An Approach to Mine Business Rule Intents from Domain-specific Documents

Abhidip Bhattacharyya, Pavan Kumar Chittimalli, Ravindra Naik
TRDDC, TCS Innovation Labs,
Pune, India
{abhidip.bhattacharyya1, pavan.chittimalli, rd.naik}@tcs.com

ABSTRACT

An enterprise system enables business by providing various services that are guided by set of well-defined processes, and adhere to certain business rules and constraints. The business rules are usually written using English in operating procedures, terms and conditions, and various other supporting documents. For implementing the business rules in a software system, or expressing them as UML use-case specifications, analysts manually interpret the documents, leading to potential discrepancies, ambiguities, and quality issues in the software system that can be resolved only after testing.

To minimize such errors, we propose a novel method to mine the documents automatically to extract the fundamental atomic facts in every sentence - called as business rule intents. We adopt dependency tree parser to parse the rule sentences and extract rule intents from them. Our experiments using few publicly available sample documents in the financial domain yielded very promising results, where rule intents extraction produced an average precision of 78% and recall of 80%.

1. INTRODUCTION

Business rules are the backbone of enterprise systems that provide various business services. These services are catered by executing a set of rules defined in a given process. The business rules eventually become core of the implementation within an IT system that automates the processes. The 'as-is' (implemented) rules can be found in the source code of the IT system, while the rules defined by business owners and policy makers are found in documents like manuals, user guides, requirements documents, terms and conditions, and do's & don'ts. In general, business rules are defined by the needs of the business and constrained by regulations - policy guidelines of the business and government authorities. Usually created by business analysts, the business rules typically reside in documents written in natural language. IT system owners and designers need to "understand" the business rules before they can be implemented in the IT system. To analyze the rules, extracting them from the documents and expressing them in a comprehensible manner is required. Given that the documents are usually very large, contain several rules with lot of noise, and are written in natural language, the manual extraction of business rules from these documents is an extremely cumbersome activity. The extracted rules if expressed yet again in natural language, are not always easy for humans to comprehend and analyze for inconsistencies. Therefore, extracting the specific business rules from various documents and expressing them in a comprehensible manner are both important concerns.

Semantics of Business Vocabulary and Rules^{TM} (`SBVR`™) [4] is a comprehensive standard for business rule representation by Object Management Group (`OMG`) [2]. `SBVR` is a Controlled Natural Language (`CNL`) and describes rules considering only a set of predefined business vocabularies. `SBVR` is largely based on first order predicate logic with equality, though it also supports restricted deontic and alethic modal logic, and set theory. Its natural language interface with formal logic makes it ideal for representing business rules. Looking like English, this formal representation can enable not only detecting inconsistencies, but other rule verifications too. With the `SBVR` representation, the problem is now reduced to extracting `SBVR` rules from various documents. The closest work is by Bajwa et al. [6], which generates `SBVR` format from a single line of English text. In this work, a user given English rule sentence (having a *subject*, *object*, and a *verb*) is converted into a `SBVR` rule automatically. To the best of our knowledge, there has been no further work in mining `SBVR` rules or rules in any other form. A typical document consisting of rule statements with lot of noise words makes it hard to comprehend and identify meaningful business rules. Another challenge is rule sentences with several clauses that make rule comprehension even more difficult. Moreover adherence to a particular format of the document or format of the output may put question on the reusability of the approach. Several techniques are proposed for extracting rules from legacy code [16, 26, 28, 8, 29, 12] and knowledge extraction from documents [19, 18, 10]. But there is little work [22] in the area of business rule extraction from documents.

In this paper, we propose a method for extracting rule intents from the rule sentences. Rule sentence is a sentence from the document that represent a fact or combination of facts. Consider an example rule sentence - "*No account is opened in anonymous or fictitious name*". A *rule intent* is an atomic fact or predicate present in a rule sentence. The *rule intents* are *isAnonymous*(`name`), *isFictitious*(`name`) and

ISEC '17, February 05 - 07, 2017, Jaipur, India
© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4856-0/17/02...$15.00
DOI: http://dx.doi.org/10.1145/3021460.3021470

$\neg doOpen$(`account, in name`). We propose *rule intents* as an intermediate step in eventually extracting business rules aligning with SBVR™ standard. After extracting rule intents, a combination of business vocabulary and the techniques proposed by Bajwa et al. [6] can be a potential way forward to create the SBVR rules automatically. Moreover an approach like P.Chittimalli and K.Anand [9] may be helpful to verify the consistency of rule intents represented using SBVR.

The main experiments consist of:

1. extract basic information as subject and object of a verb in the sentence.
2. extract relations among words connected by the prepositions.
3. build a compound rule intent where one of the earlier rule intents can be argument to another rule intent.
4. Splitting of rule intent based on argument.
5. negate the rule intent appropriately.

The paper is organized as follows: Section 2 describes the related work done in this area. Section 3 provides a motivating example to elaborate rule intents and significance of the relations between them. In section 4, we describe the detailed approach of identifying the rule intents. Section 5 illustrates the prototype tool and experimental results. Section 6 describes the future work and Section 7 describes the conclusion.

2. RELATED WORK

The text mining work focuses largely on applications towards predictive classification or populating a database or search index with extracted information. There appears little research specific to mining of business rules or creating their formal representations. The related work can be broadly classified into two categories 1) NLP techniques using shallow parsing 2) NLP techniques using finer level of models.

The first category of techniques [20, 30, 25] uses shallow parser [7]. R. Pandita et al. [20] proposed a method to generate code contracts from API document. The authors feed the data to the shallow parser after pre-processing. The sentences are converted into First Order Logic (FOL) formulae using predefined templates on shallow parser. Xaio et al. [30] proposed a method called `Text2policy` using shallow parsing to extract access rule and action steps from access control policies (ACP). The authors implement negative meaning implication (e.g. verb like *disallow*) and negative inference (word like *never, not*). A. Sinha et al. [25] proposed a Linguistic Analysis Engine to infer Use Case models from Use Case Description (UCD) based on shallow parser. Their engine is built on top of Unstructured Information Management Architecture (UIMA). They incorporate context annotators to allocate roles to the nouns occurring in the predicates. These roles are either ACTORs or SYSTEMs. Ultimately, they build process models by identifying the use case action sequence and by scanning the actions against known use case patterns. The three works described above use a domain dictionary to classify the verbs into some predefined classes depending on their semantic equivalence. Their techniques serve specific purposes and are centred on particular types of documents. They largely benefit from the structure of the documents.

The second category of related work goes much finer level than shallow parsing. The method proposed by S.Ghaisas et al. [14] focused on retrieving rule intents from requirement documents by matching them against patterns made up of sequence of Part Of Speech (POS) tags, key words and their repetitions denoted by wild characters like '*', '+' etc. They discovered 29 intents (such as chronology, activity, temporal check, threshold to denote the intention of *rule intent*) and 517 rule intent patterns. The authors used agglomerative clustering to place co-occurring rules together.

Other than these two classes of work, there are few other works that address the problem of extracting knowledge from documents. Colette Rolland et al. [24] proposed an approach to guide the construction of textual use case specifications from documents. They use the concept of case grammar introduced by Fillmore [13]. Their approach classifies the semantic patterns into Clause and Sentence semantic patterns to capture different surface structures of the sentences having same deep meaning. The approach further used set of rules and guidelines for converting the input into a use case. Atkinson et al. [5] used genetic algorithms for inferring hypothesis from scientific document written in natural language. H.Zhong et al. [31] proposed a method of inferring resource specifications from API document using machine learning. The knowledge extraction techniques described above use Hidden Markov Model (HMM) [23] based named entity recognition. Putrycz et al. [22] uses patterns and keywords derived from source code to create mappings between business rules from source code to documents.

All the above techniques basically focus on specific kind or class of documents and take advantage of the structure and format of the document. Most of the techniques make use of predefined templates. The first category of the related work uses shallow parser with predefined templates, making these techniques tightly coupled and dependent on the document structure. The second category of work of predefined templates using POS tag sequence makes it vulnerable to noise since the POS tag sequence can produce incorrect rule intents due to noisy word sequences. This brings us the problems such as eliminating noise, identifying rule sentences in the structured/unstructured documents, extracting rule intents from sentences, and extracting relationship among the rule intents.

```
 The increasing complexity and volume of fi-
nancial transactions necessitate that customers
do not have multiple identities within a bank,
across the banking system and across the finan-
cial system.

Existing individual customers were required
to be allotted UCIC by end-May 2013.
```

Table 1: Fragment of the KYC document

3. MOTIVATING EXAMPLE

In this section, we illustrate our approach using a simple but real-life example. Consider a fragment of the Know Your Customer (KYC) document given in Table 1 for understanding the overall objective of extracting business rule intent and the relations among them.

The marking shows a relevant rule sentence in green color. Noise is shown in color ~~red~~ . Noise elimination from the sentence though is not in scope of our current approach we put this as example to offer a broader view of our overall problem.

```
~~The increasing complexity and volume of
financial transactions necessitate that~~ cus-
tomers do not have multiple identities within a
bank, across the banking system and across the
financial system.

Existing individual customers were required to
be allotted UCIC by end-May 2013.
```

The task in our approach is to automatically extract relevant atomic facts from the *rule sentence*. The facts extracted from the rule sentences identified in the previous step are shown below.

f_1 : has(customer, identity, bank)
f_2 : has(customer, identity, banking system)
f_3 : has(customer, identity, financial system)
f_4 : $isMultiple$(identity)
f_5 : $isExisting$(customer)
f_6 : $isIndividual$(customer)
f_7 : $allocate$(customer, UCIC, May 2013)

Thus, the fragment of document in Table 1 has total 7 rule intents. The relations between rule intents extracted for the example are shown below.

$$Rule_1 : f_1 \wedge f_2 \wedge f_3 \rightarrow \neg f_4$$
$$Rule_2 : f_5 \wedge f_6 \rightarrow f_7$$

Table 2: The relations between rule intents for KYC document

Subsequently, the extracted rules (rule intents and relations between them) will be converted to SBVR models, which is a machine manipulatable format, to perform analyses for verification & validation [9]. This representation serves as (one of the) unambiguous functional specification for multiple purposes like traceability to rules in IT system, alignment with standards, and serve as critical part of the formal specification of the business system. In this paper we discuss the procedure for extracting the rule intents.

4. APPROACH

In this section, we first introduce our approach, and describe each phase of the approach in later parts of the section.

The block diagram in Figure 1 illustrates our approach. Rule sentences are parsed by natural language parser to produce the dependency trees. Our approach for rule intent extraction takes this dependency tree as input and provides possible rule intents as output. The extraction of rule intents from dependency tree is done in four stages. In first stage, the subject and object for each rule is detected. In second stage, the prepositional arguments are detected. In third stage, the objective arguments are extracted with the help of rule intents detected in the first stage. In fourth stage, a negation of a rule intent is handled. In this stage, a rule intent may be split into multiple rule intents based on argument type and count.

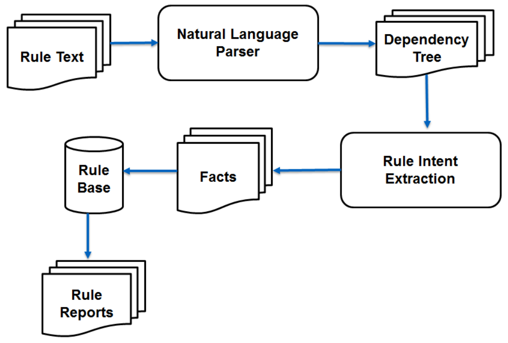


Figure 1: Block diagram of our approach.

4.1 Rule Intent Extraction

The objective of the Rule Intent Extraction (RIE) is to extract the rule intents from the rule sentences. Pattern/template based rule intent extraction may bind the approach with structure of the document. Hence we propose heuristic rules defined on the *dependency tree* of the rule sentence. A *dependency tree* captures the textual relation shared by the words in that particular rule sentence. To create the dependency tree, we used Stanford Dependency parser [11].

In this subsection, we use following four rule sentences as examples to illustrate the rules of RIE.

(rs_1) No account is opened in anonymous or fictitious name.

(rs_2) If the ordering bank fails to furnish information on the remitter, the beneficiary bank should consider restricting or even terminating its business relationship with the ordering bank.

(rs_3) In case the address mentioned as per 'proof of address' undergoes a change, fresh proof of address may be submitted to the branch within a period of six months.

(rs_4) Bank should verify identity and the address of the customer.

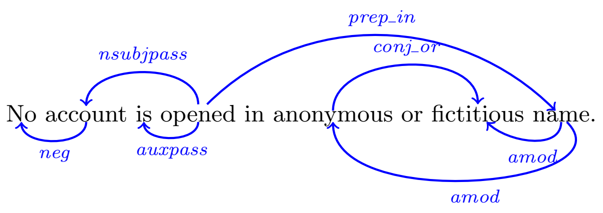


Figure 2: Dependency tree of rs_1

Consider the rule sentence rs_1 for the illustration of dependency tree. The corresponding dependency tree is shown in Figure 2 using labelled edges. The dependency between '*opened*' and '*account*' is '*nsubjpass*'. The edge labelled '*nsubjpass*' denotes the '*passive nominal subject*' dependency - a passive nominal subject is a noun phrase which is the syntactic subject of a passive clause. The dependency relation is captured by Stanford Parser as - *nsubjpass*(opened, account). The word '*account*' is dependent on '*opened*'(*governor*) being the subject in passive form. A detailed description of all the relations in dependency tree can be found in Stanford Dependency manual [11].

Rule Sentence	Rule Intents
rs_1	(ri_1) *isAnonymous*(`name`)
	(ri_2) *isFictitious*(`name`)
	(ri_3) ¬*doOpen*(`account`, `in name`)
rs_2	(ri_4) *isOrdering*(`bank`)
	(ri_5) *furnish*(`information`, `remitter`)
	(ri_6) *fails*(`bank`, *furnish*(`information`, `remitter`))
	(ri_7) *terminating*(`relationship`[`business`],`with bank`)
	(ri_8) *restricting*(`relationship`[`business`],`with bank`)
	(ri_9) *consider*(`bank`[`beneficiary`],*terminating* (`relationship` [`business`], `with bank` `relationship` [`business`], `with bank`))
rs_3	(ri_{10}) *isFresh*(`proof`)
	(ri_{11}) *isMentioned*(`address`, `per proof`)
	(ri_{12}) *undergoes*(`address`, `change`)
	(ri_{13}) *is-of*(`proof`, `address`)
	(ri_{14}) *doSubmit*(`proof`, `branch`)
	(ri_{15}) *is-of*(`period`, `months`([`six`]))
rs_4	(ri_{16}) *is-of*(`customer`, `address`)
	(ri_{17}) *verify*(`bank`, `address`)
	(ri_{18}) *verify*(`bank`, `identity`)

Table 3: The rule intents for the rule sentences rs_1, rs_2, and rs_3

Our approach uniquely contributes by defining a set of rules to extract basic propositional rule intents from rules sentences by analysing the corresponding dependency trees. Typically the combination of *verb, noun, adjective* makes a propositional rule intent.

The rule intent is formally expressed as $R(S, H, A)$ where-

S - is the string representing the predicate of the intent.

H - is the head word. The main word of the dependence tree which creates the rule intent becomes the head-word of the extracted rule intent.

A - is the collections of arguments. Each argument in A has two attributes. 1) '*value*' of the argument and 2) '*role*' of the argument. The attribute '*role*' of a argument tries to capture the association of the attribute with the concerned rule intent. The '*role*' can hold following values:

(1) *subject* - marks the argument as subject.

(2) *object* - marks the argument as object.

(3) *pSubject* - marks the argument as prepositional subject.

(4) *pObject* - marks the argument as prepositional object.

(5) *objective* - when a rule intent become argument to another then the role of the previous one is marked as argument. The earlier argument is marked as *objective.*

The following notations are used for describing the rules. The '*p*' denotes a parent node, '*c*' denotes a child node, '*gp*' denotes grand-parent node. The string value of a node (n) in the tree can be retrieved by accessing the word attribute ($n.word$). A '$*$' in '$R(S, H, A)$' means it is either unchanged or not handled with in the rule. An edge between p and c is denoted as $edgename(p, c)$. For example, the $nsubj(p, c)$ denotes that an edge from p (`governor`) to c(`dependent`) labelled *nsubj*.

The heuristic rules are grouped according to edges of the dependency tree of the rule intents. These groups are described in the subsequent sub sections.

4.1.1 Subject/Object Rules

The *subject/object* rules are the most primitive rules. These rules are applied first, before any other rules. These primitive rules (R_1 to R_8) are described below.

$$\forall c_i \ nsubj(p, c_i) \rightarrow \begin{cases} R(p.word, p, \{a\}) \ where \\ a.value = c_i \wedge a.role = subject \ if \ : i = 1 \\ R(*, p, A \cup \{a\}) \ where \\ a.value = c_i \wedge a.role = subject \ if \ : i > 1 \end{cases} \quad (R_1)$$

$$\forall c_i \ nsubjpass(p, c_i) \rightarrow \begin{cases} R(\text{`do'} + p.word, p, \{a\}) \ where \\ a.value = c_i \wedge a.role = object \ if \ : i = 1 \\ R(*, p, A \cup \{a\}) \ where \\ a.value = c_i \wedge a.role = object \ if \ : i > 1 \end{cases} \quad (R_2)$$

$$\forall c_i \ dobj(p, c_i) \rightarrow \begin{cases} R(p.word, p, \{a\}) \ where \\ a.value = c_i \wedge a.role = object \ if \ : i = 1 \\ R(*, p, A \cup \{a\}) \ where \\ a.value = c_i \wedge a.role = object \ if \ : i > 1 \end{cases} \quad (R_3)$$

$$\forall c_i \ amod(p, c_i) \rightarrow R(\text{`is'} + c.word, p, \{a\}) \wedge (a.value = c_i \wedge a.role = subject) \quad (R_4)$$

$$\forall c_i \ agent(p, c_i) \rightarrow \begin{cases} R(p.word, p, \{a\}) \ where \\ (a.value = c_i \wedge a.role = subject) \ if \ : i = 1 \\ R(*, p, A \cup \{a\}) \ where \\ (a.value = c_i \wedge a.role = subject) \ if \ : i > 1 \end{cases} \quad (R_5)$$

$$\forall c_i \ acomp(p, c_i) \rightarrow \begin{cases} R(p.word + c.word, p, \{a\}) \ where \\ (a.value = c_i \wedge a.role = object) \ if : i = 1 \\ R(*, p, A \cup \{a\}) \ where \\ (a.value = c_i \wedge a.role = object) \ if : i > 1 \end{cases} \quad (R_6)$$

$$rcmod(gp, p) \rightarrow R(*, p, \{a\}) \wedge (a.value = gp \cup a.role = object) \quad (R_7)$$

$$vmod(gp_i, p) \wedge postag(p, \text{‘verb’}) \wedge \nexists aux(p, \text{‘TO’}) \rightarrow R(\text{‘is’} + p.word, p, A \cup \{a\}) \wedge (a.value = gp_i \wedge a.role = object) \tag{R_8}$$

Consider the rule sentence (rs_1) from Table 3, the corresponding dependency tree is shown in Figure 2. The predicate *doOpen* (*open* as headword) with ‘`account`’ as an argument is extracted by applying rule R_2. The ‘*account*’ is dependent on *open* with a dependency relation ‘*nsubjpass*’ (*nsubjpass*(opened, account)). Consider the ‘*amod*’ edge, the predicates ‘*isFictitious*(name)’ and ‘*isAnonymous*(name)’ are created by applying rule R_4 with head word ‘*name*’. The concept of *headword* is borrowed from the Named Entity Recognition (NER) [32, 27] and Relation Extraction (RE) [17, 15, 21]. The headword in each propositional rule intent is used in other group of heuristic rules.

4.1.2 Prepositional Rules

Prepositional edges are denoted as *prep_‘x’*(p, c) where ‘x’ is the preposition that is used. The semantic meaning of the phrase that uses the preposition depends on the word at p, c, and on the preposition. The following are the rules that handle the preposition edges.

$$\forall c_i \; prep\_\text{‘x’}(p, c_i) \wedge postag(p, \text{‘verb’}) \wedge postag(c_i, \text{‘noun’}) \rightarrow \begin{cases} R(p.word, p, \{a\}) \wedge \\ (a.value = c_i \wedge a.role = pObject) \; if & : i = 1 \\ R(*, p, A \cup \{a\}) \wedge \\ (a.value = c_i \wedge a.role = pObject) \; if & : i > 1 \end{cases} \tag{R_9}$$

$$\forall c_i \; prep\_\text{‘x’}(p, c_i) \wedge postag(p, \text{‘verb’}) \wedge postag(c_i, \text{‘adjective’}) \rightarrow \begin{cases} R(p.word, p, \{a\}) \wedge \\ (a.value = c_i \wedge a.role = pObject) \; if & : i = 1 \\ R(*, p, A \cup \{a\}) \wedge \\ (a.value = c_i \wedge a.role = pObject) \; if & : i > 1 \end{cases} \tag{R_{10}}$$

$$prep\_\text{‘x’}(p, c_i) \wedge postag(p, \text{‘noun’}) \wedge postag(c_i, \text{‘noun’}) \rightarrow R(\text{‘is\_x’}, p, \{a_1, a_2\}) \wedge (a_1 = p \wedge a_1.role = pSubject) \wedge (a_2.value = c_i \wedge a_2.role = pObject) \tag{R_{11}}$$

$$prep\_\text{‘x’}(p, c_i) \wedge postag(p, \text{‘adjective’}) \wedge postag(c_i, \text{‘noun’}) \rightarrow R(\text{‘is\_p.word\_x’}, p, \{a\}) \wedge (a.value = c_i \wedge a.role = pObject) \tag{R_{12}}$$

Consider the rule sentence rs_1 from Table 3 and corresponding dependency tree in Figure 2. We applied the primitive rule R_2 and identified a rule intent ‘*doOpen*(account)’. But the word ‘*open*’ has a ‘prep_in’ edge to ‘*name*’. The meaning conveyed by the edge is - ‘account is opened in name’. The ‘*prep_in*’ edge connects a *verb* (open) to a *noun* (name). On applying R_9, the rule intent is modified as ‘*doOpen*(account, in name)’ where ‘*(*name)’ is the prepositional object of the rule intent.

Consider rs_3 from Table 3, the nodes ‘*proof of address*’, ‘*proof*’ have a ‘prep_of’ edge to ‘*address*’ where the words ‘*proof*’ and ‘*address*’ are nouns. A new rule intent is created with *governor* as headword. This rule intent has both *governor* and *dependent* as arguments. Argument *governor* is added as prepositional subject and *dependent* is added as prepositional object. The predicate is built by appending the preposition with ‘*is-*’. The phrase ‘*proof of address*’ from rs_3 of Table 1 is created as ‘*is-of*(proof, address)’ using rule R_{11}.

4.1.3 Objective Rules

The objective rules are meant for handling special situation where a rule intent can be argument to another. For example, consider “*...bank fails to furnish information on the remitter,...*”, a segment of rule sentence rs_2 for illustration. The corresponding dependency tree is shown in Figure 3. The objective ‘*furnish information on remitter*’ is accomplished by ‘*bank fails*’. The appropriate rule intent for this statement can be represented as - ‘*fails*(bank, *furnish*(information, on remitter))’.

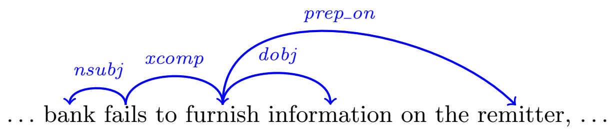


Figure 3: Dependency tree for segment of rs_2

$$\forall c_i \; xcomp(p, c_i) \wedge \exists R(S, c_i, A) \wedge \exists R(S', p, A') \rightarrow R(S', p, A' \cup \{a_i\}) \wedge (a_i.value = R(S, c, A) \wedge a_i.role = objective) \tag{R_{13}}$$

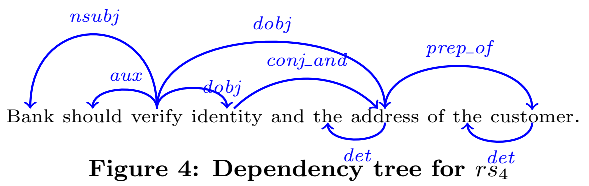
$$\forall c_i \; ccomp(p, c_i) \wedge \exists R(S, c_i, A) \wedge \exists R(S', p, A') \rightarrow R(S', p, A' \cup \{a_i\}) \wedge (a_i.value = R(S, c, A) \wedge a_i.role = objective) \tag{R_{14}}$$

$$\forall c_i \; vmod(p, c) \wedge \exists R(S, c, A) \wedge \exists aux(p, \text{‘}TO\text{’})) \wedge \exists R(S', p, A') \rightarrow R(S', p, A' \cup \{a_i\}) \wedge (a_i.value = R(S, c, A) \wedge a_i.role = objective) \tag{R_{15}}$$

In above example, the rule R_{13} is applied as a dependency exists for two verbs (*fails, furnish*) with *xcomp*. In this case, the dependency tree is traversed in level-order fashion and relevant primitive rules (R_1, R_3) and prepositional rule (R_9) are applied before rule R_{13}. As a result the rule intent ri_5 will be added as an argument to rule intent ‘*fails*(bank)’(ri_6).

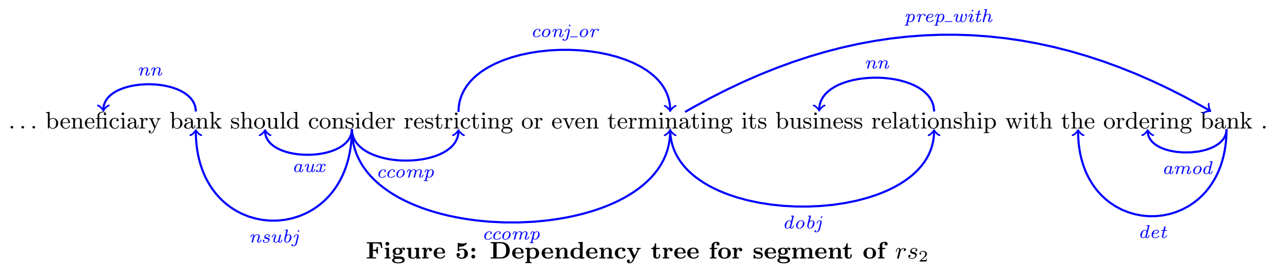
4.1.4 Split Rules

The split rules are applicable in scenario where the rule intent has more than one *subject* or *object*. Consider the sentence rs_4 from the example sentences, the dependency tree is shown in Figure 4. The subject/object rules R_1 and R_3 are applied to get the predicate ‘*verify*(bank, identity, address)’. Ideally, the above predicate should be split into ‘*verify*(bank, identity)’ and ‘*verify*(bank, address)’ to make it more meaningful. The following rules are applied to handle above scenarios.



nsubj
dobj
aux
dobj
conj_and
prep_of
Bank should verify identity and the address of the customer.
det
det

Figure 4: Dependency tree for rs_4



nn
conj_or
prep_with
nn
. . . beneficiary bank should consider restricting or even terminating its business relationship with the ordering bank .
aux
ccomp
amod
nsubj
ccomp
dobj
det

Figure 5: Dependency tree for segment of rs_2

let $\{\ \mathscr{A}_{role_k} \subset A \mid \forall i,j\ a_i, a_j \in \mathscr{A}_{role_k} \wedge (a_i.role \equiv a_j.role \equiv role_k)$ where $i \neq j\ \}$, then-

$$\begin{aligned} &if\ R(S,H,A) \wedge (|\mathscr{A}_{role_k}| \geq 2),\ then\ \forall a \in \mathscr{A}_{role_k} \\ &\quad (R(S,H,A^a)\Big| A^a = (\{a\} \cup (A - \mathscr{A}_{role_k})) \end{aligned} \qquad (R_{16})$$

For sake of simplicity we are not considering '*pSubject*' and '*pObject*' arguments for splitting.

There exists a scenario where a new rule intent is created out of an existing one. In this scenario, two verbs are connected with a '*conjunction*' edge and they share common argument in the sentence. The common argument, which may be a word or phrase (which creates the argument), is attached to any one of the verb in the dependency tree. The other verb is overlooked simply because of being deprived of having a proper edge (edge other than '*conj*' edge). To illustrate, consider the phrase a segment of rs_2 and corresponding dependency given in Figure 5. In this, '*restricting*', '*terminating*' are connected with edge '*conj_or*'. '*Conjunction*' edges are denoted by '*conj_<x>*' where '*x*' is that particular conjunction. In the example, the '*or*' conjunction is used and both the verbs share the same argument. Though '*relationship*' and '*bank*' are directly connected to '*terminating*', they are not directly connected with '*restricting*'. Hence the node with '*restricting*' will not satisfy any of the rules R_1-R_{15}. An ideal rule intent '*terminating*(with bank, relationship)' is preferred.

$$\begin{aligned} &\forall c_i\ conj\_\text{'x'}(p, c_i) \wedge postag(p, \text{'verb'}) \wedge postag(c_i, \text{'verb'}) \wedge \\ &(\exists r \mid r \in \{p, c_i\}, degree(r) \not> 1) \wedge \exists R(S, \{p, c_i\} - r, A) \rightarrow \\ &\qquad\qquad R(r.word, r, A) \end{aligned} \qquad (R_{17})$$

The rule intent '*restricting*(with bank, relationship)' can be extracted on applying R_{17}. If an edge $conj\_\text{'}x\text{'}(p, c_i)$ exists then a node r is chosen from $\{p, c_i\}$ with which a rule intent can not be created. In the earlier scenario, the word '*restricting*' is chosen and a rule intent with this node r created by copying the arguments from a node other than r from $\{p, c_i\}$.

4.1.5 Negation Rule

$$\begin{aligned} &neg(p,c) \wedge \nexists R(S,p,A) \rightarrow R(\text{null}, p, \phi) \wedge \\ &\qquad markNegate(R(\text{null}, p, \phi)) \end{aligned} \qquad (R_{18})$$

Now let $\mathscr{R} = \{R(S', gp, A') \mid gp \text{ is parent of } p\}$

$$\begin{aligned} &markNegate(R(null,p,\phi)) \wedge \neg postag(p, \text{'verb'}) \\ &\qquad \wedge \mathscr{R} \neq \phi \rightarrow \forall r \in \mathscr{R}\ markNegate(r) \end{aligned} \qquad (R_{19})$$

$$neg(p,c) \wedge \exists R(S,p,A) \rightarrow markNegate(R(S,p,A)) \qquad (R_{20})$$

The negation rules (R_{18}-R_{20}) handles the '*neg*' edge and marks a rule intent in negated form by using the function '*markNegated*'. These rules also check whether there exists any rule intent with *source* of the *neg* edge as head. If exists then the rule intent is marked as negated (R_{20}). If there is no rule intent with the *source* then a dummy rule intent is created with a *null* as predicate and argument (R_{18}). The rule R_{19} is applied to handle dummy rule intents after creating rule intents from all other nodes by applying rules R_1-R_{17}. The rule R_{19} checks for the existence of rule intent with the grand parent word as head and marks that rule intent negated. All the dummy intents will be discarded at the end. Consider the dependency tree in Figure 2, the '*neg*' edge between '*account*' to '*No*' create a dummy rule intent with '*account*'. The parent of '*account*' is '*opened*' and there exists a rule intent '*doOpen*(account,in name)'. The intent is marked as '$\neg$*doOpen*(account,in name)' by applying R_{19}.

The rules R_1-R_{20} are applied by traversing the dependency tree in level-order. This is done to have the knowledge of the children before processing any parent. These heuristics driven rules are based on the edges as defined in the dependency tree (of Stanford parser), and may need to be extended further to cover all possible edges.

5. EXPERIMENT AND RESULTS

In this section, we discuss the technologies used in the implementation of our prototype system, the time complexity of the implementation, the experimental studies conducted. In the end, we also discuss about observations during the experimentation and the limitations of our approach.

5.1 Implementation

We used Stanford coreNLP parser [11] for parsing the rule sentences, `POS` tagging, and for creating the dependency tree. We implemented `RIE` using heuristic rules defined in Section 4.2. We built a prototype tool to integrate the above phases. We built this as an eclipse plug-in and as a part of the tool `BuRRiTo` [9]. We experimented our prototype tool on `Windows 7` machine with `CORE`$i5$ processor and 2 GB RAM. A snapshot of the tool [9] with `SBVR` vocabulary from sentence rs_1 and rs_4 is given in Figure 7. Our approach is helpful to provide this kind of output from rule sentences in natural language. Terms and simple facts presented in the editor (e.g. line no 1-6 or 11-27) are generated from the knowledge extracted by our RIE technique. However some human intervention is required to edit the extracted facts to make them correct for further processing.

5.2 Experimental Study

We evaluated our prototype with two sets of subjects:

1. Know Your Customer (`KYC`) document.
   The `KYC` document consists of guidelines for banks about collecting various details of their customer in conducting their business. The training set consists of `KYC`

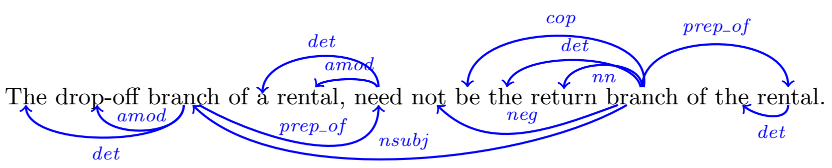
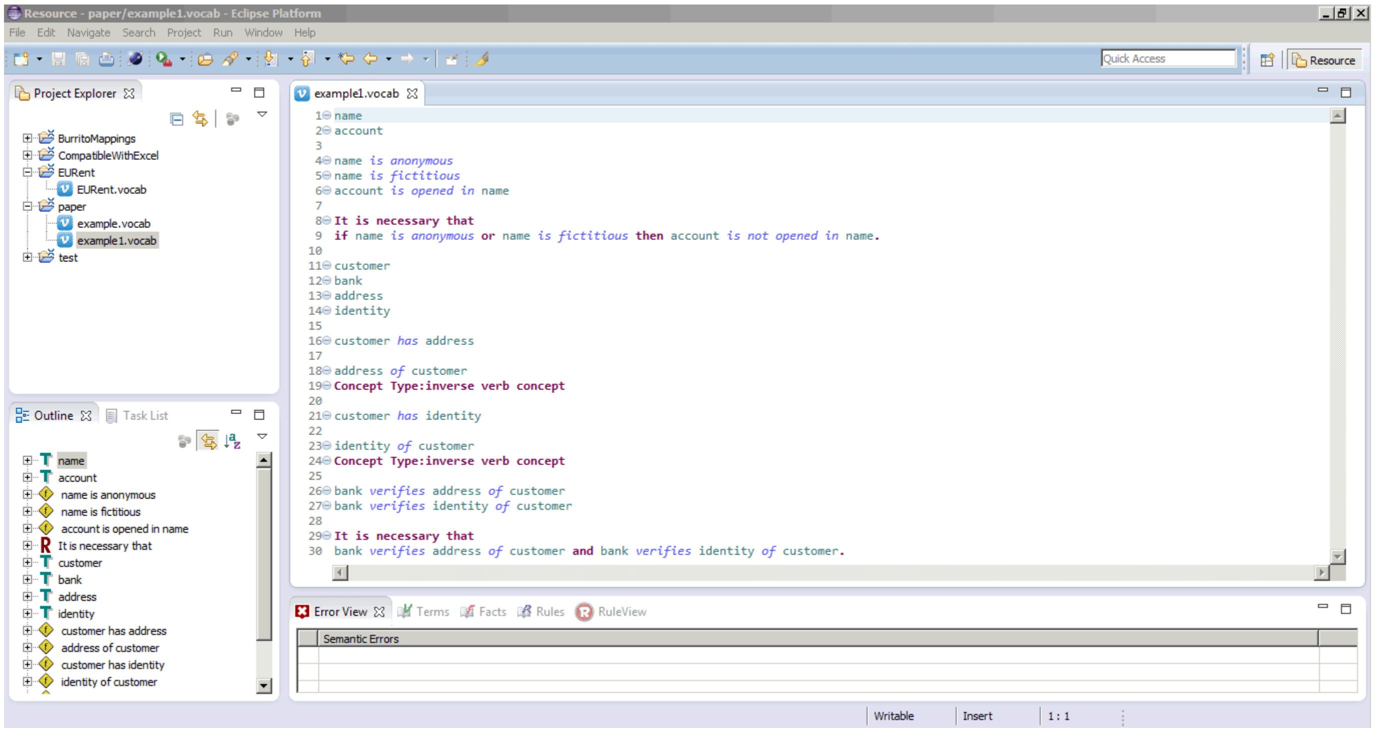

cop
prep_of
det
det
amod
nn
The drop-off branch of a rental, need not be the return branch of the rental.
amod
neg
prep_of
nsubj
det
det

Figure 6: Dependency tree of 'The drop-off branch of a rental need not be the return branch of the rental.'



```
name
account

name is anonymous
name is fictitious
account is opened in name

It is necessary that
if name is anonymous or name is fictitious then account is not opened in name.

customer
bank
address
identity

customer has address

address of customer
Concept Type:inverse verb concept

customer has identity

identity of customer
Concept Type:inverse verb concept

bank verifies address of customer
bank verifies identity of customer

It is necessary that
bank verifies address of customer and bank verifies identity of customer.
```

Figure 7: Screenshot of BuRRiTo guided by RIE.

guidelines taken from Reserve Bank of India (RBI) KYC guideline document [3] The set comprised of 630 sentences, out of them 185 sentences were marked as non-rule or non-relevant sentences.

2. EU-RentACar.
we also used requirements for a fictitious car rental company EU-RentACar [1]. The contents of this document has 71 rule sentences.

In evaluating our approach, we measured the efficacy on the basis of two parameters 1) recall, and 2) precision. The *Recall* is defined as the ratio of number of True Positive Instances (TPI) detected to the number of Actual Positive Instances (API) is shown in 1.

$$recall = \frac{|TPI|}{|API|} \tag{1}$$

The *precision* is defined as ratio of number of TPI to the total number of TPI and False Positive Instances (FPI) is shown in 2.

$$precision = \frac{|TPI|}{|TPI| + |FPI|} \tag{2}$$

The efficacy of RIE is measured for recall and precision with (1) and (2). A single *rule sentence* may have more than one *rule intent*. We used sentences from KYC guidelines of RBI and business rule document of EU-RentACar case study. Efficiency of RIE depends on the structure of the sentences because these sentences may consists of more than one clause. A typical example for a complex structure consisting of clauses and conjunctions is shown in Table 4.
The sentences from EU-RentACar case study are simple

"*In case of transactions carried out by a walk-in customer, where the amount of transaction is equal to or exceeds rupees fifty thousand, whether conducted as a single transaction or several transactions that appear to be connected, the customer's identity and address should be verified.*".

Table 4: Example complex sentence.

and it clearly reflects in the result shown in Table 5 .
Figure 8 and 9 shows the usage pattern of the heuristics rules. The Subject/Object category rules are the most

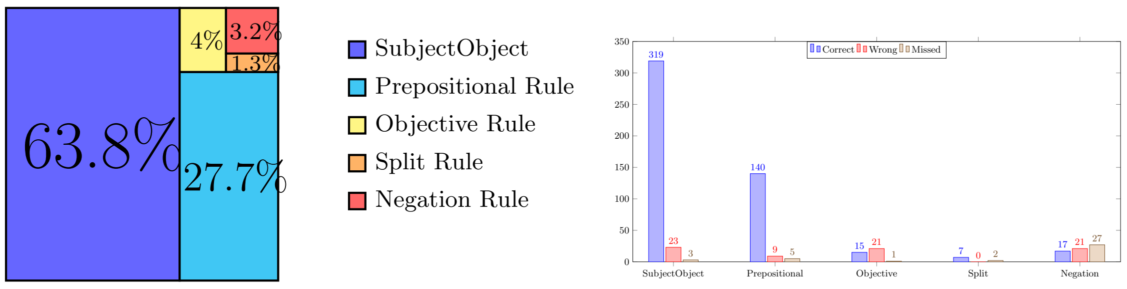


Figure 8: Eu-RentAcar Study:Pie chart shows overall usage pattern(left). Bar chart shows experimental results

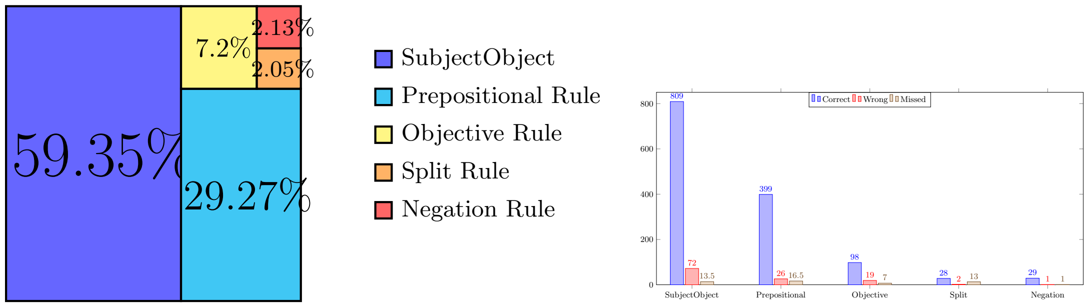


Figure 9: KYC Study:Pie chart shows overall usage pattern(left). Bar chart shows experimental results

Subject	Recall	Precision
KYC document simple sentences	0.8108	0.7834
EU-RentACar Sentences	0.78	0.76
KYC document complex sentences	0.557	0.5114

Table 5: Result of Rule Intent Extraction

frequently used rules in our experimentation. The second most used rule category is preposition rules. The errors in Subject/Object rule group is primarily contributed by R_4. The words like ‘other’, ‘due’ have created spurious rule intents ((e.g. *isOther (customer)*, *isDue(diligence)* etc.)). The presence of pronouns made Subject/Object rules to detect incorrect subjects and objects.

These pronouns occur with valid edges like `nsubj`, `dobj`. In some scenarios, the pronouns occur as only child with a valid edge. In those scenarios, ignoring the pronouns may lead to the loss of rule intent. The co-reference resolution is required to overcome such scenarios. The structure of a rule sentence has contributed to errors that are caused by prepositional rules. Consider the sentence given in Figure 6, the rule intent ‘*is-of*(branch,need)’ is extracted by using rule R_{11}. For same sentence when Subject/Object rules followed by Negation rules are applied, extract a wrong rule intent of the form ‘$\neg$*branch*(branch)’. In the rule intent, the ‘`branch`’ is extracted from the word at twelfth position of the sentence and argument ‘`branch`’ is extracted from third position from the same sentence.

5.3 Limitations

The NL sentences pose a big challenge as a sentence can be expressed in several ways. The NL sentences can be simple (with only one clause), complex (having multiple clauses). The variations in way of interleaving clauses elicit our heuristic rules to be enriched enough to handle such sentences. A typical example of a complex sentence is shown in Table 4. Our approach performs well in case of simple sentences and sentences with non-interleaving clauses. A trivial change in usage of punctuation mark may change the structure of the dependency tree as shown in Figure 10 and 11. As a consequence from the tree of Figure 10 in, we get a rule intent like ‘*use*(Banks, for verification, evidence)’. We also get another rule intent from the tree in Figure 11 as ‘*use*(Banks, evidence)’. Our method has shown promising results when the rule sentences are syntactically and semantically correct.

Sentences can also be formed by interleaving clauses in more than one ways. Sentences with same semantic meaning but being syntactically of different structures give different dependency trees. So rule intents extracted from these sentences may differ in their form. We need to come up with some procedure to normalize either in sentence space or in rule intent space to get congruent result from sentences with same semantic meaning. Consider, for example, rule sentence rs_4 table 3 and a slight variation of the same can be **‘`Bank needs to verify identity and the address of the customer.`’** From the later one we will have rule-intent of the form ‘*needs*(bank, *verify*(address))’. This rule-intent semantically equivalent to ri_{17}.

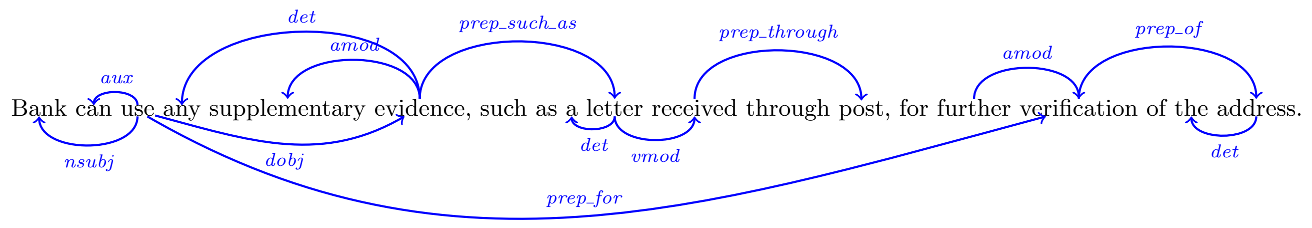

det
amod
prep_such_as
prep_through
amod
prep_of
aux
Bank can use any supplementary evidence, such as a letter received through post, for further verification of the address.
nsubj
dobj
det
vmod
det
prep_for

Figure 10: Dependency tree of an example with punctuation mark ',' after '*evidence*'.

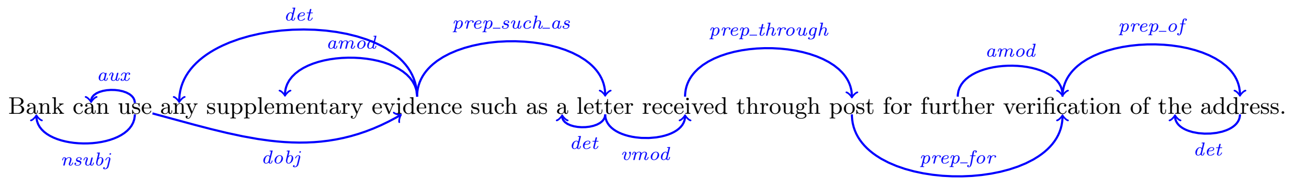

det
amod
prep_such_as
prep_through
amod
prep_of
aux
Bank can use any supplementary evidence such as a letter received through post for further verification of the address.
nsubj
dobj
det
vmod
prep_for
det

Figure 11: Dependency tree of an example without punctuation mark ','.

6. FUTURE WORK

During the experimentation, we made numerous observations for improving and extending our approach. Few of them are:

- We would like to handle complex and compound sentences. We plan to extend the approach 1) by using dependency parser with newer patterns, 2) by creating more rules on processing dependency tree, and 3) The simplification of a complex / compound sentences may be done in couple different ways: 1) A complex sentence can be summarized into a single fact (rule intent). 2) A complex sentence can be broken into simple sentences by keeping proper note of how they are related. We plan to explore either of these strategies for further experimentation.

- We have made progress to extract basic facts and to some extent the vocabulary. To make them perfect we are delving with neamed- entity recognition techniques. To represent the facts as a business rule and to qualify them for further processing we are investigating mining relationship among those facts. This relationship can be of two types- intra-sentence and inter-sentences.

- Co-references within and across sentences is a major issue to be handled. For example, consider the sentence "*Circumstances, in which a customer is permitted to act on behalf of another person_entity, should be clearly spelt out*".The word '*Circumstances*' refers to the situation under which '*a customer is permitted to act on behalf of another person_entity*'. This is an example of co-reference in a sentence. The sentence "*In the case of cross-border customers, there is the additional difficulty of matching the customer with the documentation and the bank may have to rely on third party certification. In such cases, it must be ensured that the third party is a regulated and supervised entity and has adequate KYC systems in place.*", is a similar co-reference example but spread over more than one sentence. In the second sentence '*such cases*' refers to the situation described in the first sentence.

- We plan to convert the propositional form of the rule intents in `SBVR` format. The `SBVR` is a well-known format in representing the business rule. It enables to do certain kinds of reasoning with the business rules. We are currently made some progress to extract vocabulary associated with rule-intents and express rule-intents in a format close to SBVR.

- Based on our knowledge and exploration, we have not found relevant data sets or benchmarks. We plan to create a exhaustive annotated data set as benchmark.

7. CONCLUSION

The larger objective of our work is to extract formal business rules (and processes) by analysing requirements document, guidelines and do's and don'ts documents. To achieve the objective, we split the problem into multiple parts. The rule intent extract is one of these identified problems. This current work of ours focuses on the extraction of rule intents. Depending upon the heuristics that are independent of the business domain, we extracted the business intents from the sentences. We have to experiment with many more documents to establish the adequacy of the heuristics, which we may parametrize in the long run. Our experiments with openly available documents and subsequent improvements in the methods yielded promising results in terms of the measures of precision, recall and accuracy. The above success give us enough motivation to move further towards expressing the rules using a formal notation such as `SBVR`, and various improvement and extensions to the techniques to make them useful for industry application.

8. REFERENCES

[1] EU-RentACar case study. http://www.businessrulesgroup.org/first_paper/br01ad.htm. [Online; accessed 18-April-2016].

[2] Object Management Group(OMG). http://www.omg.org. [Online; accessed 29-September-2015].

[3] Reserve Bank of India (RBI), Master Circulars. https://rbi.org.in/scripts/BS_ViewMasCirculardetails.aspx/?id=9031. [Online; accessed 18-April-2016].

[4] Semantics Of Business Vocabulary And Rules (SBVR). http://www.omg.org/spec/SBVR/. [Online; accessed 29-September-2015].

[5] J. Atkinson-Abutridy, C. Mellish, and S. Aitken. Combining information extraction with genetic algorithms for text mining. *IEEE Intelligent Systems*, 19(3):22–30, May 2004.

[6] I. S. Bajwa, M. G. Lee, and B. Bordbar. SBVR business rules generation from natural language specification. In *AI for Business Agility, Papers from the 2011 AAAI Spring Symposium, Technical Report SS-11-03, Stanford, California, USA, March 21-23, 2011*, 2011.

[7] B. K. Boguraev. Towards finite-state analysis of lexical cohesion. In *Proceedings of the 3rd international conference on finite-state methods for NLP*, 2000.

[8] C. C. Chiang. Extracting business rules from legacy systems into reusable components. In *2006 IEEE/SMC International Conference on System of Systems Engineering*, pages 6 pp.–, April 2006.

[9] P. K. Chittimalli and K. Anand. Domain-independent method of detecting inconsistencies in sbvr-based business rules. In *Proceedings of the International Workshop on Formal Methods for Analysis of Business Systems@ASE 2016*, pages 9–16. ACM, 2016.

[10] F. Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'01, pages 1251–1256, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[11] M.-C. De Marneffe and C. D. Manning. Stanford typed dependencies manual. Technical report, Technical report, Stanford University, 2008.

[12] A. B. Earls, S. M. Embury, and N. H. Turner. A method for the manual extraction of business rules from legacy source code. *BT Technology Journal*, 20(4):127–145, Oct. 2002.

[13] C. J. Fillmore. The case for case, dins. In E. Bach and R. Harms, editors, *Universals in Linguistic Theory.* Holt, Rinehart, and Winston, 1968.

[14] S. Ghaisas, M. Motwani, and P. Anish. Detecting system use cases and validations from documents. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, pages 568–573, Nov 2013.

[15] Z. GuoDong, S. Jian, Z. Jie, and Z. Min. Exploring various knowledge in relation extraction. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 427–434, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[16] H. Huang. Business rule extraction from legacy code. In *Proceedings of the 20th Conference on Computer Software and Applications*, COMPSAC '96, pages 162–, Washington, DC, USA, 1996. IEEE Computer Society.

[17] N. Kambhatla. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In *Proceedings of the ACL 2004 on Interactive Poster and Demonstration Sessions*, ACLdemo '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[18] I. Muslea et al. Extraction patterns for information extraction tasks: A survey. In *The AAAI-99 Workshop on Machine Learning for Information Extraction*, volume 2, 1999.

[19] T. Nasukawa and T. Nagano. Text analysis and knowledge mining system. *IBM Systems Journal*, 40(4):967–984, 2001.

[20] R. Pandita, X. Xiao, H. Zhong, T. Xie, S. Oney, and A. Paradkar. Inferring method specifications from natural language api descriptions. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 815–825, Piscataway, NJ, USA, 2012. IEEE Press.

[21] S. Pawar, P. Bhattacharyya, and G. K. Palshikar. Semi-supervised relation extraction using em algorithm. https://www.cse.iitb.ac.in/~pb/papers/icon13-ie-em.pdf, 2014.

[22] E. Putrycz and A. W. Kark. *Rule Representation, Interchange and Reasoning on the Web: International Symposium, RuleML 2008, Orlando, FL, USA, October 30-31, 2008. Proceedings*, chapter Connecting Legacy Code, Business Rules and Documentation, pages 17–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[23] L. R. Rabiner and B.-H. Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16, 1986.

[24] C. Rolland and C. B. Achour. Guiding the construction of textual use case specifications. *Data Knowl. Eng.*, 25(1-2):125–160, Mar. 1998.

[25] A. Sinha, A. Paradkar, P. Kumanan, and B. Boguraev. A linguistic analysis engine for natural language use case description and its application to dependability analysis in industrial use cases. In *Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, pages 327–336, June 2009.

[26] H. M. Sneed. Extracting business logic from existing cobol programs as a basis for redevelopment. In *Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on*, pages 167–175, 2001.

[27] N. Sobhana, P. Mitra, and S. Ghosh. Conditional random field based named entity recognition in geological text. *International Journal of Computer Applications*, 1(3):143–147, 2010.

[28] C. Wang, Y. Zhou, and J. Chen. Extracting prime business rules from large legacy system. In *Computer Science and Software Engineering, 2008 International Conference on*, volume 2, pages 19–23, Dec 2008.

[29] X. Wang, J. Sun, X. Yang, Z. He, and S. Maddineni. Business rules extraction from large legacy systems. In *Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on*, pages 249–258, March 2004.

[30] X. Xiao, A. Paradkar, S. Thummalapenta, and T. Xie. Automated extraction of security policies from natural-language software documents. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 12:1–12:11, New York, NY, USA, 2012. ACM.

[31] H. Zhong, L. Zhang, T. Xie, and H. Mei. Inferring resource specifications from natural language api documentation. In *Proceedings of the 2009*

IEEE/ACM International Conference on Automated Software Engineering, ASE '09, pages 307–318, Washington, DC, USA, 2009. IEEE Computer Society.

[32] G. Zhou and J. Su. Named entity recognition using an hmm-based chunk tagger. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 473–480, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.