Automated Identification of Component State Transition Model Elements from Requirements

Kaushik Madala, Danielle Gaither, Rodney Nielsen, Hyunsook Do
Computer Science and Engineering
University of North Texas
{kaushikmadala, daniellegaither}@my.unt.edu, {rodney.nielsen, hyunsook.do}@unt.edu

***Abstract*—Most system requirements are currently written in common, i.e., unstructured, natural language, which existing requirements analysis tools are poorly equipped to handle. Extracting mentions of model elements from common natural language requirements is a first step toward the automation of model-driven requirements analysis. We propose an approach in which we identify mentions of elements of a component state transition (CST) model in natural language requirements by creating classifiers using a recurrent neural network with long short-term memory. To evaluate our approach, we performed a study on a pacemaker system requirements document, and the results show promising directions for future research.**

I. Introduction

Identifying defects earlier in the software development life cycle reduces the costs of software development and maintenance. Martin [1] stated that 56% of defects in software development originate during the requirements engineering phase. In application domains like embedded systems, such defects might result in safety issues when not dealt with properly. The use of model-driven methods for requirements analysis can facilitate communication between requirements analysts and system stakeholders. However, generating such models by a fully manual process is tedious, extremely time consuming, and is still error-prone.

The idea of automated requirements modeling is not new. However, currently available techniques can only generate models from natural language requirements that are very simple or adhere to a previously specified structure. Such methods do not adequately deal with the complex, unstructured sentences used by 79% of software companies to capture requirements [2].

Friedrich et al. [3] explored the idea of using natural language text to generate models in an automated way, although their work does not deal with requirements models as such. However, their work in using natural language to generate Business Process Modeling Notation (BPMN) models can inform our approach. One example of an automated approach to building requirements models comes from the work of Yue et al. [4], who automatically generated UML analysis models from use case diagrams by using a syntactic parser. However, the technique can only identify model elements expressed as a single word. Popescu et al. [5] proposed an approach to generating an analysis model to determine inconsistencies in requirements using a natural language parser, but it can identify model elements only if the sentences containing them are short and simple. None of the methods previously described address requirements expressed using natural, unstructured sentences.

In our previous work, we proposed a model-driven requirements analysis approach, the causal component model (CCM), which uses system components and the interactions between them as the basis for analyzing system requirements [6], [7]. Using the CCM approach, we identified the relevant model elements to build a component state transition (CST) model. The details of these model elements will be explained in Section III. Currently the model building process is performed manually and requires significant human effort. The work in this paper aims to reduce the required human effort by automatically extracting CST mentions from requirements documents (a mention is a text span that maps to an instance of a model element). The need for close attention to requirements in systems such as embedded systems is due to their criticality. If the interactions between components in such systems are not studied, the results can be catastrophic [8].

Our approach can be adapted for various model-driven, component-based requirements analysis methods such as RE-UML [9] and contract-based frameworks [10].

The major goals of our proposed approach are to reduce the amount of human effort needed when providing input for component-based requirements analysis tools such as CCM, and to advance the state of automated model-driven requirements analysis. We believe this will aid stakeholders in their decision-making process. In order to analyze common natural language requirements, we need a technique that can accurately recognize words with similar semantic characteristics. Converting words into semantic space vectors [11] helps identify semantically similar items that can be used to construct a model for analyzing requirements.

In this paper, we propose a technique in which we identify CST mentions using recurrent neural networks (RNNs) with long short-term memory (LSTM) [12] using TensorFlow [13]. We train a classifier for each type of model element on human-annotated data and use it to identify those elements within another unseen test document. Using these classifiers, we were able to automatically find 76% of the mentions of system components in a requirements document, which we believe will provide substantial value and time savings in constructing our model. Our research aims to answer the question: how effective are recurrent neural networks at automatically identifying CST mentions in natural language requirements documents?

978-1-5386-3488-2/17 $31.00 © 2017 IEEE
DOI 10.1109/REW.2017.73

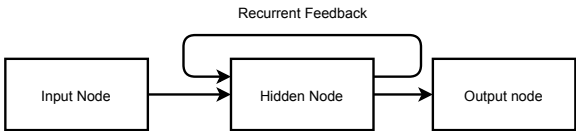


Fig. 1: Simple recurrent neural network

The remainder of this paper is organized as follows. Section II gives basic information on RNNs and word embeddings. Section III describes our proposed approach. Section IV presents an empirical study, the results of which are given in Section V. Section VI discusses the results and possible areas for improvement, and Section VII presents related work. Finally, Section VIII provides conclusions and proposes future research directions.

II. Background

This section provides information on topics used in our proposed approach. Natural language processing involves tasks like language modeling and sequence labeling (e.g., part of speech (POS) tagging and named entity recognition (NER)). Identifying CST mentions is similar to NER, which is a sequence labeling task. A sequence labeling task is a classification task where the classifier is trained on data with ground truths and used on unseen data. In our approach, we used recurrent neural networks (RNN) for classification and word embeddings as their features.

A. Recurrent Neural Networks

Neural networks (NNs) [14] have been widely used for classification tasks since the 1990s. In sequence labeling tasks, such as one of tasks in our approach that labels the sequence of words describing CST mentions, the label of the current word in sequence depends on the previous words. While basic neural networks ignore such state information from the previous instances, recurrent neural networks (RNN) retain such information.

Figure 1 shows a simple RNN with one input, hidden, and output node. The input node in RNN refers to the input data we feed into RNN, the hidden node in RNN is where the input is transformed into a format suitable for output node to predict the label. By giving the feedback to the hidden node, RNN will consider the previous words when classifying the current word.

Recurrent neural networks pertain a long-term dependency problem. Long short-term memory (LSTM) networks [12] overcome this problem by limiting the amount previous information to consider.

Like other classification algorithms, RNNs have parameters that must be tuned to improve the classifier's performance. For RNNs with LSTM, these parameters are the number of units in the LSTM cell, the number of epochs, the optimizer, and the learning rate. An epoch is one complete iteration on training data. The learning rate is the rate at which the network discards old beliefs for new ones. The optimizer is the function that aids in reducing error.

There exist several implementations of neural networks [13], [15]. In our research, we used TensorFlow [13], Google's machine-intelligence library, to implement RNN with LSTM.

B. Word Embeddings

In the field of natural language processing, n-gram models have been widely used to predict language-related entities. However, they suffer from the inability to detect patterns due to the curse of dimensionality [16], as words need to be converted to numerical representation for neural networks. Traditionally, words have been represented as sparse vectors, each of which has a length of the number of unique words in the entire data set, often called vocabulary size. Word embeddings [11] address this issue by reducing the dimensionality of a semantic space from the size of the vocabulary, which could be in the millions, to a smaller size, typically 300.

These word embeddings make up the semantic space representation where similar words are closer. Pre-trained word embeddings are typically derived from a large set of documents. In our research, we tested both Glove [17] and Google [18] embeddings with the former containing word vectors for 400,000 unique words and the latter containing 3,000,000 unique words and phrases. The Glove embeddings we used are generated from Wikipedia entries and large collections of news wire articles. The Google embeddings we used are generated from various news articles. Word embeddings have the advantage of reducing the need of human annotation as they yield better results when trained on small labeled data.

III. Approach

Our approach identifies the mentions of model elements of state transition diagrams (CST mentions) for each component in natural language requirements of a component-based system in the domain of embedded systems. We generated classifiers for each type of model element to identify the elements in requirements. These classifiers can be used on new requirements documents to extract the model elements from the document.

While our approach can be adapted to other model-driven approaches [9], [10], we describe our approach considering the component state transition (CST) model that we defined in our previous research [6], [7]. The CST model consists of three types of model elements, which are briefly explained here. The full definitions can be found in [6], [7].

1) Component: A component is any entity that is part of system's composition and can change states. For example, the components of a robot might include a motor, an ultrasonic sensor, and actuators.
2) State: A state refers the operating condition of a component. For example, the component Motor can be in an "off" state or an "'on" state.
3) Transition condition: A transition condition is defined as the environmental or system-related condition that will trigger in state change of component. For example, a user pressing the "off" button could be a transition condition for a motor to transition from the "on" state to the "off" state.

An overview of our proposed approach is illustrated in Figure 2. The figure depicts inputs and outputs in bubbles and major processing elements in rectangles with the step numbers. Figure 2a illustrates the process of generating classifiers, and Figure 2b illustrates the usage of these generated classifiers on unseen or new requirements documents. Note that generation of classifiers is done only once. Any new or unseen requirements documents that are not part of the generation of classifiers use the classifiers for identification of CST mentions. By doing this, requirements engineers can expect great effort savings over time. The following subsections discuss the generation of classifiers and using them for testing in detail.

A. Generating classifiers

Before labeling the training set of requirements, preliminary guidelines must be established. Guidelines help annotators with identification of modeling elements. Once these guidelines are established, the following steps are followed:

1.) Annotate requirements: In order to train classifiers that will be used to process new unseen requirements documents, we need to create training data sets. We need two or more engineers (annotators) to independently label the training set requirements documents to identify the CST mentions of interest (component, states, and transition conditions). At least two annotators are required to ensure the quality of annotated data and to assure that all and only relevant mentions are annotated by engineers. Specifically, they need mark each span of words indicating a model element.

The F_1-measure (see Section IV-B) is useful to get inter-annotator agreement even on imbalanced data sets. After the agreement is found, the annotations from both annotators must be combined into one document, and the annotators decide which annotations must be removed and which can be kept in the document to reach a consensus. If there is a disagreement, experts adjudicate. The guidelines are updated based on the analysis of the adjudication results, and if required, the documents or sections are re-annotated by following the revised guidelines. This task is repeated until the all documents are annotated. The final adjudicated documents are considered to be the gold standard and are used for classification tasks.

2.) Generate IOB data: Because CST mentions are frequently longer than a single word, we used IOB tags [19] in our classification task. IOB tags [19] identify the Inside (I), Outside (O), and Beginning (B) of the modeling element. For example, for the requirement "When the magnet is removed the device shall automatically assume pretest operation", the IOB labels for component, state and transition condition are illustrated in Figure 3. As shown in the third section of Figure 2, there is a transition condition that begins (B) with "When" and continues (I) through "the magnet is removed".

As mentioned in Section II word embeddings make up the semantic space representation where similar words are closer and pre-trained word embeddings reduce the amount of annotated data required for training to yield good results. Google's pre-trained word embeddings or Glove's pretrained embeddings, which are of 300 dimensions; that is, every word is represented as a vector of 300 real number values are pre-trained word embeddings that can be used. Using these word embeddings, labeled data with IOB tags in Step 2 will be converted into labeled featured vectors for recognition of CST mentions. Then a portion of the data must be kept aside as validation data to evaluate the classifiers' performance and determine if the parameters need tuning.

4.) Generate classifiers for each element type in component state transition diagram: Once the labeled feature vectors are generated, classifiers are trained using recurrent neural network (RNN) with long short-term memory (LSTM). We generate a classifier for each type of model element of CST diagram.

5.) Tune the parameters by evaluating the classifiers on validation data: The generated classifiers need to be tested on validation data to evaluate their performance and to check if the parameters (explained in Section II used are overfitting to the training data. The parameters will be tuned until the classifier finds the best fit, at which point the tuning process stops. If the parameters used are not giving satisfactory results, new values will be considered for parameters and Step 4 must be repeated. The parameter values that optimize the F_1-measure on validation data must be utilized to build final classifiers.

B. Using Classifiers

The final classifiers are evaluated on previously unseen requirements documents. The following steps detail the process of using classifiers on unseen requirements documents.

1.) Extract feature vectors: In order to find CST mentions in a unseen or new requirements documents, feature vectors must be extracted for the document similar to Step 3 in sub-section III-A.

2.) Use classifiers to find model elements: The data generated in Step 1 will be fed into the final classifiers generated from the training (see sub-section III-A) to find CST mentions in the unseen requirements documents. This step serves as evaluation of classifiers.

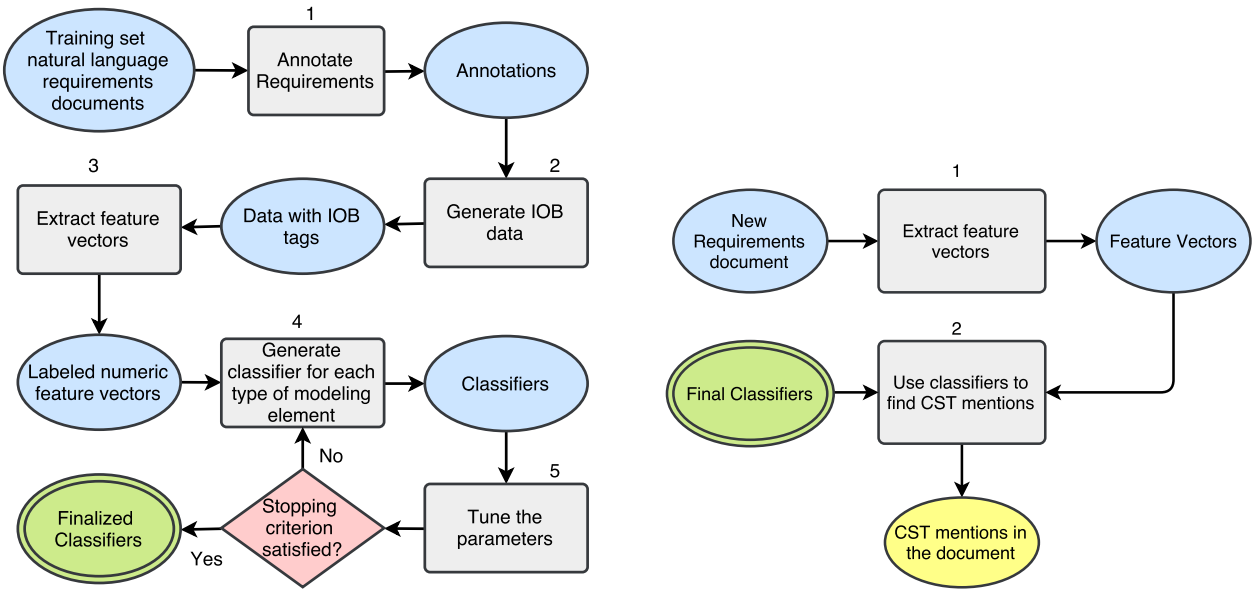
IV. Empirical Study

To evaluate our proposed approach, we performed an empirical study considering the following research question using pacemaker requirements specification:

RQ: How effective are recurrent neural networks at automatically identifying CST mentions in natural language requirements documents?

A. Object of Analysis

We used a pacemaker requirements document [20] to conduct our study. The document is a requirements document for an earlier-generation pacemaker manufactured by Boston Scientific. Several researchers have used the document in their own research work [21], [22], [23]. The pacemaker requirements document is 35 pages long and is partitioned into five sections.



(a) **Generation of classifiers** (b) **Using classifiers**

Fig. 2: **Overview of the approach**

IOB tags for Component

When	the	magnet	is	removed	the	device	shall	automatically	assume	pretest	operation
O	O	B	O	O	O	B	O	O	O	O	O

IOB tags for State

When	the	magnet	is	removed	the	device	shall	automatically	assume	pretest	operation
O	O	O	O	B	O	O	O	O	O	B	O

IOB tags for Transition conditions

When	the	magnet	is	removed	the	device	shall	automatically	assume	pretest	operation
B	I	I	I	I	O	O	O	O	O	O	O

Fig. 3: **Example of IOB tags for each model element**

B. Measures

It is common to evaluate the performance of detecting mentions of interest within text using precision P, recall R, and the F_β-measure. R is the percent of gold-standard mentions correctly identified by the classifier; P is the percent of the items classified by the system as Component/State/Transition (CST) mentions that were correct; and F_β-measure balances P and R, with β indicating which should be weighted more heavily (we use $\beta = 1$, as is common, which results in F_1 being the harmonic mean of P and R):

$$R = \frac{TP}{TP+FN} \qquad P = \frac{TP}{TP+FP} \qquad F_1 = 2 \cdot \frac{P \cdot R}{P+R}$$

where TP (true positives) is the number of gold-standard CST mentions correctly identified by the classifier, FN (false negatives) is the number of gold-standard mentions that the classifier missed, and FP (false positives) is the number of times the classifier classified a CST mention that was not in the gold standard. We use lenient matching such that a partial overlap between the gold-standard and the classified mention is a correct identification, since it is trivial for an analyst to add any required modifiers to the element's name. Throughout the remainder of this document, we will primarily evaluate our results using the F_1-measure.

C. Experimental Process

To perform our experiment, we followed the process we explained in Section III. In order to annotate the requirements document, preliminary guidelines are established and referred while annotating CST mentions. For example, a guideline for identifying components is that a component is an entity with

states. The pacemaker requirements document consists of five sections. Two doctoral students from the University of North Texas who are among the authors of this paper annotated each section. They perform the roles of annotators as mentioned in Section III. We calculated the inter-annotator F_1-measure for each section of the document. More specifically, we considered one of the annotators' labels as a gold standard and the other's as the predicted values and then calculated the F_1-measure. It does not matter which annotator is considered the gold-standard; the reverse results in switching P and R, but F_1 is the same. This is done to calculate inter-annotator agreement.

Table I provides information on inter-annotator agreement, which is the F_1-measure for training and test data of the pacemaker document. The F_1-measures in Table I are the lenient F_1-measures.

TABLE I: **Inter-Annotator agreement values (lenient)**

Dataset	**F_1-measure**		
	Component	**State**	**Transition Condition**
Training (sec. 1, 2, 3, and 4)	0.84	0.80	0.60
Test (sec. 5)	0.86	0.73	0.64

Consensus and Adjudication: For every section annotated, once the F_1-measure was calculated, the annotations from the two annotators were analyzed to obtain consensus. Two faculty members at the University of North Texas with expertise in requirements engineering and natural language processing who are among the authors of this paper were consulted to adjudicate any unresolved disagreements. They played the role of experts as mentioned in Section III. The guidelines were revised as needed.

Data Generation: Once the entire document was annotated, we created labeled feature vectors so a machine learning algorithm could identify the feature patterns consistent with CST mentions.

Classification: We used RNN with LSTM for training the classifier for each type of model element. We partitioned the data into three sets; training (sec. 1, 2 & 3) to build the model, validation (sec. 4) to tune model parameters, and test (sec. 5) to evaluate the final model. We allocated the sections to sets arbitrarily.

We tuned the following parameters: the choice of word embeddings, learning rate, the type of RNN optimizer and the number of training epochs used. The choice of word embeddings defines the semantic space in which we try to find patterns for identifying CST mentions. The goal is to choose word embeddings that give the most accurate semantic representation for the data being used. We evaluated Glove [17] and Google [18] pre-trained word embeddings. Google embeddings provided better results on validation data for all three classifers. In the case of learning rate, which is used to find the optimal weights for RNN, we found that a learning rate of 0.01 yielded the best prediction results. We used the Adam optimizer [24] and Gradient Descent optimizer to check which minimizes error better and chose the Adam optimizer as it yielded better results. We ran experiments with 10, 20, 50, and 100 epochs and found the best results for component and state using 50 epochs, while 10 epochs yielded the best results for transition conditions.

Once the parameters were tuned to find the best fit on the validation data, the final classifiers were trained on Sections 1, 2, 3, and 4. We withheld Section 5 from the classifier generation process and used it as test data to evaluate the classifiers' performance on unseen data. The recall, precision, and F_1-measure values were computed for each type of model element in test data and results were analyzed. Our results and analysis are discussed in Section V.

V. Results

Table II shows the F_1-measure for the classifier of components is 0.71. Note that the F_1-measures mentioned in the table are for all mentions of model elements in the document. The number of unique model elements are smaller when compared to all mentions of model elements. In the case of components, we are able to find 9 out 10 unique components, not all mentions of those 9 components in the text. The classifier for components showed the best performance among all three modeling elements. The F_1-measure of state is 0.52. The classifier for state did not perform as well as the classifier for component. The value from the classifier of transition condition is 0.16. The result for the transition conditions classifier is worse than we expected and clearly it requires improvement. We conjecture that the reason we obtain such a result is that the majority of the transition conditions are unique and we considered only words as features, so the classifier was not able to learn consistent patterns.

However, another important measure to be considered here is recall. Because the end goal of the proposed approach is to help users identify components, states, and transition conditions without reviewing the entire document in detail, the percentage of gold standard labels (human-annotated values) being identified is highly important. Our goal is to achieve 100% recall with the maximum precision possible. Table II shows the recall for components is 0.76, which means that 76% of component mentions in the document were recognized. With respect to unique components than all mentions of components, the recall is 0.90 as the classifier is able to find 9 out of 10 unique components as mentioned earlier. The recall values for state and transition condition are 0.49 and 0.37, respectively, which suggests the need for future improvements. In addition, we measured time taken for manually writing rules for model generation and writing rules with an aid of CST mentions identified by the classifier. While the manual writing of rules took 7 hours, using aid of CST mentions by classifier took 3 hours. Section VI discusses some possible solutions for improvement.

TABLE II: **Pacemaker test data results**

Type of model element	**Recall**	**Precision**	**F_1-measure**
Component	0.76	0.67	0.71
State	0.49	0.55	0.52
Transition condition	0.37	0.10	0.16

VI. Discussion

This section presents error analysis of our experimental results and discusses the limitations of our proposed approach.

While the inter-annotator agreements on component, state, and transition condition for Section 5 of pacemaker document are 0.86, 0.73 and 0.64 as shown in Table I, the F_1-measure from classifiers for component, state, and transition condition are 0.71, 0.52 and 0.16 respectively as shown in Table II. While it is theoretically possible for the F_1-measure of an extremely well-performing machine learning algorithm to approach or even exceed the F_1-measures of human work, this is rare in practice. In this context, we can consider the results from our study to be a good start for future research. However, results leave ample room for improvement. So, we performed error analysis.

We found that the difficulty in recognizing patterns is mainly due to three major reasons in addition to the highly imbalanced nature of the data. The first reason is that while the entire document has only 1330 unique tokens, 203 of them do not have word embeddings. Majority of the tokens include the tokens that we labeled as states. In addition, most of the transition conditions have these tokens that results in a lack of recognition of appropriate patterns. This emphasizes the need to use word embeddings trained on requirements documents, which is also necessary due to the difference between semantics of textual documents and requirements documents. The second most common cause of errors that we found is annotating text in the tables in requirements document. We identified numerous states in the tables throughout the requirements. However, entries in the tables do not have context and therefore are difficult for the machine learning algorithm to learn from or to classify correctly. We plan to address this issue by considering text in tables separately and identifying mentions in them after finding elements in the text. The third most common cause of errors is the lack of proper representative samples in training data. This is because of inconsistent use of bullet and numbered lists as well as special symbols in test data that are not present in the training data. This reason affected the results of transition conditions. We plan to address this issue by using pre-processing techniques. We observed that the time taken for RNN with LSTM to learn is quite long. However, this is not an issue for the requirements engineers who would use the final pre-trained models, since applying the model to new data is very fast.

On the other hand, one limitation of the proposed approach is that it does not perform well on documents from other domains. However, by performing domain adaptation techniques and generating word embeddings trained on requirements documents written in various styles, it will be possible to address this limitation. Another limitation of the approach is that extracted mentions are not free from ambiguity or incompleteness if the original requirements are incomplete or ambiguous. We can mitigate the issue by analyzing requirements for ambiguity before finding mentions.

VII. Related Work

In this section, we discuss existing work related to natural language processing, mainly focusing on using NLP techniques to extract some sort of meaningful structure from natural language requirements, which are typically captured, at least initially, in an unstructured fashion [2].

Bajwa et al. [26] worked with natural language specifications and ultimately generated Alloy specifications via a series of model transformations. However, for the approach to work, the natural language had to be in a specific structure; namely, one that mapped easily to a Semantic Business Vocabulary and Rules (SBVR) representation.

Another method for providing structure to natural language requirements was created by Sharma et al. [27], who used the Stanford POS tagger [28] and parser [29] to extract grammatical knowledge patterns (GKP), which were then loaded into frames. The frame information was then used to generate UML activity diagrams and sequence diagrams. However, one of the limitations of this work is that requirements documents typically contain redundancies and ambiguities, which would be reflected in any diagrams generated using this method. Because our ultimate goal is requirements analysis, redundancies and ambiguities would be obstacles to such analysis.

Mu et al. [30] applied a set of rules to documents analyzed by the Stanford parser to create what they called an Extended Functional Requirements Framework (EFRF). An EFRF includes semantic cases such as action, goal, and constraint. Items were classified based on dependency analysis and sentence structure; e.g., actively voiced sentences were analyzed differently from passively voiced sentences. The framework performed well in identifying agents and actions but performed poorly in identifying location and temporal information. While the attempt to provide some semantic information is helpful, the semantic analysis would need to be more reliable to be suitable for analysis purposes.

Zeni et al. [31] used semantic annotation to create requirements models with legal documents. A tool called GaiusT was created, which builds on a previous tool the authors created called Cerno [32]. GaiusT semi-automatically generated elements of requirements models, such as actors and obligations. The tool was applied to legal documents in English as well as Italian to generate a framework for compliance that can later be applied to information technology systems. For the English text, results for GaiusT compared favorably with the results of expert annotators and appeared to improve the performance of novice annotators, as well as reduce the required time to edit annotations. While the results were impressive, the tool relies on a previously constructed ontology for its effectiveness.

VIII. Conclusions and Future Work

In this paper, we proposed an approach that helps users find relevant CST mentions in natural language requirements by running a recurrent neural network (RNN) classifier with long short-term memory (LSTM). Our proposed approach can identify components to an acceptable standard (0.76% recall),

with states and transition conditions needing further study to bring them to the level of component recognition.

We intend to improve the results by considering other features in addition to words as well as by using word embeddings which are trained on requirements documents. We intend to explore domain adaptation techniques for the proposed approach as a part of future work in order to improve its generalization to the requirements documents of new systems. By extending our proposed approach, we want to find relevant CST mentions and then use semantic relations to automatically find the relations between model elements. This information can then be used to generate the model and analyze the requirements in a way that reduces the amount of human effort required and is also easy for stakeholders to analyze.

ACKNOWLEDGMENT

This work was supported, in part, by NSF CAREER Award CCF-1564238 to University of North Texas.

REFERENCES

[1] J. Martin, *An information systems manifesto*, ser. The James Martin books os computer systems and telecommunications. Prentice-Hall International, 1984.

[2] M. Luisa, F. Mariangela, and N. I. Pierluigi, "Market Research for Requirements Analysis Using Linguistic Tools," *Requirements Engineering*, vol. 9, no. 1, pp. 40–56, 2004.

[3] F. Friedrich, J. Mendling, and F. Puhlmann, "Process model generation from natural language text," in *Advanced Information Systems Engineering (CAiSE)*, 2011, pp. 482–496.

[4] T. Yue, L. C. Briand, and Y. Labiche, "aToucan: An automated framework to derive UML analysis models from use case models," *ACM Transactions on Software Engineering and Methodology*, vol. 24, no. 3, 2015.

[5] D. Popescu, S. Rugaber, N. Medvidovic, and D. M. Berry, "Reducing ambiguities in requirements specifications via automatically created object-oriented models," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5320 LNCS, 2008, pp. 103–124.

[6] D. Aceituna and H. Do, "Exposing the susceptibility of off-nominal behaviors in reactive system requirements," in *IEEE 23rd International Requirements Engineering Conference (RE)*, Aug 2015, pp. 136–145.

[7] D. Aceituna, H. Do, and S. Srinivasan, "A systematic approach to transforming system requirements into model checking specifications," in *International Conference on Software Engineering (ICSE), Software Engineering In Practice (SEIP)*, May 2014, pp. 165–174.

[8] N. G. Leveson, "The role of software in spacecraft accidents," *Journal of Spacecraft and Rockets*, vol. 41, no. 4, pp. 564–575, 2204.

[9] S. Mahmood and R. Lai, "Re-uml: A component-based system requirements analysis language," *The Computer Journal*, vol. 56, no. 7, pp. 901–922, 2013.

[10] A. Cimatti and S. Tonetta, "Contracts-refinement proof system for component-based embedded systems," *Science of Computer Programming*, vol. 97, no. 3, pp. 333–348, 2015.

[11] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *the 26th International Conference on Neural Information Processing Systems*, ser. NIPS'13. USA: Curran Associates Inc., 2013, pp. 3111–3119.

[12] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent Neural Network Regularization," *International Conference on Learning Representations (ICLR)*, no. 2013, pp. 1–8, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329

[13] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[14] R. Tadeusiewicz, "Neural networks: A comprehensive foundation," *Control Engineering Practice*, vol. 3, no. 5, pp. 746 – 747, 1995.

[15] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[16] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[17] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[18] S. Bengio and G. Heigold, "Word embeddings for speech recognition," in *the 15th Conference of the International Speech Communication Association, Interspeech*, 2014.

[19] K. Hacioglu, B. Douglas, and Y. Chen, "Detection of entity mentions occurring in english and chinese text," in *the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, ser. HLT '05. Stroudsburg, PA, USA: Association for Computational Linguistics, 2005, pp. 379–386.

[20] Boston Scientific, "Pacemaker system specification," Boston Scientific, Tech. Rep., 2007.

[21] D. Méry and N. K. Singh, *Trustable Formal Specification for Software Certification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 312–326.

[22] N. K. Singh, A. Wellings, and A. Cavalcanti, "The cardiac pacemaker case study and its implementation in safety-critical java and ravenscar ada," in *the 10th International Workshop on Java Technologies for Real-time and Embedded Systems*, ser. JTRES '12. New York, NY, USA: ACM, 2012, pp. 62–71.

[23] S. Teufl, D. Mou, and D. Ratiu, "Mira: A tooling-framework to experiment with model-based requirements engineering," in *21st IEEE International Requirements Engineering Conference (RE)*, July 2013, pp. 330–331.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[25] J. Misra, "Terminological inconsistency analysis of natural language requirements," *Information and Software Technology*, vol. 74, pp. 183–193, 2016.

[26] I. S. Bajwa, B. Bordbar, M. Lee, and K. Anastasakis, "Nl2 alloy: A tool to generate alloy from nl constraints," *Journal of Digital Information Managment*, vol. 10, no. 6, pp. 365–372, 2012.

[27] R. Sharma, S. Gulia, and K. K. Biswas, "Automated generation of activity and sequence diagrams from natural language requirements," *9th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE*, pp. 69–77, 2014.

[28] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer, "Feature-rich part-of-speech tagging with a cyclic dependency network," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, ser. NAACL '03. Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 173–180. [Online]. Available: http://dx.doi.org/10.3115/1073445.1073478

[29] D. Chen and C. D. Manning, "A fast and accurate dependency parser using neural networks," in *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 740–750.

[30] Y. Mu, Y. Wang, and J. Guo, "Extracting software functional requirements from free text documents," in *2009 International Conference on Information and Multimedia Technology, ICIMT 2009*, 2009, pp. 194–198.

[31] N. Zeni, N. Kiyavitskaya, L. Mich, J. R. Cordy, and J. Mylopoulos, "Gaiust: supporting the extraction of rights and obligations for regulatory compliance," *Requirements Engineering*, vol. 20, no. 1, pp. 1–22, 2015.

[32] N. Kiyavitskaya, N. Zeni, J. R. Cordy, L. Mich, and J. Mylopoulos, "Cerno: Light-weight tool support for semantic annotation of textual documents," *Data & Knowledge Engineering*, vol. 68, no. 12, pp. 1470–1492, 2009.