Towards Seamless Analysis of Software Interoperability: Automatic Identification of Conceptual Constraints in API Documentation

Hadil Abukwaik[1], Mohammed Abujayyab[2], and Dieter Rombach[1]

University of Kaiserslautern,
Gottlieb-Daimler-Straße 47,
67663 Kaiserslautern, Germany
[1]{abukwaik,rombach}@cs.uni-kl.de, [2]mohabujayya@gmail.com

Abstract. Building successful and meaningful interoperation with external software APIs requires satisfying their conceptual interoperability constraints. These constraints, which we call the COINs, include structure, dynamic, and quality specifications that if missed they may lead to unexpected mismatches with additional cost and running-late projects. However, for software architects and analysts, manual analysis and identification of conceptual interoperability constraints from unstructured text in API documents is tedious and time consuming task. In this paper, we present our novel ideas for addressing the aforementioned issues by utilizing machine learning techniques. In our research, we followed an empirically-based methodology. We started with a multiple-case study that resulted in our first contribution, which is a ground truth dataset. Then, we built a model for this dataset that we tested its robustness through experiments using different machine learning text-classification algorithms. The results show that our model helped the classification algorithms in achieving 70.4% precision and 70.2% recall for identifying seven classes of constraints (i.e., Syntax, Semantic, Structure, Dynamic, Context, Quality, and Not-COIN). This achievement increased to 81.9% precision and 82.0% recall when identifying two classes (i.e., COIN, Not-COIN). Finally, we implemented a tool prototype that demonstrates the value of our findings for architects in practical context.

Keywords: Interoperability analysis, conceptual constraints, black-box interoperation, API documentation, empirical study, machine learning

1 Introduction

Interoperating with externally developed black-box web service or platform APIs is restricted with their **Co**nceptual **i**nteroperability co**n**straints (COINs), which are defined as the characteristics controlling the exchange of data or functionalities at the following conceptual classes: syntax, semantics, structure, dynamics, context, and quality) [2]. Hence, to build a successful interoperation, software architects and analysts need to identify and fulfil these conceptual constraints of

the external APIs. Otherwise, unexpected conceptual mismatches will prevent the whole interoperation or make its results meaningless. This consequently cause resolution expenses at later stages of the projects [8]. Therefore, it is necessary to effectively analyze shared documents about a software API of interest.

Current analysis approaches relies on manual investigation of shared API documents [9]. However, such manual reading and inspection of natural language text in these documents is an exhausting, time-consuming, and error-prone task [19]. Add to this, it requires knowledge about the different conceptual constraints and linguistic analysis experience.

In this paper, we elaborate on and extend our proposed conceptual interoperability analysis framework [2]. In particular, we automate the identification of COINs in API documentations' text by employing machine learning (ML) techniques. Our goal is to assist software architects and analysts in performing effective and efficient conceptual interoperability analysis. We followed a systematic empirically-based research methodology, which has two main parts. In the first part, we conducted a multiple-case study that yielded our first contribution, which is the ground truth dataset. This dataset is a reusable asset that represents a repository of textual sentences that we collected from multiple API documents and manually labeled them with a specific COIN class. In the second part, we contributed a classification model for the COINs in the ground truth dataset, and we evaluated it through experiments using different ML text-classification algorithms. Our experiments revealed promising results towards automating the identification of COINs in text of API documents. We achieved up to 70.4% precision and 70.2% recall for identifying seven classes of constraints (i.e., syntax, semantics, structure, dynamics, context, quality, and Not-COIN). Moreover, the accuracy increased to reach 81.9% precision and 82.0% recall for identifying two classes (i.e., COIN, Not-COIN). Finally, we developed a tool prototype that demonstrates the value of our ideas in serving software architects during their interoperability analysis task. In specific, the tool allows the architect to select a textual sentences in an API document webpage and it will detect and report the existence of COIN with its type. Such a service can definitely enhance the interoperability analysis results especially for inexperienced architects.

The rest of this paper is organized as follows. Section 2 introduces a background, Section 3 overviews the related works, and Section 4 outlines our research methodology. Section 5 and Section 6 detail our first and second research parts. Section 7 presents our tool support and Section 8 offers the conclusion.

2 Background

In this chapter we present a brief introduction for conceptual interoperability constraints and the used machine learning techniques in our research.

2.1 Conceptual Interoperability Constraints

The presented work in this paper is based on the Conceptual Interoperability Constraints (COIN) model [2], which focuses on the non-technical constraints

of interoperable software systems and can be applied to different types of software systems (e.g., information systems, embedded systems, mobile systems, etc.). That is, COINs are the conceptual characteristics that govern the software systems interoperability with other systems. That is, wrong understanding or missing of COINs might defect the desired interoperability by conceptual inconsistencies or meaningless results. There are six classes of COINs that we summarize as the following: (1) *Syntax COINs* that state the constraints packaging (e.g., used terminology or modeling language). (2) *Semantic COINs* that express meaning-related constraints (e.g., goals of methods). (3) *Structure COINs* that depict the systems elements, their relations, and arrangements affecting the interoperation results (e.g., data distribution). (4) *Dynamic COINs* that restrict the behavior of interoperating elements (e.g., synchronization feature). (5) *Context COINs* that pertain to external settings of the interoperation (e.g., user and usage properties). (6) *Quality COINs* that capture quality characteristics related to exchanged data and services (e.g., interoperation response time).

2.2 Machine Learning for Text Classification

In order to enable the automatic detection of COINs in text, we employed ML text classification algorithms (e.g., NaveBayes [10] and Support Vector Machine [18]). The accuracy results of such algorithms depend on the quality and the size of the dataset [4], which consists of manually labeled sentences with one of the predefined classification classes. Text classification process consists of:
- *Building the classification model*, in which all features of the sentences in the dataset are identified and modeled mathematically. In our research, we used popular techniques for building our model: (1) Bag of Words (BOWs) [6] that considers each word in a sentence as a feature, and accordingly a document is represented as a matrix of weighted values; (2) N-Grams [16] that considers each N adjacent words in a sentence as a feature, where ($N > 0$).
- *Evaluating the classification model*, in which the manually labeled dataset is divided into a training and testing sets. The training set is used for training the ML classification algorithm about the features captured in the model, while the testing set is for evaluating the classification accuracy. For our research we used *k-fold Cross-validation* [11], in which our ground truth dataset (i.e., COINs Corpus) is divided into k folds. Then, ($k-1$) folds are used for training and one fold is used for testing. Finally, an average of k evaluation rounds is computed.

3 Related Work

A number of previous works proposed automating the identification of specific types of interoperability-related constraints from API documents. Wu et al. [19] targeted parameters dependency constraints, Pandita et al. [13] inferred formal specifications for methods pre- and post- conditions, and Zhong et al. [20] recognized resource specifications. Our work complements these works and elaborates

on Abukwaik et al. [2] idea of extracting a comprehensive set of conceptual interoperability constraints.

On a broader scope, other works proposed retrieving information to assist software architects in different tasks. Anvaari and Zimmermann [3] retrieved architectural knowledge from documents for architectural guidance purposes. Figueiredo et al. [7] and Lopez et al. [12] searched for architectural knowledge in emails, meeting notes, and wikis for proper documentation purposes. Although, these are important achievements, they do not meet our goal of assisting architects in interoperability analysis tasks.

In general, our work and the aforementioned related works intersect in the utilization of natural language processing techniques in retrieving specific kind of information from documents. However, they used rule-based and ontology-based retrieval approaches, while we explored ML classification algorithms that are helpful for information retrieval in natural language text. Add to this, we performed our research systematically and contributed a reusable ground truth dataset to enable research replication and results comparison by other researchers.

4 Research Methodology

In this research, we systematically revealed the potentials of automating the extraction of COINs from API documents using ML techniques. Our research goal formulated in terms of GQM goal template [5] is: *to* support the conceptual interoperability analysis task *for the purpose of* improvement *with respect to* effectiveness and efficiency *from the viewpoint of* software architects and analysts *in the context of* analyzing text in API documentation within software integration projects. We translate this goal into the following research questions:
RQ1: *What are the existing conceptual interoperability constraints, COINs, in the text of API documentation?*
This question explore the current state of COINs in real API documents. It aims at building the ground truth dataset (i.e. COINs Corpus representing a repository of sentences labeled with their COIN class). This forms a main building block towards the envisioned automatic extraction idea.
RQ2: *How effective and efficient would it be to use ML techniques in automating the extraction of COINs from text in API documentations?*
This question explores the actual benefits of utilizing ML in supporting software architects and analysts in analyzing the text. It aims at building a classification model that will be evaluated through well-known ML classification algorithms.

In order to achieve the stated goal and answer the aforementioned questions, we performed our research in two main parts as follows:
Research Part 1 (Multiple-case study). In this part, we systematically explored the state of COINs in six cases of API documentations. The result of this part is a ground truth dataset (i.e., COINs Corpus). We detail the study design and results in Section 5.
Research Part 2 (Experiments). In this part, we started with using the ground truth dataset resulted in the previous part to build the COIN Classifi-

cation Model. Afterwards, we investigated the capability of different ML classification algorithms in identifying the COINs in text using our model. We detail the process and results of this part of our research in Section 6.

Our systematic research provided us with the traceability between the different activities and their results. Moreover, it enables future researchers to independently replicate our work and compare the results.

5 Multiple-Case Study: Building the Ground Truth Dataset for COINs

In this section, we describe our multiple-case study design, execution, and results.

5.1 Study Design

Study goal. We aim at answering the first research question RQ1 that we stated in Section 4. In order to do so, we needed to examine real-world API documentations to discover the state of conceptual interoperability constraints in them.
Research method. We decided to perform a multiple-case study with literal replication of cases from different domains. Such a method aids in collecting significant evidences and drawing generalizable results.
Case selection. For systematic selection of cases of API documentations, we considered the following selection criteria:
SC1: Mashup Score. This is a published statistical value[1] that represents the popularity of a web service API in terms of its integration frequency by developers into new bigger APIs.
SC2: API Type. This can be either Web Service API or Platform API.
SC3: API Domain. We also consider the application domains for the selected API documents (e.g., social blogging, audio, software development, etc.).
Analysis unit. Our case study has a holistic design, which means that we have a single unit of analysis. This unit is "the sentences in API documents that include COIN instances". To document and maintain the analyzed sentences, we designed a data extraction sheet that we implemented as an MS Excel sheet. This sheet consists of demographic fields (i.e., API name, date of retrieval, mashup score, API type, API domain, and no. of sentences) and analysis fields (i.e., case id, sentence id, sentence textual value, and the COIN class).
Study protocol. Our multiple-case study protocol includes three main activities that are adapted from the processes proposed by Runeson [17]. The study activities are case selection, case execution, and cross-case analysis that we summarize in Fig. 1 below and describe in details within the next subsection.

5.2 Study Execution and Results

Based on our predefined case selection criteria, in August 2015 we chose six API documentations. Four API documents from the web services type (i.e.,

[1] Programmable web: http://www.programmableweb.com/apis/directory

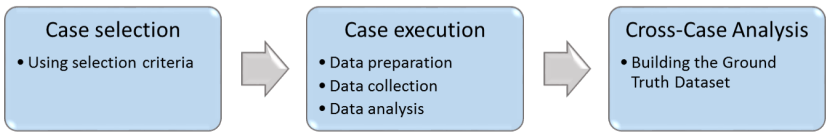

Fig. 1. Multiple-Case Study Process

SoundCloud, GoogleMaps, Skype, and Instagram) and two from the platform type (i.e., AppleWatch and Eclipse-Plugin Developer Guide). These cases cover different application domains (i.e., social micro-blogging, geographical location, telecommunication, social audio, and software development environment). With regards to the mashup criteria, our four cases of web service APIs are chosen to cover a wide range of scores starting from 30 for Skype and ending with 2582 for GoogleMaps. After selecting our cases, we executed each case as the following:
Data Preparation. We started this step with fetching the API documentation for the selected case from its online website. Then, we read the documents and determined the webpages that had textual content offering conceptual software description and constraints (e.g., the Overview, Introduction, Developer Guide, API Reference, Summary, etc.). Subsequently, we started processing the text in chosen webpages by performing the following:
- *Automatic Filtering*. We implemented a simple PHP code using Simple HTML DOM Parser [2] library to filter out the text noise (i.e., headers, images, tags, symbols, html code, and JavaScript code). Thus, we passed the URL link of the chosen webpage (input) to our implemented code. Then, we get back as a .txt file containing the textual content of the webpage (output).
- *Manual Filtering*. The automatic filtering fells short in excluding specific types of noise (e.g., text and code mixture, references like "see also", "for more information", "related topics", copyrights, etc.). These sentences could mislead the machine learning in our later research steps, so we removed them manually.
Data Collection. In this step, we cut the content of the text file resulted from previous step into single sentences within our designed data extraction sheet (.xsl file) that we described in Subsection 5.1. We completed all the fields of the data sheet for each sentence except for the "COIN class" filed that we did within the next step. *Note that*, we maintained a data storage, in which we stored the original HTML webpages of the selected API documentations, their text file, and their excel sheet. This to enable later replication of our work by other researchers as documentations change continuously.
Data analysis. We manually analyzed each of the collected sentences in the extraction sheet and carefully assigned it a COIN class. This classification was based on an interpretation criteria, which is the COIN Model with its six classes (i.e., Syntax, Semantic, Structure, Dynamic, Context, and Quality). A seventh class was created for sentences that have no COIN instance (i.e., Not-COIN

[2] Simple HTML DOM: http://simplehtmldom.sourceforge.net/

class). For example, a sentence like "A user is encapsulated by a read-only Person object." was classified as a "Structure COIN". While, a sentence like "You can also use our Sharing Kits for Windows, OS X, Android or iOS applications", was classified as a "Not-COIN" as it did not express a conceptual constraint, but rather a technical information.

The result of this step was a very critical point towards our envisioned automatic COIN extraction idea. Hence, the data analysis was performed by two researchers, who classified all sentences for each case separately. Then, in multiple discussion sessions, the two researchers compared their classification decisions and resolved conflicts based on consensus.

Obviously, the case execution process consumes time and mental effort, especially in the data analysis step. Table 1 summarizes the distribution of our collected 2283 sentences among the cases along with the effort (in terms of hours) that we spent in executing them. Noticeably, SoundCloud and Instagram have small documents, and consequently they have the smallest share of sentences included in our study (i.e., 9.5% and 11%). Meanwhile, Eclipse documentation is the largest and consequently has the highest share of sentences (i.e., 28.5%).

Table 1. Case-share of sentences and execution effort

API Document	Total number of sentences	Total execution efforts (Hours)
Sound Cloud	219	7.7
GoogleMaps	473	6.5
AppleWatch	360	8.0
Eclipse Plugin	651	12.0
Skype	325	4.5
Instagram	255	4.8
Total	**2283**	**43.5**

Cross-Case Analysis (Answering RQ1: What are the types of existing conceptual interoperability constraints, COINs, in the text of current API documentations?). After executing all cases, we arranged the incrementally classified sets of sentences from all cases (i.e., 2283 sentences) into one repository that we call the *ground truth dataset* or *the COINs Corpus* as called in ML. We have developed two versions of this dataset as the following:
Seven-COIN Corpus, in which, each sentence belongs to one of the seven classes (i.e., Not-COIN, Dynamic, Semantic, Syntax, Structure, Context, or Quality).
Two-COIN Corpus, in which, each sentence belongs to one of two classes rather than seven (i.e., COIN or Not-COIN). In fact, the Two-COIN Corpus is derived from the Seven-COIN Corpus by abstracting the six COIN classes into one class. In Table 3, we present the difference between the two Corpora with example sentences.

Table 2. Example of content in the Seven-COIN and Two-COIN Corpus

Sentence ID	Sentence	Seven-COIN Class	Two-COIN Class
s1	You can also use our Sharing Kits for Windows, OS X, Android or iOS applications.	Not-COIN	Not-COIN
s2	When it is finished manipulating the object, it releases the lock.	Dynamic	COIN
s3	A user is encapsulated by a read-only Person object.	Structure	COIN
s4	A user's presence is a collection of information about the users' availability, their current activity, and their personal note.	Syntax	COIN
s5	A dynamic notification interface lets you provide a more enriched notification experience for the user	Semantic	COIN
s6	This service is not designed to respond in real time to user input	Context	COIN
s7	Your interfaces need to display information quickly and facilitate fast navigation and interactions.	Quality	COIN

The aim behind building these two versions of the corpus is to better investigate the ML performance results, when using the different versions of the ground truth dataset later in our research. We explain this in more details in Section 6.

COIN-share in the contributed ground truth dataset. In Fig. 2, we illustrate the distribution of sentences among the COIN classes within the Seven-COIN Corpus (on the left). It is noticed that the Not-COIN class that expresses technical constraints rather than conceptual ones is the dominant among the other six COIN classes (i.e., 42%). Then, Dynamic and Semantic classes have the second and third biggest shares. Remarkably, the Structure, Syntax, Quality, and Context are very few with convergent shares ranging between 1% and 5%. An aggregated share for all COIN classes is shown in the Two-COIN Corpus (on the right of Fig. 2).

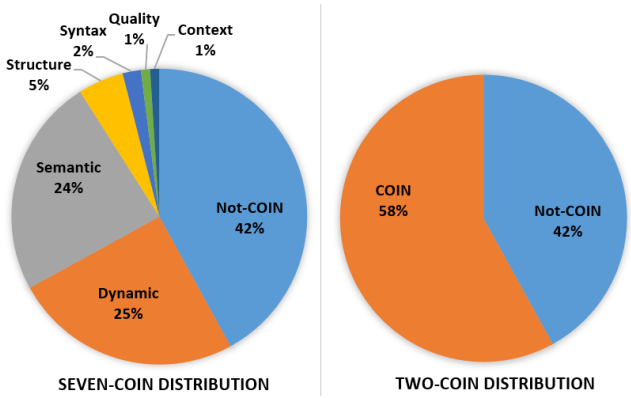

Fig. 2. COIN-share in the Ground Truth Dataset

COIN-share in the cases. On a finer level, we have investigated the state of COINs in each case rather than in the whole ground truth dataset. We found that the content of each API document was focused on the Not-COIN, Dynamic and Semantic classes similarly as in the aggregated findings on the complete dataset seen in Fig. 2. For example, in the case of AppleWatch documentation, 40.8% of the content is for Not-COIN, 26.1% for Dynamic, and 25% for Semantic. Add to this, all cases had less than 10% of its content to the Structure, Syntax, Quality, and Context classes (e.g., Eclipse-Plugin gave them 8.5%).

5.3 Discussion

Technical-oriented API documentations. The Not-COIN class reserves 42% of the total sentences in the investigated parts of the API documents that were supposed to be conceptual (i.e., overview and introduction sections). A noteworthy example is the GoogleMaps case, which took it to an extreme level of focus on the technical information (i.e., 63% of its content was under the Not-COIN class, 11.2% for Dynamic class, and 13.1% for Semantic class, and the rest shared by the other classes). Accordingly, it is important to raise a flag about the lack of sufficient information about the conceptual aspects of interoperable software units or APIs (e.g., usage context, terminology definitions, quality attributes, etc.). This concern needs to be brought to the notice of researchers and practitioners who care about the usefulness and adequacy of content in API documentations. This also obviously has a direct influence on the effectiveness of architects and analysts in the conceptual interoperability analysis activities.
Considerable presence of Dynamic and Semantic constraints. Our study findings reveal that the Dynamic and Semantic classes have apparently big shares in current API documents (i.e., 25% and 24% of the dataset). This reflects the favorable awareness about the importance of proper and explicit documenting of the API semantics (e.g., data meaning, service goal, conceptual input and output, etc.) and dynamics (e.g., interaction protocol, flow of data, pre- and post-conditions, etc.). Nevertheless, based on the tedious work we went through our manual analysis for the six cases, we believe that it would be of great help for architects and analysts to have clear boarders between these two classes of constraints within the verbose of text. For example, it would be easier to skim the text if API goal get separated from its interaction protocol, rather than blending them into long paragraphs. This would offer architects and analysts with a better experience and consequently enhance their analysis results.
COIN-deficiency in Platform and Web Service API documents. From our investigated cases, we perceived a convention on assigning insignificant shares for the Structure, Syntax, Quality, and Context classes. Interestingly, the cases varied with regards to what they chose to slightly cover out of these four classes. On one hand, the cases of Web Service APIs were the main contributors to the Context, Quality, and Syntax classes in the ground truth dataset. That is, the documents of GoogleMaps, SoundCloud, Skype, and Instagram provided 82.5% of the Syntax COINs, 70.4% of the Quality COINs, and 92% of the Context COINs. Such a contribution cannot be related to the nature of the APIs as

all functionalities need explicit information about these COINs. For example, it would be critical for a FarmerWatch application to know what the approximate response time it is for the Notification service offered by AppleWatch APIs.
On the other hand, the Platform API documents participated with a 56.1% of the Structure COINs in the ground truth dataset, while the Web Service API documents participated with %43.9. Note that, this is not related to the larger amount of sentences that these two documents contribute to the dataset, but rather due to the internal case share of Structure COINs. On average, the Platform API documents allocates about 6% of their content to structural constraints, while Web Service API documents allocate about 3.6% for these constraints.
Observed patterns for the dominant classes in the ground truth dataset. From the considerable amount of sentences for the Not-COIN, Semantic, and Dynamic classes, we observed a number of patterns in terms of frequently occurring terms and sentences. We envision that using the patterns in combination with the BOW in future experiments would enhance the results of the automatic COIN identification. Below we describe some of these patterns.
Patterns of the Not-COIN class. We observed the presence of "Technical Keywords", which are mainly abbreviations of software technologies (e.g., XML, iOS, XPath, JavaScript, ASCII, etc.). With further analysis, we found that 30.7% of the Not-COIN instances in the ground truth dataset have technical keywords. Another pattern for this class is variables words with special format (e.g., "XML responses consist of zero or more ¡route¿ elements."). Also, sentences beginning with specific terms (e.g., "for example", "for more information", "see", etc.) have been recurring in 12.8% of the Not-COIN instances.
Patterns of the Dynamic class. We found a number of recurrent terms related to actions and data/process flow. Hence, we gathered them into a list that we call the "Action Verbs", which includes: create, use, request, access, plug, lock, include, setup, run, start, call, redirect, and more. In fact, 35.8% of the total number of sentences with dynamic COINs have one or more of these terms. Furthermore, 24% of the Dynamic COINs were recognized as sentences containing a conditional statement that expresses a pre- or post- condition. For example, the sentence "If a command name is specified, the help message for this command is displayed" has a Dynamic COIN expressing a pre-condition.
Patterns of the Semantic class. We noticed repeated terms that we organized into: "Input/Output Terms" that showed up in 18.8% of the Semantic COINs (e.g., return, receive, display, response, send, result, etc.) and "Goal Terms" that were seen in 16.4% of the Semantic COINs (e.g., allow, enable, let, grant, permit, facilitate, etc.). For example, the sentence "A dynamic notification interface lets you provide a more enriched notification experience for the user" has a Semantic COIN expressing a goal.

5.4 Threats to Validity

Generalizability. To obtain significant findings and draw generalizable conclusions, we included multiple cases for building the ground truth that plays prominent role in our research. We literally replicated six API documents (i.e.,

SoundCloud, GoogleMaps, Skype, Instagram, AppleWatch and Eclipse-Plugin Developer Guide) from two different types (Web Service and Platform APIs).
Completeness. Due to resource limitations (i.e., time and manpower), we were unable to analyze the large API documents completely. However, we were careful with respect to selecting inclusive parts of such large documents. For example, out of the huge document of Eclipse APIs, we covered the Plugin part.
Researcher bias. To build our ground truth dataset in a way that guarantees results accuracy and impartiality, we replicated the manual classification of the cases sentences by two researchers separately based on the COINs Model as an interpretation criteria. In multiple discussion sessions, the researchers compared their classification decisions and resolved conflicts based on consensus.

6 Experiments: Automatic Identification of COINs Using ML

In this section, we detail the experiments design, execution, and results.

6.1 Experiments Design

Experiments goal. This part of our research aims at answering the second research question RQ2 that we stated in Section 4. In order to do so, we needed to examine ML techniques to discover their potentials in supporting architects and analysts in automatically identifying the COINs in text of API documents.
Research method. We built a classification model and ran multiple experiments employing different ML text classification algorithms. Such a method enables us to compare between the algorithms results and to draw solid conclusions about the advantages that ML can bring in addressing the challenges of manual interoperability analysis.
Evaluation method and metrics. We used k-fold Cross-validation, which we explained in the background section, with $k = 10$. With regards to the evaluation metrics for the accuracy of the classification algorithms, we used the following commonly used measures [14]:
Precision: the ratio of correctly classified sentences by the machine to the total number of sentences classified by the machine either correctly or incorrectly.
Recall: the ratio of correctly classified sentences by the machine to the total number of sentences in the corpus.
F-measure: the harmonic mean of precision and recall that is calculated as: $(2 * Precision * Recall)/(Precision + Recall)$.
Experiments protocol. Our experiments protocol includes three main activities that are: feature selection, feature modeling, and ML algorithms evaluation. We illustrate this protocol in Fig. 3, and we describe it in details within the next subsection. We ran this protocol twice, once for the Seven-COIN Corpus and another for the Two-COIN Corpus.

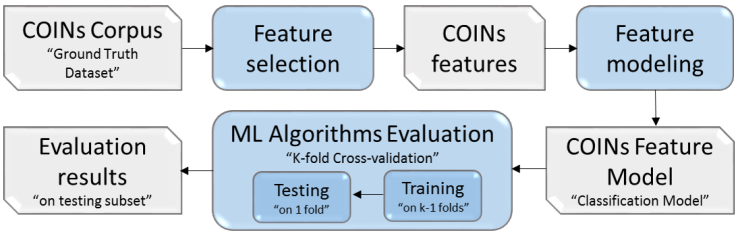

Fig. 3. Experiments Process

6.2 Experiments Execution and Results

We performed all our execution on Weka v 3.7.11 [3], which is a suite of ML algorithms written in Java with result visualization capabilities. This started with processing the textual sentences in our contributed dataset (i.e., COINs Corpus) using natural language processing (NLP) techniques. The processing included tokenizing sentences into words, lowering cases, eliminating noise words (e.g., is, are, in, of, this, etc.), and stemming that put words into their root format (e.g., encapsulating and encapsulated are returned as encapsulate).

Feature selection. After processing the text, we started the, which is the identification for the most representative features or keywords for the COIN classes within the COINs Corpus. We performed this automatically using the Bag-of-Words (BOWs) and N-Gram approaches, which we explained in the background section. That is, each sentence was represented as a collection of words. Then, each single word and each n-combination of words in the sentence was considered as a feature, where N was between 1 and 3. For example, in a sentence like "A user is encapsulated by a read-only Person object", the word "encapsulate" and the combination "read-only" were considered as two of its features. The output of this step is the set of features for the COINs Corpus.

Feature modeling. In this stage, the whole COINs Corpus is transformed into a mathematical model. That is, it is represented as a matrix, in which headers contain all extracted features from the previous phase, while each row represents a sentence of the corpus. Then we weighted the matrix, where each cell [row, column] holds the weight of a feature in a specific sentence. For weighting, we used the Term Frequency-Inverse Document Frequency (TF-IDF) [15], which is often used for text retrieval. The result of this was the COINs Feature Model (or the classification model), which is considered as a reusable asset reserving knowledge about conceptual interoperability constraints in API documents.

ML Algorithms Evaluation. We selected a number of well-known ML text classification algorithms (e.g., NaiveBayes versions, Support Vector Machine, Random Forest Tree, K-Nearest Neighbor KNN, and more). Then, we ran these algorithms on the classification model resulted from the modeling activity.

Evaluation results (Answering RQ2: How effective and efficient would it be to use ML techniques in automating the extraction of COINs

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

from text in API documentations?).

Effectiveness of identifying the COINs using ML Algorithms. We report the

Table 3. COINs identification results using different ML algorithms

ML Algorithm	*Seven-COIN Corpus*			*Two-COIN Corpus*		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure
ComplementNaïveBayes	70.4%	70.2%	70.0%	81.9%	82.0%	81.9%
NaïveBayesMutinomialupdatable	66.0%	65.1%	65.4%	81.9%	82.0%	81.8%
Support Vector Machine	59.3%	60.0%	59.0%	75.7%	75.7%	75.7%
Random Forest Tree	60.4%	56.3%	52.3%	73.7%	73.9%	73.7%
Simple Logistic	52.5%	54.4%	52.4%	68.2%	68.4%	67.2%
KNN K=1	54.8%	45.5%	40.8%	64.2%	52.3%	47.8%
KNN K=2	49.8%	36.1%	30.1%	64.4%	48.7%	40.6%

effectiveness results in terms of accuracy metrics in two cases:
- *Seven-COIN Corpus Case.* The evaluation results showed that the best accuracy in automatically identifying seven different classes of interoperability constraints in text was achieved by the ComplementNaveBayes algorithm (see Table 3). It achieved 70.4% precision, 70.2% recall, and 70% F-measure. In the second place comes NaveBayesMutinomialupdatable algorithm with about 5% less accuracy than the first algorithm. The rest of the results show accuracy F-measure between 62.8% and 59.0%. The worst results were from the KNN algorithms.
- *Two-COIN Corpus Case.* By applying the same algorithms on the Two-COIN Corpus, we obtained better results. In particular, the accuracy increased with almost 11% compared to the results in the Seven-COIN case with the ComplementNaiveBayes algorithm. That is, the precision increased to 81.9%, recall to 82.0%, and F-measure to 81.9%. Similar to the previous case, NaveBayesMutinomialupdatable comes in the second rank and the 2-Nearest Neighbor algorithm has the worst results as seen in Table 3. Note, we have achieved results show improvement in accuracy compared to our preliminary investigation [1], in which we achieved F-measure of 62.2% using the NaveBayes algorithm.

Efficiency of identifying the COINs using ML Algorithms. Obviously, the machine beats the human performance in terms of the spent time in analyzing the text. As we mentioned earlier, analyzing the documents costed us about 44 working hours, while, it took the machine way less time. For example, training and testing the NaveBayesMultinominalupdate took about 5 seconds on our complete corpus with 2283 sentences). This efficiency would enhance when using machines with faster and more powerful CPU (we ran the experiments on a machine with Intel core i5 460 M CPU with 2.5 GHZ speed).

6.3 Discussion and Limitations

Towards automatic conceptual interoperability analysis. The achieved effectiveness of the automatic identification of constraints (e.g., 81.9% F-measure) are promising and shows the capabilities of ML in serving software architects through their interoperability analysis tasks. It is worth mentioning that although results did not reach a 100% accuracy, it is still high. This is due to the fact that the algorithms results are being compare to our extensive sentence-by-sentence analysis for the API documents, which we did for the sake of building a robust corpus. However, in practice the sentences are not examined in such a heavy way especially when projects are limited in time and manpower.
Larger corpus, better accuracy results. It is known in ML that the more classification classes you want to train the machine on identifying, the more training data it requires to be fed with. This explains the higher accuracy we achieved using the Two-COIN Corpus compared to the Seven-COIN Corpus even with the same amount of sentences in both. Therefore, we conclude that the larger the corpus, the better accuracy we can achieve using ML algorithms.
Unbalanced amount of instances for each class in the corpus. As noticed, the number of instances for the COIN classes is not balanced in the COINs Corpus. That is, dominant classes (i.e., Not-COIN, Dynamic, and Semantic) contribute with the majority of sentences in the data set (i.e., 91%). While, the other classes (i.e., Structure, Syntax, Quality and Context) are smaller and share the left 9% of the corpus. This affects the classification accuracy of the classes with fewer instances. Therefore, we plan for the future work to increase the number of instances in the training data to achieve better results.

7 Tool Support (A Prototype)

To bring our ideas to practical life, we designed a tool as a web browser plugin that aims at assisting software architects and analysts in their conceptual interoperability analysis. The tool takes a sentence from an API document and automatically recognizes if it has a conceptual interoperability constraint and reports its COIN class (see Fig. 4). Such a functionality saves time and effort for performing the interoperability analysis and it has potentials in improving the results effectiveness especially for inexperienced architects and analysts. We built an easy-to-use prototype for the tool, which encapsulates our contributed classification model and mirrors its accuracy.

We implemented our tool prototype as a plugin for the Chrome web browser using Java and JavaScript languages. The functionality was offered as a web service and all communication is over the Simple Object Access Protocol (SOAP). The tool design includes: (1) *Front-End component* that we developed using JavaScript to provide the graphical user interface. (2) *Back-End component* that we developed using Java and Weka APIs. It is responsible for locating our service on the server, passing it the input sentence, and carrying back the response.

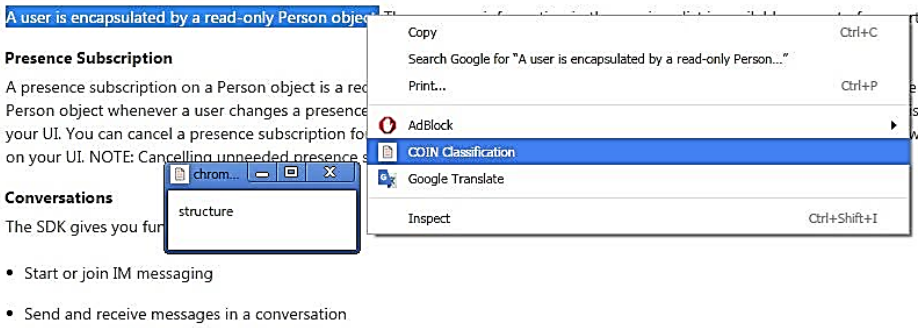

Fig. 4. A tool prototype encapsulating the ML supported identification of COINs

8 Conclusion and Future Work

In this paper we have presented our ideas about supporting software architects in performing seamless conceptual interoperability analysis. The contribution pursued by this work was to utilize ML algorithms for effective and efficient identification of conceptual interoperability constraints in text of API documents. Our systematic empirically-based research included a multiple case study that resulted in the ground truth dataset. Then, we built a ML classification model that we evaluated in experiments using different ML classification algorithms. The results showed that we can achieve up to 70.0% accuracy (in terms of F-measure) for identifying seven classes of interoperability constraints, and it increases to 81.9% for identifying two classes.

In the future, we plan to analyze further API documents to advance the generalizability of our results. Besides, this would enrich the ground truth dataset, allowing better training for the ML algorithms and accordingly better accuracy in identifying the conceptual interoperability constraints. With regards to the tool, we will extend it to generate full reports about all interoperability constraints in a webpage and to collect instant feedback from users about automation results. We also plan to empirically evaluate our ideas in industrial case studies.

9 Acknowledgment

This work is supervised by Prof. Dieter Rombach and funded by the PhD Program of the CS Department of Kaiserslautern University. We thank Mohammed Abufouda for his valuable feedback and comments.

References

1. Abukwaik, H., Abujayyab, M., Humayoun, S.R., Rombach, D.: Extracting conceptual interoperability constraints from api documentation using machine learning. In: ICSE'16 (2016)
2. Abukwaik, H., Naab, M., Rombach, D.: A proactive support for conceptual interoperability analysis in software systems. In: WICSA'15 (2015)
3. Anvaari, M., Zimmermann, O.: Semi-automated design guidance enhancer (sadge): A framework for architectural guidance development. In: Software Architecture (2014)
4. Banko, M., Brill, E.: Scaling to very very large corpora for natural language disambiguation. In: Proceedings of 39th Annual Meeting of the Association for Computational Linguistics (2001)
5. Caldiera, V., Rombach, H.D.: The goal question metric approach. Encyclopedia of software engineering 2(1994) (1994)
6. Chu, W., Lin, T.Y.: Foundations and advances in data mining (2005)
7. Figueiredo, A.M., Dos Reis, J.C., Rodrigues, M.A.: Improving access to software architecture knowledge an ontology-based search approach (2012)
8. Garlan, D., Allen, R., Ockerbloom, J.: Architectural mismatch or why it's hard to build systems out of existing parts. In: ICSE'95 (1995)
9. Hallé, S., Bultan, T., Hughes, G., Alkhalaf, M., Villemaire, R.: Runtime verification of web service interface contracts. Computer 43(3) (2010)
10. John, G.H., Langley, P.: Estimating continuous distributions in bayesian classifiers. In: Conference on Uncertainty in artificial intelligence (1995)
11. Kohavi, R., et al.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Ijcai. vol. 14 (1995)
12. López, C., Codocedo, V., Astudillo, H., Cysneiros, L.M.: Bridging the gap between software architecture rationale formalisms and actual architecture documents: An ontology-driven approach. Science of Computer Programming 77 (2012)
13. Pandita, R., Xiao, X., Zhong, H., Xie, T., Oney, S., Paradkar, A.: Inferring method specifications from natural language api descriptions. In: ICSE'12 (2012)
14. Powers, D.M.: Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation (2011)
15. Robertson, S.: Understanding inverse document frequency: on theoretical arguments for idf. Journal of documentation 60(5) (2004)
16. Rojo, A.: Quantitative methods in corpus-based translation studies: A practical guide to descriptive translation research. Journal of Research Design and Statistics in Linguistics and Communication Science 1 (2013)
17. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical software engineering 14(2) (2009)
18. Tong, S., Koller, D.: Support Vector Machine Active Learning with Applications to Text Classification. J. Mach. Learn. Res. 2 (2002)
19. Wu, Q., Wu, L., Liang, G., Wang, Q., Xie, T., Mei, H.: Inferring dependency constraints on parameters for web services. In: WWW'13 (2013)
20. Zhong, H., Zhang, L., Xie, T., Mei, H.: Inferring resource specifications from natural language api documentation. In: ASE'09 (2009)