these sort of system use cases, a single feature is not efficient enough in extraction. The features are as explained in the following sections.

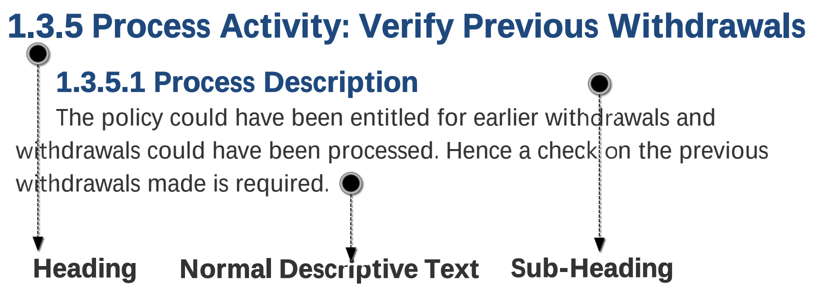

The policy could have been entitled for earlier withdrawals and withdrawals could have been processed. Hence a check on the previous withdrawals made is required.
1.3.5 Process Activity: Verify Previous Withdrawals
1.3.5.1 Process Description
Heading
Normal Descriptive Text
Sub-Heading

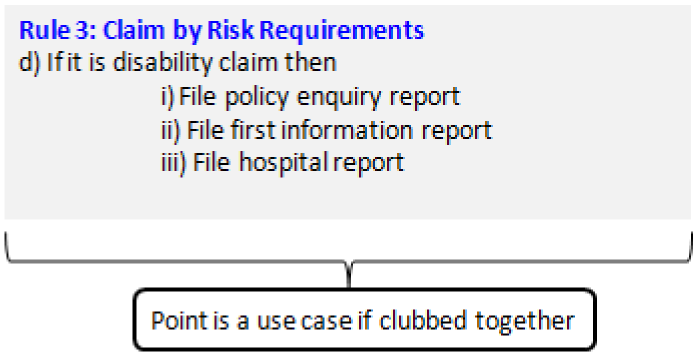
Figure 2.3: Example illustrating structural forms of text

SUCI

A system use case is a descriptive text and not a heading/sub-heading or any bulleted text segment - *SUCI* lets us differentiate between these segments of text. In SUCI, statements are first inspected on the basis of text segment lengths, bold, italics and level of text. The Apache POI toolkit lets us identify the level of text in terms of standard heading levels. A descriptive text is written in normal level of text in standard document writing process. If a text statement is a descriptive text (more than 9 terms) and without any bold/italics effect then statement is further inspected through the syntactic feature check F8, F9, F10. Figure 2.3 illustrates the difference between the structurally different text segments where there is a heading, a sub-heading and then a descriptive text. Our proposed approach scans through the text and skips checking headings and sub-headings and moves on to identify descriptive text using levels and effects of text. This descriptive text is then checked for existence of certain regular expression pattern, if any.

UCPI

A use case can also be expressed as a bulleted point description where each point in itself may fail to be makrked as a system use case but if the points are combined together then the complete segment is a system use case. UCPI detects a use case written in bulleted text form as shown in Figure 2.4. Bulleted text is recognized by its textual level in the document and the text segments at sub-level are joined together to form a single text statement. If after joining together the text segment contains more than 9 terms then the statement is further inspected through the syntactic feature check F8, F9, F10. As illustrated by Figure 2.4, the text is not a use case if considered line by line but is a valid SUC when bullet points are clubbed together. This is identified first by combining the points together and then the statement satisfies the patterns defined for conditional use case.



Rule 3: Claim by Risk Requirements
d) If it is disability claim then
i) File policy enquiry report
ii) File first information report
iii) File hospital report

Figure 2.4: Example illustrating use case written in parts

NAUI

A noun action system use case is where an actor performs an action and can be extracted using Noun-Verb patterns. The pattern for identifying noun-action statements is ** NN * VB ** where *NN* represents any form of noun and *VB* represents any form of verb, but not helping verbs. Before checking the text segment for existence of regular expressions, the stopwords are removed from it. To avoid false positives in extraction, we include a length check on the statement to be of more than 9 terms. For instance - ”*The claim value will need to be adjusted to take into account any amounts owing to the insurer in the form of loans or outstanding premiums, or to reimburse any overpayments received*”. This example sentence does not satisy our proposed syntactic patterns because this sentence is neither conditional based not event triggered. Hence, we need to define a feature which can capture the system use cases which are not extracted from previously proposed features.

2.3 Automated Traceability Links Recovery

The last phase of our solution approach is to find the traceability links between a legal system use case and the regulations it needs to comply with. For this, we compute the textual similarity or relation between extracted system use cases (taken as a query) and the regulations from the regulatory authority. We use SEMILAR API [1] to implement the similarity measurement techniques that assign each regulation a similarity score between 0 and 1 for a system use case. The regulations are then sorted on the basis of the similarity score assigned in decreasing order and top 5 regulations are extracted out from the regulations dataset. The top 5 regulations matches found are then manually validated for being relevant/irrelevant to the system use case query. As shown in Table 2.4, the system use case prompts the insured to fulfill the requirements in support of the claim. The top three regulations found by our approach in the regulations of IRDA are mentioned in Table 2.4. These regulations

Table 2.4: Instance of regulatory compliance of system use case

System Use Case: In order to ***process*** the deceased ***claim***, further ***requirements*** such as ***proof of death*** needs to be ***called*** for.
Relevant Regulations
A life insurance policy shall state the primary **documents** which are normally **required** to be submitted by a claimant in **support of a claim**.
A claim form submitted shall be **accompanied** by such other **documents** in **support of the claim** as the Board may **require**.
Every insurer shall inform and keep informed periodically the insured on the **requirements** to be fulfilled by the insured regarding **lodging of a claim**.

The comparison techniques used to find traceability links are as follows:

2.3.1 C1: METEOR

The Meteor evaluation metric scores regulations by aligning them to system use cases on the basis of exact, stemmed, synonymous, and paraphrase matches between words and phrases of text statements [1].

2.3.2 C2: Lexical

Lexical overlap technique assesses system use case and regulations and computes similarity score by finding the number of words they have in common (lexical overlap variations). The score of 1 (or 100%) indicates that the regulation has a total overlap with system use case, whereas 0 indicates there is no match between system use case and regulation [1].

2.3.3 C3: BLEU

BLEU metric compares n-gram of system use case with n-grams of regulations and counts the number of matches found in n-grams of these two text segments. The matches between SUC-Regulation pair are position independent. The number of n-gram matches found is divided by the number of n-grams in system use case query. The resulting score varies from 0 to 1 where 0 indicates no matches found and 1 indicates perfect match [1].

Table 2.5: Example illustrating regulatory compliance of system use case

System Use Case: The **policy holder has to fill** in all the particulars of the **switch form** and send it back to the **insurance company** in order to **switch policy**.	
Relevant Regulations	**Irrelevant Regulations**
The **policyholder shall fill in** the **portability form** along with the **proposal form** and submit the same to the **insurance company**.	A life insurance policy shall state the primary documents which are normally required to be submitted by a claimant in support of a claim.
On receipt of **portability form**, the **insurance company** shall address the existing insurance company seeking necessary details of medical history and claim history of the concerned **policyholder**.	The insurer shall furnish to the policyholder a written acknowledgment of having registered a nomination and may charge a fee not exceeding one rupee for registering such cancellation or change.
A **policy holder** desirous of **porting his policy** to another **insurance company** shall apply to such insurance company, at least 45 days before the premium renewal date of his/her existing policy.	

2.3.4 C4: CorleyMihalcea

CorleyMihalcea comparer finds similarity score between a term of system use case and terms of regulations using any word-to-word similarity measure and finds the term in regulation with maximum similarity score. This process is repeated for all the terms in system use case and an idf-weighted average is then computed over all the terms. The idf-weighted average value varies from 0 to 1 where 0 indicates no matches found and 1 indicates perfect match [1].

2.3.5 C5: LDA

Latent Dirichlet Allocation (LDA) is a probabilistic generative model where each word in a document is generated from a distribution over words that is specific to each topic. LDA based metric computes the dot product between vectors representing contribution to each term of the sentences in SUC-Regulation pair [1].

Table 2.5 shows an example of compliance of a system use case with regulations from IRDA Act, 1938. The example illustrates a system use case in which an insured wishes to switch the insurance policy. The Table 2.5 shows relevant regulations in the left column which need to be complied with the system use case whereas right column shows irrelevant regulations which can be ignored for this particular system use case. We use five similarity measurements techiques to take this system use case as a query and the regulatory document as the search base and finds out top 5 matching regulations with the query.

Chapter 3

Experimental DataSet

3.1 Dataset for system use case extraction

The experimental dataset consists of URS (consisting of system use-cases) documents and regulatory documents (consisting of regulations). We obtain 69 URS documents from insurance vertical customer application development projects of a large global IT service organization. The organization provided us with 16 regulatory documents pertaining to the application development projects. As shown in Table 3.1, the total number of system use-cases in the experimental dataset is 1518. However, only some use-cases pertain to insurance specific regulatory requirements. We notice that the regulatory documents containing legal regulations and government defined policies are lengthy and the average number of pages in a regulatory document is 50 and average number of regulations per regulatory document is 48. The average number of system use-cases per document is 22.

Table 3.1: Experimental dataset statistics

#	**Statistics**	**Count**
1	Num of SRS documents	69
2	Average num of pages in the SRS documents	40
3	Total num of system use-cases in dataset	1518
4	Average num of use cases per document	22
5	Num of Regulatory documents	16
6	Average num of pages in regulatory documents	50
7	Average num of regulations per document	48

3.2 Dataset for traceability links recovery

For regulatory compliance experiments, we divide our experimental dataset in three parts on the basis of different verticals of insurance domain in the regulatory documents. Table 3.2 shows the distribution of data in three data sets where first set constitutes of Claim Processing, Processing

Time, Policy Premium related queries (CPRQ), second set consists of Policy Registration, Policy Portability, Premium payment time restriction related queries (PRPQ) and third set of contains General Provisions of health insurance, Policy Renewals related queries (GHIQ). The sysem use cases extracted from previous step are taken as input queries in next this phase. The system use cases are chosen from three verticals for varied experiments and the regulations are taken from the regulatory data set available from IRDA (IRDA Act 1938).

Table 3.2: Experimental dataset for compliance detection

Abbr.	**Different Verticals**	**DataSet Count**
CPRQ	Claim Processing, Processing Time, Policy Premium	System Use Cases: 10 Regulations: 250
PRPQ	Policy Registration, Policy Portability, Premium payment time restriction	System Use Cases: 10 Regulations: 250
GHIQ	General Provisions of health insurance, Policy Renewals	System Use Cases: 06 Regulations: 250

Chapter 4

Empirical Analysis and Performance Evaluation

This chapter contains evaluation results of demonstration of our tools - System use case extractor and Traceability links finder.

4.1 System Use Case Extractor Results

Table 4.1: Individual features' accuracy results

#	**Feature**	**TP**	**TN**	**FP**	**FN**	**Acc.**
1	UCTI	4 (8%)	39 (83%)	1 (2%)	3 (7%)	0.91
2	SUCI	44 (54%)	36 (44%)	2 (2%)	0 (0%)	0.98
3	UCPI	15 (83%)	0 (0%)	0 (0%)	3 (17%)	0.83
4	SETD	26 (32%)	7 (9%)	0 (0%)	47 (59%)	0.41
5	IDND	68 (85%)	2 (3%)	5 (6%)	5 (6%)	0.88
6	DAAD	30 (37%)	3 (4%)	4 (5%)	43 (54%)	0.41
7	UIUI	6 (7%)	7 (9%)	0 (0%)	67 (84%)	0.16
8	CUCI	55 (69%)	5 (6%)	2 (3%)	18 (22%)	0.75
9	ETUI	67 (84%)	2 (2%)	5 (6%)	6 (8%)	0.86
10	NAUI	71 (89%)	1 (1%)	6 (7%)	2 (3%)	0.90

Table 4.1 shows demonstration results of system use case extractor for each feature when features are used to extract system use cases independent of each other. The accuracy results of spatial feature (F1), syntactic features (F8, F9) and combined features (F2, F3, F10) are above 75% which reveals their ability to extract system use cases whereas lexical features (F4, F6, F7) show relatively poor accuracy results in SUC extraction which indicates that lexicon based features cannot be used alone to extract system use cases and need to be used in combination with other features.

Our test data set consists of 54 documents from real projects in insurance domain. We run our tool Sanyojan as shown in Figure 4.1 (Combining all the features together) on the documents

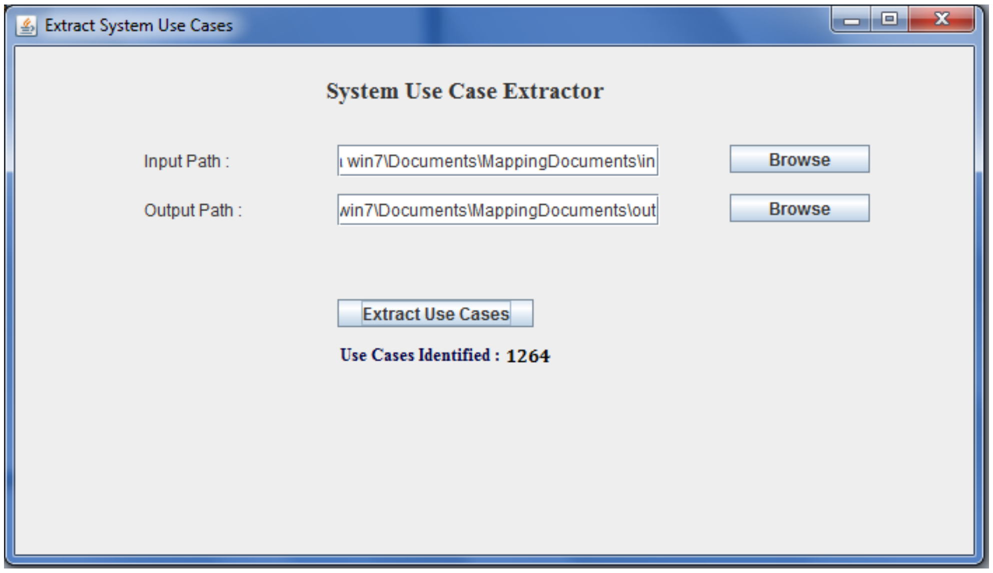

Extract System Use Cases
System Use Case Extractor
Input Path :
win7\Documents\MappingDocuments\in
Browse
Output Path :
win7\Documents\MappingDocuments\out
Browse
Extract Use Cases
Use Cases Identified : 1264

Figure 4.1: User Interface for System Use Case Extractor

and the retrieved use cases are stored in separate files. Total number of use cases extracted using our tool from the total documents is 1264. The resulting use cases are then validated manually and checked whether the extracted use case is a valid use case or not.

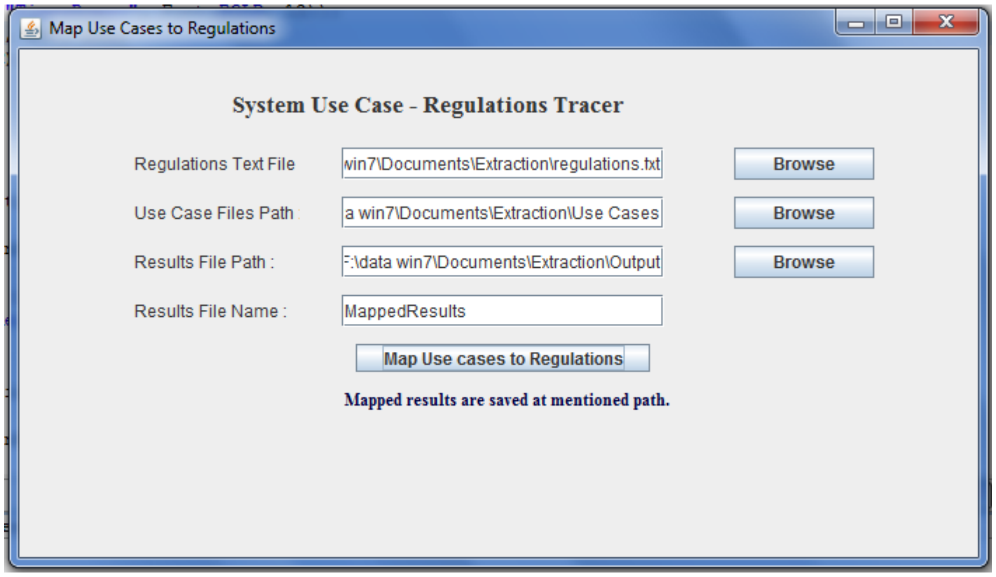
To evaluate the performance, we test our tool Sanyojan against test data set. After manual inspection of resulting system use cases we found that 96.3% (1218) of 1264 Use-Cases were true positives (manually verified to be a system use case) remaining were false positives (falsely marked as a system use case). 199 system use cases were misclassified as non-system use cases (false negatives) resulting in an accuracy of our tool to be 83.3%.

4.2 Traceability Links Recovery Results

For traceability links recovery, we demonstrate comparers as mentioned in previously on three experimental datasets (shown in Table 3.2). We run our tool Sanyojan (as shown in Figure 4.2)on the system use cases as queries and extract top 5 regulations for every use case. The regulations results are saved separately to validate results manually. For measuring the performance of different techniques we used a standard metric called Average Precision.

Average precision (AP) is a single-valued measure that reflects the performance of retrieval over all relevant documents by averaging the precision value obtained after each relevant document is retrieved. It rewards systems that retrieve relevant documents quickly (highly ranked). The formula for computing AP is as shown by Equation 4.1. Here, P(k) is the precision of document k and Rel(k) indicates relevance of document k. Rel(k) value of 0 means that kth system use case is not a valid use case whereas 1 means kth system use case is valid.

$$AP = \frac{\sum_{k=1}^{n} P(k) \times Rel(k)}{Number\ of\ relevant\ documents} \tag{4.1}$$


Map Use Cases to Regulations

System Use Case - Regulations Tracer

Regulations Text File: win7\Documents\Extraction\regulations.txt

Use Case Files Path: a win7\Documents\Extraction\Use Cases

Results File Path : F:\data win7\Documents\Extraction\Output

Results File Name : MappedResults

Mapped results are saved at mentioned path.

Figure 4.2: User Interface for traceability links recovery

Table 4.2: Average Precision scores of comparers for 10 use case queries of CPRQ

UC#	Meteor	Lexical	BLEU	CM	LDA
1	0.3	1	0.25	0.42	0.75
2	0.5	0.75	0.35	0.08	0.69
3	0.75	0.25	0.46	0.4	0.57
4	0.75	0.18	0.25	0.23	0.68
5	0.5	0.88	0.35	0.6	0.6
6	0.28	0.67	0.5	0.56	0.73
7	0.33	0.56	0.47	0.11	0.91
8	0.67	0.7	0.47	0.56	0.8
9	0.67	0.33	0.25	0.53	0.75
10	0.28	0.5	0.07	0.56	0.56

We compute the average precision for comparison techniques against three datasets and present the results through stacked bar chart as shown Figure 4.3 for data set CPRQ and Figure 4.4 for data set PRPQ and AP values for the same are as shown in table 4.2. The graph reveals that the average precision score of C5 i.e. LDA is consistent (equal height of colours) for all use cases queries whereas other comparers are query dependent and the precision scores are also low. The statistical analysis of AP values (Table 4.2) shows significantly efficient performance of LDA for finding traceability links between system use cases and regulations.

To compare the performance of these comparers, we compute another standard metric Mean Average Precision (MAP) over all 26 queries. Mean average precision for a set of queries is the mean of the average precision scores for each query as indicated by Equation 4.2. Here, AP(q) represents the Average Precision of comparer for query q and n represents number of queries.

$$MAP = \frac{\sum_{q=1}^{n} AP(q)}{n} \tag{4.2}$$

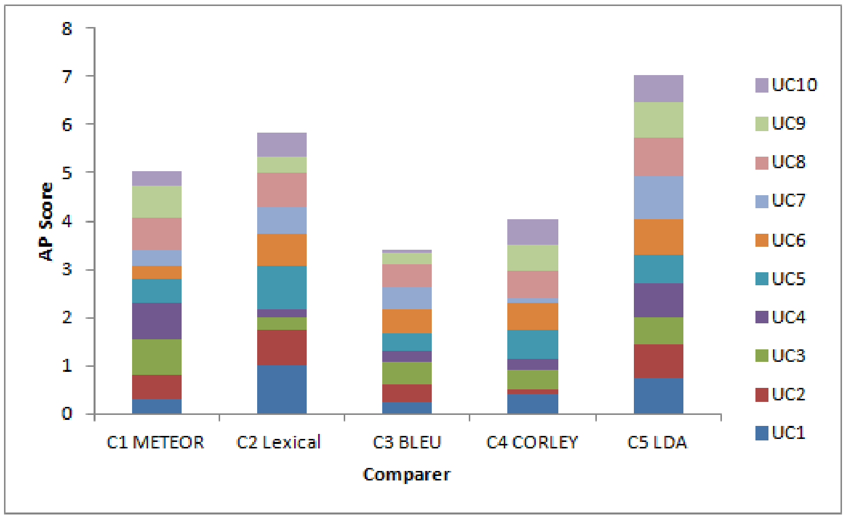


Figure 4.3: Stacked bar chart showing Average Precision scores of different comparers for CPRQ

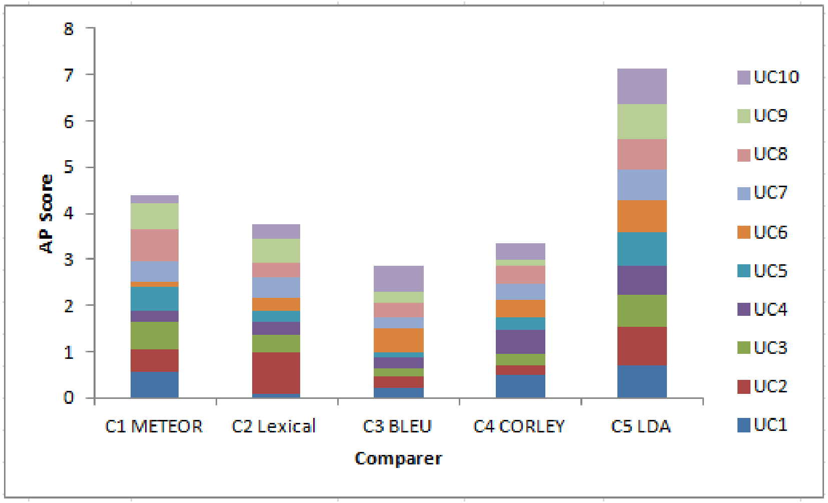


Figure 4.4: Stacked bar chart showing Average Precision scores of different comparers for PRPQ

Table 4.3 shows the MAP scores for different comparers which indicates that LDA is efficient enough and gives best results for finding traceability links between system use cases and regulations amongst all comparers.

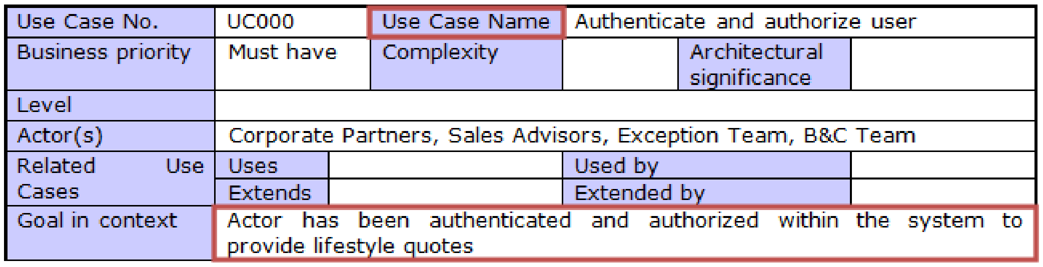
Table 4.3: Mean Average Precision scores of comparers

Meteor	**Lexical**	**BLEU**	**CM**	**LDA**
0.48	0.47	0.24	0.43	0.72

Chapter 5

Limitations

Besides beign efficient, our approach has few limitations as follows: One limitation to our solution approach is due to the document structure where the typical document writing standards like heading level, bullets etc. are not followed in writing SRS, in that case our features may fail to discriminate between SUC and non-SUC. Another limitation to our approach is from the inconsistent tabular structure of system use cases. Figure 5.1 shows an example of system use case table where the structure of table is not symmetric in terms of number of rows and columns. Our algorithm fails to apply spatial feature in such cases and misclassifies the table as a non-system use case table.



Use Case No.	UC000	Use Case Name	Authenticate and authorize user		
Business priority	Must have	Complexity		Architectural significance	
Level					
Actor(s)	Corporate Partners, Sales Advisors, Exception Team, B&C Team				
Related Use Cases	Uses		Used by		
	Extends		Extended by		
Goal in context	Actor has been authenticated and authorized within the system to provide lifestyle quotes				

Figure 5.1: Missed System Use Case Table Example

False positives for system use case extraction are due to statements like declaration statements written in the document. For instance - *V0.4 is a final document which is issued for Business Lead and Stakeholder sign off following a face to face walkthrough and final telecoms.* This statement satisfies Syntactic, Lexical and Spatial patterns and hence wrongly classified as system use case although it is a sentence for internal versioning information of the document.

5.1 Discussions and Future Work

We demonstrated our tool Sanyojan over complete dataset and the results show efficiency of our proposed approach with few limitations. The techniques proposed can still be modified in order to improve the extraction results. The lexicons can be enriched for the domain specific terminology and syntactic patterns of POS tags can be refined to avoid false positives. In addition to the features, semantic patterns can also be used to capture the structure ad writing style of system use cases in URS documents. We learnt that since URS documents are written in natural language, a particular system use-case can be written in multiple possible ways but to automate the task of extraction basic norms of document writing have to be followed. If basic norms are not followed then syntactic features will not be able to capture tabular text segments and bulleted text segments.

For traceability links recovery, we examine various techniques including n-gram comparison, lexical comparison and sematic analysis. We observe that semantic parameterization proves to be an efficient comparison technique for recovering regulations for a particular system use case but the results can be further improved using combination of comparison techniques and LSA. In future, we hope to refine the technique to improve results for system use case extraction and teaceability links recovery.

Chapter 6

Conclusions

Apart from above mentioned limitations, we observe an overall accuracy of 83.3% for the system use-case extraction step and conclude that a combination of lexicon, syntactic and spatial feature can be used to identify system use-cases from a free-form URS document (our domain of study is URS written in Microsoft Word belonging to Insurance domain projects). We achieve a Mean Average Precision (MAP) of 0.72 for LDA based semantic similarity comparer and 0.48 for Meteor comparer implemented in the SEMILAR toolkit. The results of our experiments demonstrate that an information retrieval based model in which system use-case can be formulated as a query and regulations as documents can be applied for traceability link recovery between system use-cases in URS and rules and regulations in regulatory documents.
The results retreived for the system use case extraction and traceability links recovery can further be improved by enriching lexicons and improving the regular expression patterns used in syntactic features.

Bibliography

[1] Banjade, R., Stefanescu, D., Niraula, N., Lintean, M., and Rus, V. Semilar api 1.0.

[2] Bobkowska, A., and Kowalska, M. On efficient collaboration between lawyers and software engineers when transforming legal regulations to law-related requirements. In *ICIT* (2010), pp. 105–109.

[3] Breaux, T., Vail, M., and Anton, A. Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations. In *RE* (2006), pp. 49–58.

[4] Cleland-Huang, J., Czauderna, A., Gibiec, M., and Emenecker, J. A machine learning approach for tracing regulatory codes to product specific requirements. In *SE, ACM/IEEE* (2010), vol. 1, pp. 155–164.

[5] Ghaisas, S., Motwani, M., and Anish, P. R. Detecting system use cases and validations from documents. In *Proc. ASE* (2013), IEEE, pp. 568–573.

[6] Massey, A. K., Otto, P. N., Hayward, L. J., and Anton, A. I. Evaluating existing security and privacy requirements for legal compliance. *Requir. Eng. 15*, 1 (Mar. 2010), 119–137.

[7] Maxwell, J. C., and Antón, A. I. Checking existing requirements for compliance with law using a production rule model. RELAW '09, pp. 1–6.

[8] May, M. J., Gunter, C. A., and Lee, I. Privacy apis: Access control techniques to analyze and verify legal privacy policies. CSFW '06, pp. 85–97.

[9] Otto, P., and Anton, A. Addressing legal requirements in requirements engineering. In *RE* (2007), pp. 5–14.

[10] Seresht, S. M., and Ormandjieva, O. Automated assessment of use cases elicitation from requirements text. In *WER* (2008).

[11] Slankas, J., and Williams, L. Classifying natural language sentences for policy. *Policies for Distributed Systems and Networks 0* (2012), 33–36.

[12] Slankas, J., and Williams, L. Automated extraction of non-functional requirements in available documentation. In *NaturaLiSE* (2013), pp. 9–16.

[13] Xiao, X., Paradkar, A., Thummalapenta, S., and Xie, T. Automated extraction of security policies from natural-language software documents. FSE '12, pp. 12:1–12:11.