Extracting Conceptual Interoperability Constraints from API Documentation using Machine Learning

Hadil Abukwaik[1], Mohammed Abujayyab[2], Shah Rukh Humayoun[1], Dieter Rombach[1]
University of Kaiserslautern
67663 – Kaiserslautern, Germany
[1]{abukwaik, humayoun, rombach}@cs.uni-kl.de,[2]abujayya@rhrk.uni-kl.de

ABSTRACT

Successfully using a software web-service/platform API requires satisfying its conceptual interoperability constraints that are stated within its shared documentation. However, manual and unguided analysis of text in API documents is a tedious and time consuming task. In this work, we present our empirical-based methodology of using machine learning techniques for automatically identifying conceptual interoperability constraints from natural language text. We also show some initial promising results of our research.

1. INTRODUCTION

Conceptual **i**nteroperability co**n**straints (**COINs**) are restrictions on interoperable software units and their related data elements at different conceptual levels (i.e., syntax, semantics, structure, dynamics, context, and quality) [1]. For successful interoperations, such constraints need to be identified and fulfilled. Otherwise, they may cause conceptual mismatches that hinder the interoperation or produce meaningless results, and consequently lead to expensive resolution at late project stages. Therefore, third-party clients need to effectively analyze the shared documentation of external APIs. However, manual sifting of natural language (NL) text within API documents is a tedious and time consuming task, which also requires lexical and linguistic analysis skills.

To cope with these challenges, we elaborate on Abukwaik et al. [1] ideas of extracting a complementary set of conceptual constraints from text in API documentation using machine learning (ML) and natural language processing (NLP) technologies. Our goal is to support analysts in performing the conceptual interoperability analysis effectively, while keeping the associated cost of identifying COINs low. In our work, we follow a systematic empirical-based methodology that has two advantages, i.e., tracing and verifying documented results between the research phases, and repeating the defined activities in our protocol by other researchers to address the bias threat to validity. The remainder of the paper explains our research methodology.

ICSE '16 May 14-22, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2889160.2892642

2. EMPIRICAL INVESTIGATION OF API DOCUMENTATIONS

In this phase, we explored the current state of COINs (in terms of representation and recurring patterns) within real-world API documentations, with the objective to find the potentials of automating their extraction. In order to get generalizable results, we performed a multiple-case study with literal replication of cases from different domains. The study protocol included the following three activities:

Case Selection: Our selection criteria were *content relevance* to research goal (i.e., document was not fully technical, but it contained conceptual constraints too) and *API popularity* according to published statistics[1]. We selected four cases from the *Web-service API* domain (i.e., Skype, GoogleMaps, SoundCloud, and Instagram) and two cases from the *Platform API* domain (i.e., AppleWatch and Eclipse).

Case Execution: For each case, we carried out three tasks: *1) Data Preparation* – We implemented a code using the PHP Simple HTML DOM Parser[2] library to preprocess the API documentation text by eliminating its noise (i.e., headers, images, pure code, etc.), breaking it into single sentences, and saving them in our predefined data extraction sheet. For time limitations, we selected inclusive parts of large API documents (e.g., the Plug-in part of Eclipse). *2) Data Collection* – We recorded the demographic information about each case (e.g., popularity score, developing company, development year, etc.) and some statistical information (e.g., total number of sentences). Also, we maintained a repository for the original API documents and the data extraction sheet. *3) Data analysis* – We manually examined each sentence in the extraction sheet and classified it with one of the COIN categories [1] if it applies; otherwise, we classified it as a Not-COIN category. For example, the sentence *"a user is encapsulated by a read-only Person object"* is classified as a Structure COIN. As the classification has critical effect on the subsequent phases, two of the authors replicated it, where inconsistencies were discussed and resolved based on consensus. Executing the six cases was tedious and it took about 239 person-hours.

Cross-Case Analysis: Finally, we aggregated the classified sentences from all cases into one output, which we call *COINs Corpus* and it represents the ground truth for the later tasks. Table 1 shows, in descending order, the number of instances for each COIN category found across the cases. The imbalance between the number of instances for

[1]Programmable Web: *www.programmableweb.com*
[2]Simple HTML DOM: *http://simplehtmldom.sourceforge.net*

Table 1: COIN instances in the investigated cases

COIN category	No. of instances
Not-COIN	960
Dynamic	570
Semantic	551
Structure	107
Syntax	57
Quality	25
Context	13

each category is related to the technical-orientation of the current API documentation. For example, the number of sentences describing the service usage context or quality attributes are typically less than these describing its semantic goal or dynamic process.

3. AUTOMATIC IDENTIFICATION OF COINS VIA MACHINE LEARNING CLASSIFIER

In this phase, we aim at automating the identification of COINs in NL text of API documents. To achieve this goal, we first built a feature model for our contributed ground truth data set (i.e., the *COINs Corpus*), then we utilized it within a ML classifier. Figure 1 outlines this phase activities, which we performed them all using the Weka version 3.7.12[3]. Note that, we adopted an experimental strategy to find the best combination of features and a ML classifier based on the accuracy of the classification results.

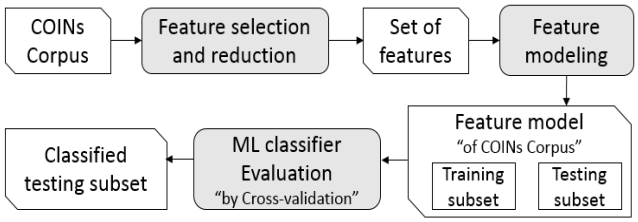

Figure 1: Using ML for COINs identification

Building the Feature Model of the COINs Corpus: To supervise the classifier learning, we started with the *feature selection* to identify the most significant and informative features of the corpus data set. Therefore, we selected the frequently occurring keywords and sentence structures using a number of NLP techniques (e.g., tokenizing, stopwords filtering, stemming, etc.). For example, the keyword "encapsulate" is a feature for the Structure COIN category. To enhance the classification performance, we applied the *feature reduction* based on the occurrence frequency of features using the N-Grams [7] model. Finally, we built the feature model of our COINs corpus as a matrix of features and COIN sentences. To weight the matrix cells, we used the Term Frequency-Inverse Document Frequency (TF-IDF) [6] that calculates the weight according to the feature frequency in the sentence and its importance across the whole corpus.

Classifier Selection: We experimented the accuracy of our feature model for the COINs Corpus within a number of classifier algorithms using a k-fold cross-validation [8] with $k = 10$ (i.e., the data set was divided into 10 subsets). For 10 runs, one subset was for testing and 9 subsets were for training the classifier. Finally, we took the average of the 10 runs. Table 2 shows that the Naive Bayes (NB) [4] and Support Vector Machine (SVM) [2] algorithms achieved the best accuracy results. These two algorithms have been proven to be the most effective text classifier algorithms [3].

Initial Results: For classifying 7 COIN categories, we have achieved accuracy f-measure of 62.2% for NB and 59.5% for SVM (see Table 2). We believe that these results are promising, knowing that we have a relatively small size corpus (i.e., less than 3K sentences). Also, limiting the classification categories to COIN and Not-COIN, the f-measure increases to 76% for NB and 72% for SVM (see Table 3).

Table 2: Initial results of classifying 7 COIN classes

Classifier	Precision	Recall	F-measure
Naive Bayes (NB)	63.7%	61.8%	**62.2%**
Support Vector Machine (SVM)	59.7%	60.4%	**59.5%**
Random Forest	58.9%	55.2%	50.9%
Decision Tree	48.1%	49.5%	48.2%
KNN, K=1	53.8%	48.9%	41.9%
KNN, K=2	50.6%	44.5%	31.4%

Table 3: Initial results of classifying 2 COIN classes

Classifier	Precision	Recall	F-measure
Naive Bayes (NB)	76.6%	76.5%	**76.0%**
Support Vector Machine (SVM)	71.9%	72.0%	**72.0%**

Limitations: *Accuracy* – Training the classifier using unbalanced amount of data for each COIN class is a challenging task that limits the accuracy results. However, if we have a larger corpus and consequently more training data, then the obtained accuracy could be increased. *Robustness* – Reading API documentations written by non-native English speakers or developers who lack technical-writing skills is a challenging task for humans. Similarly, the correctness of our model results decreases with informal phrases, ambiguous sentences, and implicit or hidden assumptions.

4. CONCLUDING REMARKS

In this paper, we presented our work on extracting conceptual interoperability constraints from NL text via ML. Previous researches on API documentation proposed identifying parameters' dependency constraints [9], methods' pre/post conditions [5], and resource specifications [10]. While some of these works used NLP with rule-based identification (e.g., [5, 9]), Zhong et al. [10] used ML to identify the name of restricted entities but not the restrictions themselves.

In the future, we plan to increase our results significance by enriching the corpus with further classified data from other cases of API documentations, and to refine the classification algorithms by including lexical references (e.g., WorldNet) and some handling techniques for data imbalance. Also, we intend to examine other classification algorithms (e.g., multinomial NB) to achieve better accuracy.

5. ACKNOWLEDGMENTS

This work is supervised by Prof. Dieter Rombach and funded by the PhD Program of the Computer Science Department of University of Kaiserslautern.

[3]Weka: *http://www.cs.waikato.ac.nz/ml/weka*

6. REFERENCES

[1] H. Abukwaik, M. Naab, and D. Rombach. A proactive support for conceptual interoperability analysis in software systems. In *Working Conference on Software Architecture. 2015. WICSA'15*, 2015.

[2] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/˜cjlin/libsvm.

[3] S. Dumais, J. Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management*, pages 148–155. ACM, 1998.

[4] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

[5] 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*, pages 815–825. IEEE Press, 2012.

[6] S. Robertson. Understanding inverse document frequency: on theoretical arguments for idf. *Journal of documentation*, 60(5):503–520, 2004.

[7] A. Rojo. Quantitative methods in corpus-based translation studies: A practical guide to descriptive translation research michael p. oakes and meng ji (eds)(2013). *Journal of Research Design and Statistics in Linguistics and Communication Science*, 1(1):113–118, 2013.

[8] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 111–147, 1974.

[9] Q. Wu, L. Wu, G. Liang, Q. Wang, T. Xie, and H. Mei. Inferring dependency constraints on parameters for web services. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1421–1432. International World Wide Web Conferences Steering Committee, 2013.

[10] 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*, pages 307–318. IEEE Computer Society, 2009.