Finding Component State Transition Model Elements using Neural Networks: An Empirical Study

Kaushik Madala, Shraddha Piparia, Hyunsook Do, Renee Bryce
Computer Science and Engineering
University of North Texas
{kaushikmadala, shraddhapiparia}@my.unt.edu, {hyunsook.do, renee.bryce}@unt.edu

***Abstract*—Use cases are popular for writing specifications of a system. However, despite their semi-structured nature, it is often time consuming and error-prone to generate component state transition diagrams from use case documents as it is done manually. While attempts to automate model generation from requirements have increased with the advent of deep neural networks (DNNs), there are limited studies in which a neural network architecture successfully extracts information used to construct a component state transition diagram from use cases. In this paper, we investigate the effectiveness of four different neural network architectures using glove and dependency embeddings to find model elements of component state transition diagrams from use case descriptions. Our results from the study show that we may achieve performance equivalent to humans with F1-scores greater than 0.80 for each model element on test data.**

I. Introduction

Requirements analysis is an important development activity because it prevents the propagation of defects to later phases of the product development life cycle. Model-driven requirements analysis [1] is a widely used technique to find problems in requirements because it offers advantages such as supporting an easy transformation to different types of models and a clear understanding of the system. However, generating models is typically done manually and requires a lot of human effort and time. To address this problem, researchers [2], [3], [4], [5] have proposed approaches that automate the model generation process from requirements. For example, Robeer et al. [5] use an approach to automatically extract conceptual models from user stories using natural language processing based heuristics. However, most of these approaches use restricted set of natural language features such as restricted syntax or limited syntactic information and do not consider semantic information thereby affecting their applicability.

Recently, with the advent of deep neural networks (NNs) [6] and their applicability for various tasks in the field of natural language processing, NNs have also been applied to the areas of software engineering [7]. For example, in our previous work [8], we used recurrent neural network (RNN) with long short-term memory (LSTM) [9] to find mentions of model elements of component state transition (CST) diagrams from common natural language requirements. While the intention in our study was to automate the process of identifying model elements to reduce human effort in the long run, we learned from our results and error analysis that common natural language requirements can have complex sentences that can affect the performance of classifiers and that we need a large amount of data to solve the problem. The observations and lessons learned from our previous work led us to consider the following questions: (1) What are the trade-offs among different NN architectures when extracting model elements? (2) How do different NN architectures affect results when applied to unseen documents written in different styles such as passive voice and different sentence types? (3) Can simpler textual descriptions such as use case descriptions aid NNs in predicting CST model elements better when compared to complex common natural language requirements from a small amount of training data?

In recent years, the use of user stories and use case descriptions in industrial requirements documents has increased [10], [4], [2] because the user stories and use case descriptions are simpler than common natural language descriptions. Thus investigating the aforementioned questions provides valuable insights into how recent technological advances can be applied to the field of requirements engineering and the benefits researchers and practitioners may gain.

To investigate these questions, we conducted an empirical study by training classifiers using four use case documents and four neural architectures: feed forward NN (FF) [11], convolutional NN (CNN) [12], RNN with LSTM [9], and RNN with gated recurrent unit (GRU) [13] to identify mentions of model elements of CST diagrams in natural language text of use cases. Our evaluation on test data shows that *RNN with GRU* gives F1-scores greater than 0.8 for all model elements. From results of our study on test data and unseen data, we draw conclusions that NNs can learn faster and predict better when they are trained with simple sentences, and that one should choose an NN architecture and its features based on what model element they want to extract rather than applying same architecture to find all model elements for a given model.

The paper is organized as follows: Section II provides background information on CST diagrams and related work. Section III gives the details of our study, and Section IV gives results analysis, limitations and threats to validity. Section V discusses conclusions and future work.

978-1-5386-8404-7/18/$31.00 ©2018 IEEE
DOI 10.1109/AIRE.2018.00014

IEEE computer society

II. Background and Related Work

A. CST diagrams and its model elements

This section provides background information about component state transition (CST) model elements, which are identified using several NNs investigated in our empirical study. We consider three major elements of a component level state transition diagram [8] : components, their states, and transition conditions. In addition, we consider a model element called actor to find transition conditions that result in state transitions due to human actions. We identify these model elements and create component transition rules in the form of *Transition Condition: Component(Current state)* $\rightarrow$ *Component(Next state)*, which we use as input to our technique, causal component model (CCM) [14], [15]. Figure 1 illustrates an example of CST diagram. A component is part of a system to be developed and its states represent possible behaviors of the component.

In Figure 1, the component is 'button' and its states are 'not clicked' and 'clicked'. A transition condition in CST diagrams can be classified into two types: system and environmental transition conditions. System transition conditions refer to components' states that cause state transitions of other components. Environmental transition conditions refer to transitions caused by human interactions and entities that are not part of the system. In use cases, while the system transition conditions can be easily found based on components and states identified, environmental transition conditions are difficult to find, in particular, when training data is small. To find environmental transition conditions, we consider any external agents that interact with the system as actors. In Figure 1, the transition condition is 'user clicked button', which changes the state of the button. It is an environmental transition condition and the actor in this transition condition is 'user'.

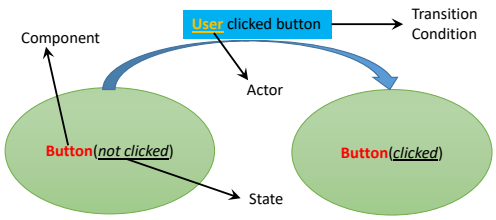

Fig. 1. Example of component state transition diagram

Using the model elements identified, we can write rules in the format we mentioned for requirements analysis.

B. Related Work

To reduce time and effort for creating models from requirements, many researchers have proposed various approaches that offer automated support. Due to the widespread use of UML that has become the de-facto standard modeling language, many automated model generation approaches use UML models [2], [5], [3], [4]. For example, Yue et al. [2] propose an approach that automatically creates class, sequence, and activity diagrams from restricted use case descriptions. Another example is the set of automated approaches by Elallaoui et al. [3], [4] to create use case diagrams and sequence diagrams from agile user stories.

There are other approaches that offer automated support for models other than UML. An approach by Friedrich et al. [16] automates creation of a business process model from the requirements. Bajwa et al. [17] propose an automatic approach to create alloy specifications, and Mu et al. [18] automate the generation process of the extended functional requirements framework. Further, a tool, GiausT developed by Zeni et al. [19], uses ontology and semantic annotations to automatically identify model elements from requirements.

To our knowledge, only one study investigated the use of NNs to generate models from requirements, which is done by our research group [8]. However, it uses only RNN with LSTM to identify CST model elements in common natural language requirements. In this paper, we study four different NN architectures to find CST model elements in simple use case descriptions rather than complex common natural language requirements, and compare their performance.

III. Empirical Study

To perform our empirical study, we implemented four different neural network architectures using the keras library [20] for training and testing the data in a computer with Windows 10 operating system and with 16 GB RAM. We investigate the following research questions:

- **RQ1:** What are the trade-offs among four NN architectures when extracting model elements?
- **RQ2:** How do different NN architectures affect results when applied to unseen documents written in different styles such as passive voice and different sentence types?
- **RQ3:** Can simpler textual descriptions such as use case descriptions aid NNs in predicting CST model elements better when compared to complex common natural language requirements from a small amount of training data?

The details of the study are as follows.

A. Dataset

We use the usecasedocs data set [21] that contains four use case documents. In addition, we use two more use case documents [22], [23] to analyze the performance of classifiers on unseen data, one written in passive voice [22] and another written in an industrial setting [23]. We only consider use cases that have model elements in their descriptions. We restrict our data to pre-conditions, basic flow, alternative flow, and post conditions sections of use cases as they provide information on behavior of the system. Table I presents the description of use case documents, the number of use cases per document, and whether the document is part of usecasedocs data set.

B. Metrics

We use the F_1-measure to evaluate the inter-annotator agreement as well as the performance of the classifiers. The F_1-measure is defined as the harmonic mean of precision (P) and Recall (R), which can be defined as follows:

TABLE I
DESCRIPTION OF USE CASE DOCUMENTS USED

ID	Description	No. of use cases	Part of usecase-docs?
UC1	Use cases for personalized health informatics (PHI) compliant system	21	Yes
UC2	Use cases for online shopping system	4	Yes
UC3	Use cases for automated guided vehicle system	2	Yes
UC4	Use cases for emergency monitoring system	4	Yes
UC5	An use case document on ambulance dispatch system [22]	10	No
UC6	Part of industrial use case document on system that offers assisted mobility for older and impaired users [23]	4	No

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R}$$

where P is the ratio of the number of correctly predicted mentions of a model element type to the total number of predicted mentions of that model element type and R is the ratio of the number of correctly predicted mentions of a model element type to the total number of mentions of the model element type in the gold standard. Here, "mention" of a model element refers to text span of the model element in a sentence. In this paper, we calculate lenient F_1 score, i.e., we consider even partial prediction of a model element as a prediction. For instance, for the model elements such as transition conditions that are phrases, as long as we find part of the phrase, we count it as we predicted the model element. In our preliminary analysis of data, we found that components and actors are usually one word or noun-stacks and states are usually one word, so partial predictions can still provide good enough information about these model elements. In the case of transition conditions, while partial predictions might miss information, transition conditions can be determined using post analysis (e.g., finding the phrase with the prediction using chunking [24]).

Trade-offs between recall and precision: While a high recall value indicates that there are only few missing model elements, a high precision value indicates that most of the model elements predicted are actual model elements. Because it is easier to eliminate predictions that are not model elements than to search for missing model elements in the requirements documents, our goal is to achieve 100% recall with the highest precision possible.

C. Experimental Procedure

Figure 2 illustrates our experimental procedure. The ovals represent the processes and the rectangles represent the inputs and outputs associated with the processes. The numbers over ovals are step numbers. We describe each step in detail. Before we conducted the experiments, we performed the preliminary analysis of data to come up with baselines and to check if lenient F_1 score is a suitable metric.

1) Annotate the documents and generate labeled data: Because we aim to find model elements that are words or phrases in a sentence written in natural language, we consider our task as a sequence labeling task. In the sequence labeling task, rather than having a single label for the entire sentence, each word in the sentence has a label (e.g., parts of speech tagging). A sequence labeling task is similar to a classification task in machine learning, and any classification task requires a labeled dataset. Hence, the first step in our approach generates a labeled dataset. In general, the labeled dataset can be generated from text mentions of model elements identified by one or more people, where text mentions are the spans of text in natural language that represent the elements of interest. However, text machine learning (ML) and natural language processing (NLP) communities require at least two people to go through a document and identify text mentions to generate a labeled dataset [25]. This is because text mentions identified by a single person can be biased and error-prone. Text mentions identified by more than one person, can not only reduce a bias by giving multiple perspectives but also can reduce the error-proneness. The people who identify the text mentions are called annotators and the process is called annotation. In this work, two graduate students annotated the documents and a third graduate student helped to resolve conflicts.

Before annotating the mentions of model elements, the annotators need to setup the guidelines such as what the definitions of model elements are, and whether certain features around model elements should be considered as part of model elements or not. After setting up guidelines, the annotators label the data separately, and compare the results. To find their level of agreement, we find inter-annotator agreement. We use F_1-measure to find our inter-annotator agreement. To find the F_1-score, which we refer as an annotation score, we treat one person's annotations as the ground truth and compare them with other person's annotations. According to the machine learning and natural language processing communities [25], [26], if an algorithm produces a value close to inter-annotator agreement, it means that the algorithm is performing similar to human.

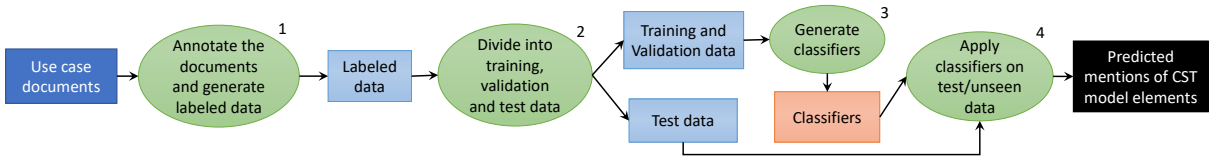

Fig. 2. Overview of approach

Once the inter-annotator agreement is found, the annotators discuss their annotated mentions and finalize the labels to be put into the gold standard. If there is a disagreement between two annotators, a third person resolves the conflict. The gold standard is the text mentions, which we use to generate our labeled dataset.

Because the text mentions of model elements can be words and phrases, we use BIO tags [27] to label the data. As transition conditions can have other model elements in it, we generate labeled data for each model element separately. BIO tags for each model element represent the beginning (B), inside (I), and outside (O) of text mentions of the model element. The usage of BIO tags for labeling modeling elements for the following sentence, "When user presses the switch, the motor runs," is illustrated in Figure 3.

Sentence:	When	user	presses	the	switch	,	the	motor	runs
Component labels	O	O	O	O	B	O	O	B	O
State labels	O	O	O	O	O	O	O	O	B
Transition condition labels	B	I	I	I	I	O	O	O	O
Actor labels	O	B	O	O	O	O	O	O	O

Fig. 3. Example of using BIO tags

2) Divide into training, validation, and test data: Once the labeled dataset is generated, we partition the use case documents into training, validation, test, and unseen data (data which is new and not a part of data set from which test data is created). Because the data size is small and the neural networks need to know information on the topic to predict, we divide each document UC1–UC4 in the ratio of 70:10:20 use cases and combine the respective proportions together to generate training, validation, and test data. As a result, we group 17 use cases from UC1, two from UC2, one from UC3, and two from UC4 as training data. We group one use case from each of UC1, UC2, and UC4 to generate validation data, and group the remaining use cases from the documents into test data. We use UC5 and UC6 as unseen data to check if the classifiers work well on use cases written in passive voice and industrial use cases, respectively. We recalculate the inter-annotation agreement based on divisions of the data sets. Due to space limitations, we only illustrate the annotation scores of testing and unseen data in Table II, which are the values we use to compare effectiveness of NNs. As mentioned in Section III-C1, to find annotation scores, we consider the first person's annotations as the ground truth and compare second person's annotations with them. The values of precision, recall and lenient F_1-score in Table II illustrates the level of agreement the annotators have for each data. We can observe from the table that the model element 'Component' has a low annotation score of 0.41. This difference is due to the fact that two annotators have different perspectives when they identify components. For example, one annotator identified 'route' as a component where as the other annotator considered it as a property of component 'app.'

TABLE II
ANNOTATION SCORES OF TEST AND UNSEEN DATA

Data	Model element	Inter-annotator agreement		
		Recall (R)	Precision (P)	F_1
Test	Component	0.9643	0.7714	0.8571
	State	1.0	0.7714	0.8710
	Actor	0.9643	0.8710	0.9153
	Transition Condition	0.9231	0.8571	0.8889
UC5	Component	0.5410	0.9167	0.6804
	State	0.7778	0.7568	0.7671
	Actor	0.9608	0.9608	0.9608
	Transition Condition	0.7368	0.9032	0.8116
UC6	Component	0.3594	0.4894	0.4144
	State	0.5278	0.8261	0.6441
	Actor	1.0	1.0	1.0
	Transition Condition	0.7500	0.6723	0.7200

3) Generate classifiers: Next, we train classifiers for each model element using training data. We also evaluate baselines for each model element except for transition conditions, because transition conditions are diverse and are difficult to have a simple baseline. We used two baselines for components, actors, and states, respectively. For components, 'baseline 1' considers all nouns to be components whereas 'baseline 2' considers only the first noun in a sentence to be a component. For actors, we have baselines that are same as components, as most actors are also nouns. For states, the 'baseline 1' considers any adjective or verb to be a state whereas 'baseline 2' considers the first verb occurred in a sentence to be a state. We formed the baselines based on the patterns we found during preliminary analysis on training data.

We evaluated the performance of the widely used mallet simple tagger (MST) [28] on all our model elements and compared the results with NNs. We consider this technique to be a reasonable baseline as it considers syntactical information when predicting elements.

The different types of NN architectures on which we trained the data are:

- **A simple feed forward (FF) neural network:** A fully connected multi-layer perceptron model [11].
- **Convolutional neural network (CNN):** A feed forward NN that uses convolutional layers instead of fully connected layers [12].
- **RNN with long short-term memory (LSTM):** A recurrent network that takes feedback from previous words in text and uses multiple memory cells [9].
- **RNN with gated recurrent unit (GRU):** A recurrent network similar to LSTM but does not store multiple memory cells [13].

We used two types of word embeddings as features for NNs:

- **glove embeddings** [29] which has context related semantic information embedded into them.
- **dependency embeddings** [30] whose semantic information considers their dependency type than strictly context in the sentence.

As neural networks take only numerical data as input, we convert all of the words into the numerical indexes by mapping every word to the its 'ID' in a word embedding dictionary. For example, let us consider a word embedding dictionary that has three indexes {0, 1, 2} which refers to three words {the, of, is}. Each of these indexes has corresponding vectors that contain the embedding values. If we have the word 'the' in the sentence, it is mapped to its 'ID', which is index '0'. We use the embedding layer as part of the neural network which automatically replaces word indexes with word embeddings. We convert the labels to their respective one-hot vector representation [8]. The neural networks use the word embeddings as features and one-hot vector labels to learn the prediction of mentions of model elements. However, how do we know if our neural network performs efficient predictions? To check its efficiency, as well as, to find possible best fit for training data, we use validation data to tune parameters such as the number of hidden layers, the size of hidden layers, the learning rate, the number of epochs, the type of optimizer, and the loss function. The set of parameters that perform well on validation data are finalized. The classifiers trained using these parameters are finalized and used to evaluate test and unseen data.

4) Apply classifiers on test/unseen data: The finalized classifiers are applied on test data to evaluate their efficiency. We compare the predicted values with the labels from the gold standard. We can use the predicted mentions of the model elements to create rules that can be used for requirements analysis using CCM [14].

D. Results

The results of the techniques used to identify the mentions of components, states, actors and transition conditions in use case descriptions are shown in Tables III, IV, V and VI, respectively.

RQ1 results: Overall, we observe from these tables that *RNN with GRU* works better in finding model elements in use case descriptions including the descriptions written in different styles compared to other NNs. However, some model elements such as components and actors are predicted well by using *dependency* word embeddings whereas other model elements such as transition conditions and states are predicted well using *glove* embeddings. This result could be attributed to the nature of the model elements. The model elements performing better when using *dependency* word embeddings signify that these model elements are dependent on their role in the sentence, whereas the model elements performing better when using *glove* embeddings signify that these model elements are more dependent on the context than the roles in the sentence.

RQ2 results: From Table III, we observe that all neural networks outperform baselines 1 and 2 when predicting components. Of all the NN architectures, *RNN with GRU* with *dependency* embeddings predicted components with the highest scores in test data and UC6. The F_1 score by this architecture on test data (0.8257) is close to its human inter-annotation agreement (0.8571), and the F_1 score by this architecture on UC6 (0.4835) outperformed its human inter-annotation agreement (0.4144). In the case of UC5, *RNN with LSTM* trained on *dependency* embeddings yielded the highest scores, whose F_1 score (0.6739) is close to its inter-annotation agreement (0.6804). MST outperformed most of neural network architectures, but it was not better than *RNN with GRU* trained on *dependency* embeddings in test and UC6 and *RNN with LSTM* trained on dependency embeddings in UC5.

The results of model element *states* in Table IV show that the NN architectures outperformed the baselines on test data. In particular, *RNN with GRU* trained on glove embeddings yielded the best results. However, *baseline 1* outperformed other techniques in UC5 and *baseline 2* outperformed other techniques in UC6. The written styles in UC5 and UC6 use passive voice and imperative sentences, respectively, which affected the outcome as they change the dependencies and contexts. MST is outperformed by most of NN architectures on test data. However, it outperformed many NN architectures when applied to UC5 and UC6. Within NN architectures, *RNN with GRU* trained on *glove* embeddings outperformed other NN architectures in test and UC6, whereas *RNN with LSTM* trained on *glove* embeddings outperformed other NN architectures in UC5. When compared to inter-annotation agreement, the highest score produced by *RNN with GRU* trained using *glove* embeddings on test data (0.8615) is very close to that of inter-annotation agreement of states in test data (0.8710). However, the highest scores by *baseline 1* in UC5 (0.3283) and *baseline 2* (0.4348) in UC6 need to be improved to reach their corresponding inter-annotator agreements (0.7671 and 0.6441).

Table V illustrates the results of actors on test data, UC5, and UC6. The results show that all NN architectures outperform baselines and MST in test data. *RNN with GRU* trained on *dependency* embeddings produced the highest scores in test data (0.8947) and UC6 (0.9014) whereas *RNN with GRU* trained on *glove* embeddings produced the highest score in UC5 (0.6097). The results also show that most values for MST, FF, CNN, and some of LSTM architecture values are zeroes. We attribute these results to the sparsity and diversity of the representatives of actors in training data. When compared to the inter-annotation agreement, the highest score by RNN with GRU trained using dependency embeddings on test data (0.8947) and on UC6 (0.9014) are very close to their corresponding inter-annotation agreements on actors (0.9153 and 1.0). While the highest score by RNN with GRU trained using glove embeddings on UC5 is 0.6097, it needs to be improved to reach its inter-annotator agreement (0.9608).

The results of transition conditions are shown in Table VI. The results show that MST was outperformed by only RNN with LSTM and RNN with GRU architectures in test data but was outperformed by all NN architectures in UC5 and UC6. For test data, RNN with GRU trained on glove embeddings has the highest score (0.9047) followed by RNN with GRU trained on dependency embeddings and RNN with LSTM trained on glove embeddings with a score of 0.9, respectively. For UC5 and UC6, out of all NN architectures RNN with LSTM trained on glove embeddings produced the highest scores, respectively (0.6744 and 0.6279).

The highest scores on test data by RNN with GRU (trained

on glove and dependency embeddings with scores 0.9047 and 0.9 respectively) and RNN with LSTM (trained on glove embeddings with score of 0.9) outperformed the corresponding inter-annotator agreement (0.8889). In the case of UC5 and UC6, the highest scores by RNN with LSTM trained using glove embeddings are 0.6744 and 0.6279, which have a difference of 0.1 in the F_1-score when compared to their corresponding inter-annotation scores (0.8116 and 0.72, respectively).

RQ3 results: We compared our results on simple use case descriptions with the results on common natural language requirements obtained from our previous work [8]. In our previous work, the results of components, states, and transition conditions on test data using RNN with LSTM trained on small data were 0.71, 0.52, and 0.16, respectively, while in this study, RNN with LSTM produced scores of ≈ 0.75, ≈ 0.7 and ≈ 0.9 for components, states and transition conditions. Therefore, we can conclude that simple textual descriptions can provide better results for a small amount of training data for NNs.

To understand the human efforts required to find model elements, the time taken for pre-processing and how NN architectures perform in terms of their execution time, we measured the time required for annotating documents, pre-processing, and training NNs. It took 22 hours for producing guidelines, annotating, and adjudicating for all documents (UC1–UC6). Because our pre-processing techniques are automated, it took only 16 seconds in total. For training architectures, NNs took less time when trained with *dependency* embeddings as features. This is because of the small dictionary size of *dependency* embeddings compared to *glove*. CNN required only 8 minutes to run, but the rest of architectures required around 20 minutes. When trained with glove embeddings, NNs took longer (almost doubled). While CNN took around 13 minutes, FF took 36 minutes, RNN with GRU took 39 minutes, and RNN with LSTM took 41 minutes. The time taken for testing data for all NNs is around 0.06 seconds.

IV. Discussion

The results of our empirical study show that when NNs are trained on simple sentences such as use case descriptions, they can predict model elements with high F_1 scores even with a small amount of data unlike the complex common natural language requirements [8]. Therefore, we can conclude that the simplicity of sentences can affect the learning ability of NNs and the requirements written in a simple structure can make automation easier. Contrary to a common belief that a same set of features and specific NN architecture must be used for all model elements to generate a model, our results show that we need to consider features and NN architecture based on the semantics and labeling style of a model element. We found that *RNN with GRU* with *dependency* embeddings can predict components and actors with F_1 scores greater than 0.8. For other model elements such as states and transition conditions that are mainly dependent on their context in the sentence, *RNN with LSTM* and *GRU* with *glove* embeddings predicted with F_1 scores greater than 0.8. For UC5, which is written in passive voice, *RNN with LSTM* produced the highest F_1 score (0.6739) for components whereas for UC6, *RNN with GRU* produced the highest F_1 score (0.4835) for components. These results indicate that different NN architectures predict well for documents written in different styles such as UC5 and UC6.

The results show that most of the outperforming NN architectures produced good precision and recall values, which result in a balanced model and require less human effort to fill in the rest of the model. Hence, we can use the model elements predicted by such NN architectures to find incomplete requirements and to analyze requirements using semi-automatic analysis techniques such as CCM [14] and FFIP [31]. We also found that *RNN with GRU* and *RNN with LSTM* architectures can perform close to or sometimes outperform the inter-annotator agreement.

We train classifiers separately for each model element and the prediction results of each model element can vary with the type of NN architecture and features used. Thus, we use different NN architectures and features for the classifiers of model elements. We combine the classifiers after collecting predictions from each classifier (as shown in Figure 4) by storing a 4-dimensional vector for each word in a sentence, where each dimension has one of the IOB labels and represents a single model element.

Limitations: Overall, we obtained good scores on test data, but the results indicate that there is room for improvement. In particular, we need to improve the scores of states and actors for UC5 and UC6, which are written in passive voice and industrial setting, respectively. We can improve the scores by considering domain adaptation techniques, combining different NN architectures, or using multi-task learning [32] that considers all model elements occurring in a sentence. The results of transition conditions need to be slightly improved for UC5 and UC6 to reach inter-annotation agreement. We can use the causal embeddings [33] to improve the results.

Threats to validity: One major threat is that the model elements identified can differ from annotators. We resolved this threat by having two annotators to consider multiple perspectives. Moreover, if there is a disagreement between annotators, a third expert helped to resolve the issue and made the decision. Another threat is that we cannot generalize our results to different data sets and requirements documents written using a common natural language or written in a different style (e.g., requirements written with different sentence structures or complex requirements). We plan to address this threat by performing experiments on industrial requirements documents and on documents written using a common natural language and in different styles.

V. Conclusion and Future Work

In this paper, we conducted an empirical study on four neural network architectures to find the mentions of model elements of CST diagrams in use case descriptions. Our results show that NNs can learn well with small data sets when used simple sentences such as user stories and use case descriptions. This makes the use of NNs more practical in industry where the use of user stories and use case descriptions has become more

TABLE III
RESULTS OF COMPONENT

Technique	Embeddings	Test			UC5			UC6		
		Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
Baseline 1		0.7407	0.2325	0.3539	0.8928	0.1858	0.3077	0.7968	0.1288	0.2217
Baseline 2		0.3148	0.4146	0.3579	0.125	0.1428	0.1333	0.2813	0.2857	0.2835
MST		0.8864	0.75	0.8125	0.4464	0.8621	0.5882	0.6571	0.3538	0.4599
FF	Glove	0.7593	0.6949	0.7256	0.4107	0.6216	0.4946	0.3334	0.4773	0.3925
	Dependency	0.7407	0.7692	0.7547	0.4107	0.8518	0.5542	0.3334	0.7241	0.4565
CNN	Glove	0.7222	0.65	0.6842	0.4107	0.7187	0.5227	0.3492	0.6875	0.4632
	Dependency	0.7222	0.7647	0.7428	0.4107	0.8518	0.5542	0.3334	0.6774	0.4468
RNN with LSTM	Glove	0.7407	0.7692	0.7547	0.4285	0.7742	0.5517	0.3809	0.5854	0.4615
	Dependency	0.7963	0.7818	0.7889	0.5536	0.8611	0.6739	0.3651	0.5476	0.4381
RNN with GRU	Glove	0.7962	0.7678	0.7818	0.4821	0.7714	0.5934	0.3651	0.5	0.4221
	Dependency	0.8333	0.8181	0.8257	0.4464	0.8333	0.5814	0.3492	0.7857	0.4835

TABLE IV
RESULTS OF STATE

Technique	Embeddings	Test			UC5			UC6		
		Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
Baseline 1		0.8	0.2593	0.3916	0.9167	0.2	0.3283	0.9117	0.1215	0.2146
Baseline 2		0.28571	0.5	0.3636	0.02778	0.04545	0.0345	0.4412	0.4286	0.4348
MST		0.4857	0.85	0.6182	0.1944	0.5833	0.2916	0.2647	0.3214	0.2903
FF	Glove	0.5143	0.75	0.6102	0.1667	0.4	0.2353	0.0588	0.0869	0.0701
	Dependency	0.4857	1	0.6538	0.0833	0.6	0.1463	0.02941	0.0909	0.0444
CNN	Glove	0.5428	0.6552	0.5937	0.1389	0.3334	0.1961	0.0294	0.0434	0.0351
	Dependency	0.5714	0.7692	0.6557	0.1389	0.625	0.2272	0.0294	0.1	0.0454
RNN with LSTM	Glove	0.5142	1	0.6792	0.2222	0.5714	0.3200	0.1176	0.4	0.1818
	Dependency	0.5428	1	0.7037	0.0833	0.25	0.125	0.0294	0.1111	0.0465
RNN with GRU	Glove	0.8	0.9334	0.8615	0.25	0.3103	0.2769	0.3529	0.3428	0.3478
	Dependency	0.6	0.9545	0.7368	0.1389	0.7143	0.1389	0.2353	0.4445	0.3076

TABLE V
RESULTS OF ACTOR

Technique	Embeddings	Test			UC5			UC6		
		Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
Baseline 1		0.9474	0.0967	0.1756	0.9230	0.1784	0.2990	1	0.1010	0.18348
Baseline 2		0.4737	0.225	0.3051	0.3269	0.2698	0.2956	0.675	0.4426	0.5346
MST		0	0	0	0	0	0	0	0	0
FF	Glove	0.7368	0.7368	0.7368	0	0	0	0	0	0
	Dependency	0.8421	0.8421	0.8421	0	0	0	0	0	0
CNN	Glove	0.7895	1	0.8824	0	0	0	0	0	0
	Dependency	0.8421	0.8421	0.8421	0	0	0	0	0	0
RNN with LSTM	Glove	0.7368	0.8235	0.7778	0.0385	1	0.0741	0.1026	1	0.1860
	Dependency	0.8421	0.8889	0.8648	0	0	0	0	0	0
RNN with GRU	Glove	0.8421	0.8421	0.8421	0.4807	0.8333	0.6097	0.8461	0.9167	0.88
	Dependency	0.8947	0.8947	0.8947	0.0577	1	0.1091	0.8205	1	0.9014

TABLE VI
RESULTS OF TRANSITION CONDITION

Technique	Embeddings	Test			UC5			UC6		
		Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
MST		0.7778	0.8235	0.8001	0.1471	0.5	0.2273	0.0625	0.2857	0.1025
FF	Glove	0.8421	0.2712	0.4103	0.7567	0.2667	0.3943	1	0.3908	0.5620
	Dependency	0.7368	0.3111	0.4375	0.7027	0.4262	0.5306	0.6764	0.2738	0.3898
CNN	Glove	0.8947	0.2982	0.4474	0.9189	0.3148	0.4689	0.9412	0.2013	0.3316
	Dependency	1	0.2567	0.4086	0.9189	0.3178	0.4722	0.9706	0.1675	0.2857
RNN with LSTM	Glove	0.9474	0.8571	0.9	0.7838	0.5918	0.6744	0.7941	0.5192	0.6279
	Dependency	0.9473	0.7826	0.8571	0.6756	0.6097	0.6410	0.6764	0.5	0.575
RNN with GRU	Glove	1	0.8260	0.9047	0.8108	0.5357	0.6451	0.9117	0.4696	0.62
	Dependency	0.9474	0.8571	0.9	0.5405	0.5882	0.5634	0.3235	0.4231	0.3667

common. We also found that components and actors can be predicted with F_1 scores greater than 0.8 using *RNN with GRU* trained with *dependency* embeddings and transition conditions and states can be found with higher F_1 scores by using *RNN with LSTM* with *glove* embeddings. As future work, we plan to address the limitations discussed in Section IV. We aim to improve our results using causal embeddings and multi-task learning. We also plan to extend our study to complex common

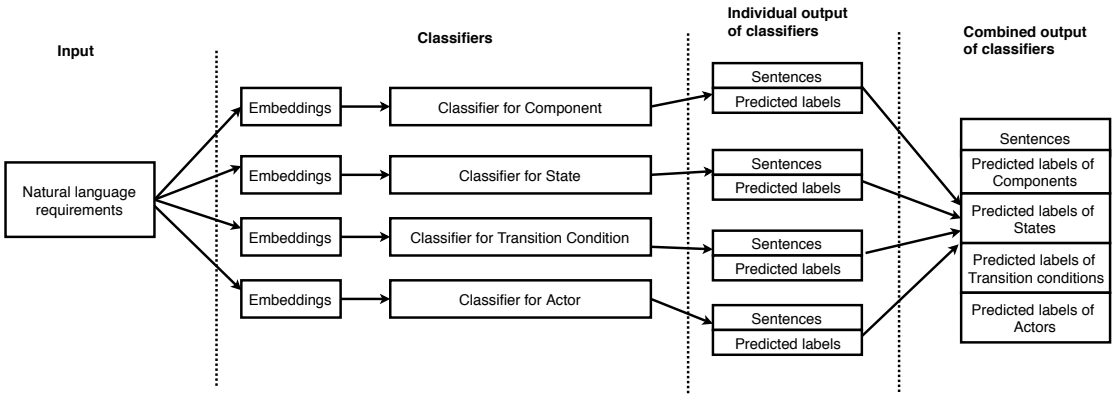

Fig. 4. Recommended architecture for combining classifiers

natural language requirements.

VI. Acknowledgment

We would like to thank Dr. Eduardo Blanco for his feedback and suggestions on our experiments. This work was supported, in part, by NSF CAREER Award CCF-1564238 to University of North Texas.

References

[1] B. Baudry, C. Nebut, and Y. L. Traon, "Model-driven Engineering for Requirements Analysis," in *EDOC 2007*. IEEE, 2007, pp. 459–459.

[2] 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.

[3] M. Elallaoui, K. Nafil, and R. Touahni, "Automatic generation of UML sequence diagrams from user stories in Scrum process," in *SITA'15*, Oct 2015, pp. 1–6.

[4] M. Elallaoui, K. Nafil, and R. Touahni, "Automatic transformation of user stories into UML use case diagrams using NLP techniques," *Procedia Computer Science*, vol. 130, pp. 42 – 49, 2018, aNT 2018 / SEIT-2018 / Affiliated Workshops.

[5] M. Robeer, G. Lucassen, J. M. E. van der Werf, F. Dalpiaz, and S. Brinkkemper, "Automated extraction of conceptual models from user stories via NLP," in *RE'16*. IEEE, 2016, pp. 196–205.

[6] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *ICML'08*. ACM, 2008, pp. 160–167.

[7] X. Li, H. Jiang, Z. Ren, G. Li, and J. Zhang, "Deep Learning in Software Engineering," *arXiv preprint arXiv:1805.04825*, 2018.

[8] K. Madala, D. Gaither, R. Nielsen, and H. Do, "Automated Identification of Component State Transition Model Elements from Requirements," in *REW '17 (AIRE '17)*, Sept 2017, pp. 386–392.

[9] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent Neural Network Regularization," *ICLR'14*, no. 2013, pp. 1–8, 2014.

[10] 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.

[11] J. A. Hertz, *Introduction to the theory of neural computation*. CRC Press, 2018.

[12] B. Hu, Z. Lu, H. Li, and Q. Chen, "Convolutional neural network architectures for matching natural language sentences," in *Advances in neural information processing systems*, 2014, pp. 2042–2050.

[13] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *ICML'15*, 2015, pp. 2342–2350.

[14] D. Aceituna and H. Do, "Exposing the susceptibility of off-nominal behaviors in reactive system requirements," in *RE'15*, Aug 2015, pp. 136–145.

[15] K. Madala, H. Do, and D. Aceituna, "A combinatorial approach for exposing off-nominal behaviors," in *40th International Conference on Software Engineering (ICSE 2018)*, 2018.

[16] F. Friedrich, J. Mendling, and F. Puhlmann, "Process Model Generation from Natural Language Text," in *Advanced Information Systems Engineering (CAiSE)*, 2011, pp. 482–496.

[17] 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.

[18] Y. Mu, Y. Wang, and J. Guo, "Extracting software functional requirements from free text documents," in *ICIMT '09*, 2009, pp. 194–198.

[19] 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.

[20] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[21] S. Liu, J. Sun, Y. Liu, Y. Zhang, B. Wadhwa, J. S. Dong, and X. Wang, "Automatic Early Defects Detection in Use Case Documents," in *ASE'14*. New York, NY, USA: ACM, 2014, pp. 785–790.

[22] "Ambulance Dispatch System Requirements Specification." [Online]. Available: http://www.utdallas.edu/ chung/CS6354/CS6354_U07_source/Team_2/delivarable_2_final.doc

[23] "SIMON: D2.1 Use case Specification Document," 2014. [Online]. Available: http://simon-project.eu/wp-content/uploads/2014/02/simon_D2_1_Use-case-specification-document_PU_v1.0.pdf

[24] E. F. Tjong Kim Sang and S. Buchholz, "Introduction to the CoNLL-2000 Shared Task: Chunking," in *Proceedings of the 2Nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7*, ser. ConLL '00. Stroudsburg, PA, USA: Association for Computational Linguistics, 2000, pp. 127–132.

[25] R. Artstein, "Inter-annotator agreement," in *Handbook of linguistic annotation*. Springer, 2017, pp. 297–313.

[26] R. J. Passonneau, T. Yano, T. Lippincott, and J. Klavans, "Relation between Agreement Measures on Human Labeling and Machine Learning Performance: Results from an Art History Domain," *Computational Linguistics for Metadata Building*, vol. 49.

[27] K. Hacioglu, B. Douglas, and Y. Chen, "Detection of Entity Mentions Occurring in English and Chinese Text," in *HLT '05*, Stroudsburg, PA, USA, 2005, pp. 379–386.

[28] A. K. McCallum, "MALLET: A Machine Learning for Language Toolkit," 2002. [Online]. Available: http://mallet.cs.umass.edu

[29] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global Vectors for Word Representation," in *EMNLP'14*, 2014, pp. 1532–1543.

[30] O. Levy and Y. Goldberg, "Dependency-based word embeddings," in *ACL '14*, vol. 2, 2014, pp. 302–308.

[31] T. Kurtoglu and I. Y. Tumer, "A graph-based fault identification and propagation framework for functional design of complex systems," *Journal of Mechanical Design*, vol. 130, no. 5, p. 051401, 2008.

[32] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[33] R. Sharp, M. Surdeanu, P. Jansen, P. Clark, and M. Hammond, "Creating causal embeddings for question answering with minimal supervision," in *EMNLP '16*, 2016, pp. 138–148.