2017 IEEE/ACM 39th IEEE International Conference on Software Engineering Companion

Statistical Translation of English Texts to API Code Templates

Anh Tuan Nguyen*, Peter C. Rigby†, Thanh Van Nguyen*, Mark Karanfil†, and Tien N. Nguyen‡
*Iowa State University, Email: {anhnt,thanhng}@iastate.edu
*Concordia University, Email: {peter.rigby,mar_kar}@concordia.ca
‡University of Texas at Dallas, Email: tien.n.nguyen@utdallas.edu

***Abstract*—We develop T2API, a context-sensitive, graph-based statistical translation approach that takes as input an English description of a programming task and synthesizes the corresponding API code template for the task. We train T2API to statistically learn the alignments between English and APIs and determine the relevant API elements. The training is done on StackOverflow, which is a bilingual corpus on which developers discuss programming problems in two types of language: English and programming language. T2API considers both the context of the words in the input query and the context of API elements that often go together in the corpus. The derived API elements with their relevance scores are assembled into an API usage by GRASYN, our novel graph-based API synthesis algorithm that generates a graph representing an API usage from a large code corpus. Importantly, it is capable of generating new API usages from previously seen sub-usages.**

***Keywords*-Text-to-Code Translation; API Usage Synthesis; Statistical Machine Translation**

Developers use the functionality of libraries via Application Programming Interfaces (API) accessing the classes, methods, and fields that make up the APIs. Software libraries can be used in different ways, but not all of them are well documented in the official documentation and programming guides [1]. Researchers have focused on supporting the discovery of knowledge of API usages [2], including suggesting existing code snippets based on English queries [3], [4], [5], [6], [7], or synthesizing common API code usages [8], [9], [10].

To recommend an API usage from a query, earlier works use *information retrieval* (IR) techniques [4], [5], [6], [7] with their goal being centered on *API code search* [2]. Recently, going beyond searching for existing code, researchers have aimed to *generate new API code*, by exploring *statistical approaches* including phrase-based statistical machine translation (SMT) [9], probabilistic CFG [10], AST-based translation [11], and deep neural network [3]. Compared to IR methods, these statistical approaches can synthesize new code, however, they have a key limitation in the use of *a sequence of code tokens or AST structure to represent an API usage*. These sequence/phrase-based and AST-based models *do not capture well the semantic relations (e.g., data or control dependencies)* among API elements in API usages. They might not recognize the same/similar API usages with the same/similar semantic relations among API elements. Thus, they might not learn from an API usage and suggest another usage having the same semantic relations. Importantly, as the popular sequence-based models (*e.g., n*-gram in SWIM [9], RNN in DeepAPI [3]) support API sequences, they do not provide a suitable *abstraction/representation* to *generate complex API code usages* due to the following:

First, *the API elements in a usage might not need to follow a strict order* as enforced in phrase-based models. For example, two files need to be opened for reading and writing. Either file could be opened first. However, these models may not be able to learn from one usage with one order and recommend a usage with the other order. Second, the *related elements of the same API usage might be far apart in a sequence*. The token sequence length might be too short and fail to recognize distant but related APIs in frequent usages. For example, between FileInputStream.open and FileInputStream.close, there might exist several other un-related APIs. As a consequence, they might not generate the related but distant API elements. Third, a phrase-based model *operates on consecutive elements* and *relies on the previous elements to produce the next API element*. Since a phrase often contains many noisy and irrelevant API elements that are part of a specific sequence, an incorrect API element might be generated as part of the API usage. Finally, no *data or control dependencies*, or *control units* are generated.

In this work, we develop T2API, a context-sensitive, **graph-based statistical translation** approach that receives an English query on a programming task and suggests an API usage template. The core of T2API is GRASYN, **a graph synthesis algorithm for API usages** that addresses the challenges we discussed above. First, the set of relevant API elements is derived from the query by our novel **contextual expansion algorithm** that maps the words in the query into those elements. Then, GRASYN ensembles those elements into a usage graph that represents an API usage with data and control dependencies among its elements. It starts from the pivotal API element in a graph and gradually adds other nodes (and the corresponding inducing edges) via a beam search strategy. It aims to cover the derived API elements and to maximize both 1) the *relevance to the words in the query* and 2) **the naturalness** of the newly built **API usage (sub)graph** (measured via GraLan [12] as its occurrence likelihood within a large corpus of API usage graphs extracted from code). With graphs, GRASYN supports *partial orders among API elements* in API usages. Data and control dependencies help connect relevant yet distant API elements in usages, and eliminate irrelevant APIs even when they are close.

To maximize the relevance between the query and the resulting API usage, our contextual expansion algorithm translates

978-1-5386-1589-8/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-C.2017.81

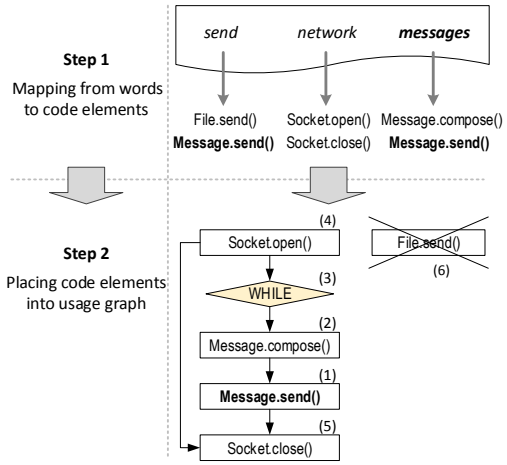

Fig. 1: Example of API Usage Graph Generation in T2API

the given query into a set of API elements with a *prioritized order*. For example, given the input *"send network messages"*, we want to produce the API elements Socket.open, Message.compose, Message.send, and Socket.close. Left-to-right order in text translation is not efficient for the input due to the large numbers of mapped API elements that need to be considered for each word. Moreover, there exist important words which affect the selections of other words during translation. For example, if we map *"send"* first, there would be many choices such as Printer.send and MailServer.send. However, if *"messages"* is selected for mapping first, the choices for *"send"* is smaller and *"send"* is most likely mapped to Message.send since *"messages"* was already translated to Message.compose (Fig. 1). Thus, to determine the next word to map, we consider the *already-translated words* since they help distinguish the context for that word. We call such words *word context*. Similarly, to select an API element among several alternatives, we consider the *already-generated API elements*. We call such API elements *code context*.

Specifically, in training, we used IBM Model [13] to learn the mappings between words and API elements from a corpus. In mapping, the words determined as higher priority (*e.g.,"messages"*) are mapped first via IBM Model. T2API then gradually expands to map the lower-priority words. Such priorities are determined based on both contexts. For example, Socket.open is selected due to its relevancy with the word *"network"* and the already-mapped element Message.send.

Fig. 1 illustrates T2API via an example with the input *"send network messages"*. T2API has two steps:

(1) **Mapping from words to API elements and Contextual Expansion**: First, we use IBM Model [13] to derive the *m*-to-*n* mappings for the pairs of words and API elements with their likelihoods. Next, contextual expansion is proceeded as follows:

- Initially, T2API selects a pair of central word and API element as a starting point for expansion. The central API element needs to be highly relevant to the input. Thus, we choose the API element having highest total mapping score with all the words (measured by the IBM Model). In the example, Message.send is the central API since it has the highest mapping scores to both *"send"* and *"messages"*. Then, the word *"messages"* becomes our central word.
- T2API then starts expanding by considering the words in the input in a prioritized order, *e.g.,"messages", "send", "network"*, with respect to the co-occurrences with the central word. It maps each word to *k* elements. The result is a collection of mapped API elements considering both contexts. In Fig. 1, the collection includes Message.send, Message.compose, Socket.open, Socket.close, and File.send.

(2) **Placing API elements into a usage graph (GRASYN)**: This step corresponds to the addition, removal and re-ordering of terms in the target language in a SMT [14]. Since we use graphs rather than sequences, we perform graph synthesis:

- T2API starts with the central API element Message.send as the initial node of the graph synthesis process. It then gradually adds other nodes (and inducing edges) according to the *naturalness (occurrence likelihood) of the new graph*. It also considers the addition of control units (e.g., for, if) to make the graph complete. During the process, there may be disconnected graphs that could be later joined by adding the nodes. The final graph is connected. In Fig. 1, the numbers show the order that the nodes are added.
- T2API stops after all APIs are covered or the score is lower than a threshold. Nodes considered redundant due to low relevancy with other nodes (*e.g.,*File.send) are removed.

For empirical evaluation, we trained T2API on 236,919 pairs of textual descriptions and the corresponding sets of API elements from StackOverflow. We compared the synthesized usages to the actual code snippets in the SO posts. We found that T2API is able to synthesize API usages with a median top-1 recall and precision of 100% and 67% on API elements in our SO benchmark. Four professional developers and five graduate students judged 77% of our synthesized code to be useful. In comparison with the state-of-the-art approaches, we show that T2API's result is more useful/relevant than that of SWIM [9] and slightly more relevant than that of DeepAPI [3]. T2API generates more complex API code templates (and usage graphs) with control units and data/control dependencies.

In conclusion, we propose a novel context-sensitive, graph-based statistical translation approach that takes a query and synthesizes complex API code templates and graphs with control units and dependencies among API elements. While we use texts from real-world StackOverflow with large numbers of words, the synthesized code captures more complex usages than the state-of-the-art sequence-based approaches. Our generated snippets are made "natural" by our graph synthesis algorithm. We also integrated T2API into Eclipse IDE for API code synthesis from text [15], [16].

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation (NSF) grants CCF-1723215, CCF-1723432, TWC-1723198, CCF-1518897, and CNS-1513263.

REFERENCES

[1] E. Duala-Ekoko and M. P. Robillard, "Asking and answering questions about unfamiliar APIs: An exploratory study," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE'12. IEEE Press, 2012, pp. 266–276.

[2] E. Duala-Ekoko and M. P. Robillard, "Using structure-based recommendations to facilitate discoverability in APIs," in *Proceedings of the 25th European Conference on Object-oriented Programming*, ser. ECOOP'11. Springer-Verlag, 2011, pp. 79–104.

[3] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep API Learning," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. ACM, 2016, pp. 631–642.

[4] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: Finding relevant functions and their usage," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE'11. ACM, 2011, pp. 111–120.

[5] C. McMillan, D. Poshyvanyk, and M. Grechanik, "Recommending Source Code Examples via API Call Usages and Documentation," in *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, ser. RSSE '10. ACM, 2010, pp. 21–25.

[6] W. Zheng, Q. Zhang, and M. Lyu, "Cross-library API recommendation using Web search engines," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. ACM, 2011, pp. 480–483.

[7] W.-K. Chan, H. Cheng, and D. Lo, "Searching Connected API Subgraph via Text Phrases," in *Proceedings of the 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. ACM, 2012, pp. 10:1–10:11.

[8] R. P. L. Buse and W. Weimer, "Synthesizing API Usage Examples," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. IEEE Press, 2012, pp. 782–792.

[9] M. Raghothaman, Y. Wei, and Y. Hamadi, "Swim: Synthesizing what i mean: Code search and idiomatic snippet synthesis," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE'16. ACM, 2016, pp. 357–367.

[10] T. Gvero and V. Kuncak, "Synthesizing Java expressions from free-form queries," in *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, ser. OOPSLA 2015. ACM, 2015, pp. 416–432.

[11] A. Desai, S. Gulwani, V. Hingorani, N. Jain, A. Karkare, M. Marron, S. R, and S. Roy, "Program synthesis using natural language," in *Proceedings of the 38th International Conference on Software Engineering*, ser. ICSE'16. ACM, 2016, pp. 345–356.

[12] A. T. Nguyen and T. N. Nguyen, "Graph-based statistical language model for code," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ser. ICSE '15. IEEE Press, 2015, pp. 858–868.

[13] P. F. Brown, V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer, "The mathematics of statistical machine translation: parameter estimation," *Comput. Linguist.*, vol. 19, no. 2, pp. 263–311, Jun. 1993.

[14] P. Koehn, F. J. Och, and D. Marcu, "Statistical phrase-based translation," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, ser. NAACL '03. Association for Computational Linguistics, 2003, pp. 48–54.

[15] T. Nguyen, P. C. Rigby, A. T. Nguyen, M. Karanfil, and T. N. Nguyen, "T2API: Synthesizing API Code Usage Templates from English Texts with Statistical Translation," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. ACM, 2016, pp. 1013–1017.

[16] "T2APIDemo," https://www.youtube.com/watch?v=Q8Kec6MHGY0&feature=youtu.be.