REST2Bot: Bridging the Gap between Bot Platforms and REST APIs

Mohammad-Ali Yaghoub-Zadeh-Fard, University of New South Wales Sydney, NSW, Australia, m.yaghoubzadehfard@unsw.edu.au (mailto:m.yaghoubzadehfard@unsw.edu.au)
Shayan Zamanirad, University of New South Wales Sydney, NSW, Australia, shayanz@cse.unsw.edu.au (mailto:shayanz@cse.unsw.edu.au)
Boualem Benatallah, University of New South Wales Sydney, NSW, Australia, b.benatallah@cse.unsw.edu.au (mailto:b.benatallah@cse.unsw.edu.au)
Fabio Casati, Servicenow Santa Clara, California, USA, fabio.casati@servicenow.com (mailto:fabio.casati@servicenow.com)

DOI: https://doi.org/10.1145/3366424.3383551 (https://doi.org/10.1145/3366424.3383551)
WWW '20 Companion: Companion Proceedings of the Web Conference 2020 (https://doi.org/10.1145/3366424), **Taipei, Taiwan, April 2020**

With the development of REST (REpresentational State Transfer) APIs, many applications have been designed to harness their potential. As such, bots emerged recently as natural interfaces to facilitate conversations between humans and API-accessible services. Existing bot development platforms (e.g., Dialogflow, Wit.ai) facilitate building bots, but bot developers are still required to provide training data by defining corresponding intents (user's intention such as *booking a hotel*) and entities (e.g., *hotel location*) for each API. Moreover, bot developers are required to build and deploy webhook functions to invoke APIs on intents detection. In this paper, we introduce REST2Bot, a tool that addresses these shortcomings (e.g., translating APIs to Intents, and invoking APIs based on detected Intents) in bot development frameworks to automate several tasks in the life cycle of the bot development process. REST2Bot relies on automated approaches for parsing OpenAPI specifications, generating training data, building bots on desired bot development frameworks, and generating deployable webhook functions to map intents and entities to APIs.

CCS Concepts: • **Information systems** → **RESTful web services**; • **Computing methodologies** → **Artificial intelligence**;

Keywords: REST APIs, Chatbots, Automated Bot Development, Paraphrasing

ACM Reference Format:
Mohammad-Ali Yaghoub-Zadeh-Fard, Shayan Zamanirad, Boualem Benatallah, and Fabio Casati. 2020. REST2Bot: Bridging the Gap between Bot Platforms and REST APIs. In *Companion Proceedings of the Web Conference 2020 (WWW '20 Companion), April 20–24, 2020, Taipei,*

Taiwan. ACM, New York, NY, USA 4 Pages. https://doi.org/10.1145/3366424.3383551 (https://doi.org/10.1145/3366424.3383551)

1 INTRODUCTION

Much of the information we receive about the world is API-regulated. Essentially, APIs are used for connecting devices, managing data, and invoking services [1, 9]. In particular, because of its simplicity, REST is the most dominant approach for designing Web APIs [11]. Meanwhile, thanks to the advances in natural language processing and machine learning, building natural language interfaces to interact with software-enabled services has gained attention by both researchers and organizations (e.g., Apple Siri, Google Virtual Assistant, IBM's Watson, Microsoft's Cortana) [11, 13]. Virtual assistants (also known as bots) serve a wide range of user tasks by mapping user utterances (also called user expressions) into appropriate intents. Examples include reporting weather, booking flights, controlling home devices, and querying databases [13, 14]. Increasingly, organizations have started or plan to use capabilities arising from advances in cognitive computing to increase productivity, automate business processes, and extend the breadth of their business offering.

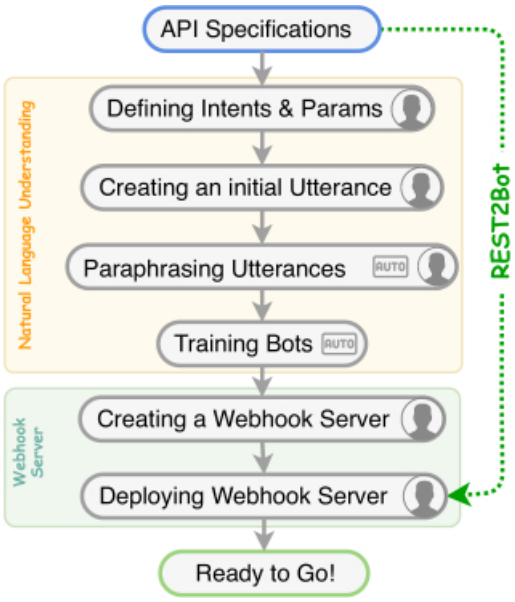


Figure 1: Typical Bot Development Process vs Bot Development Process Using REST2Bot (Green Arrow).

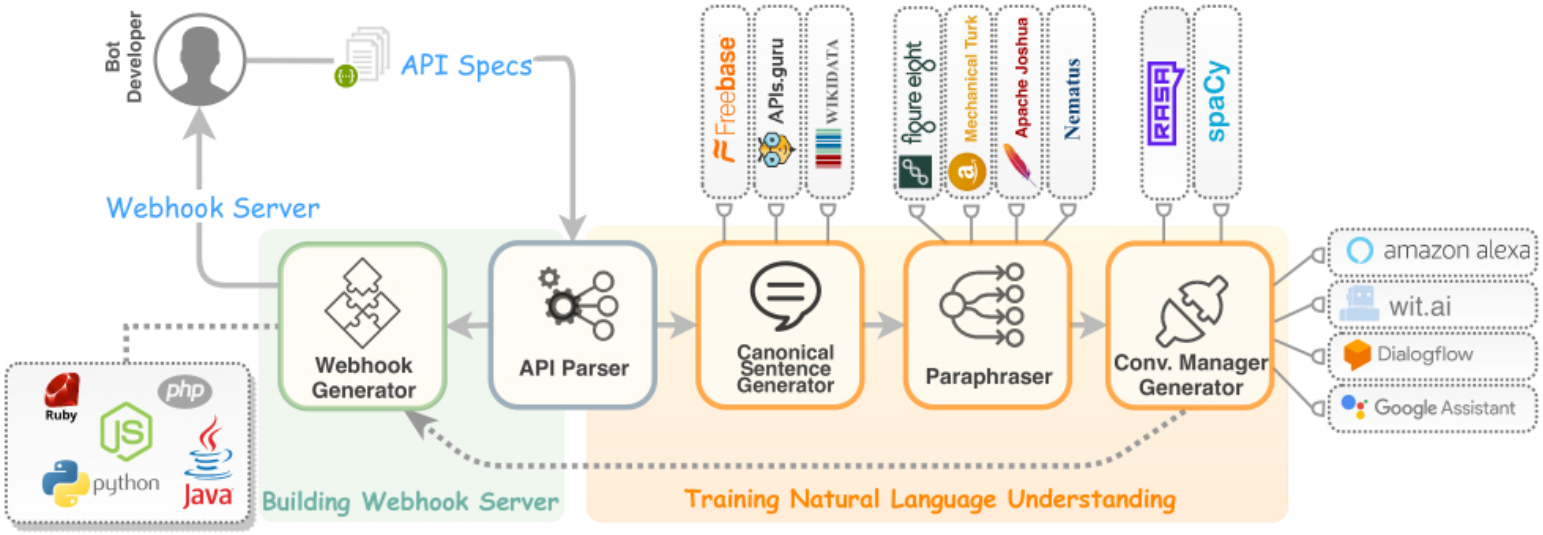


Figure 2: REST2Bot Architecture - Building conversational bots from APIs specifications.

Developing a bot typically implies the ability to invoke APIs corresponding to user utterances (e.g., *"what will the weather be like tomorrow in NYC?"*). This is done in two phases as briefly shown in Figure 1: (i) training a Natural Language Understanding (NLU) model to map user utterances to intents, and (ii) developing Webhook functions to map intents to APIs. Machine-learning based NLU techniques require definition of intents (e.g., *booking hotels*), entity types (e.g., *location*, *date*), and a set of annotated utterances in which entities are labeled with the entity types and intents. To obtain such sets of utterances, the typical approach is to obtain an initial utterance called canonical utterance (e.g., supplied by the bot developer) and then paraphrase the canonical utterance (either manually or automatically) to generate multiple and diverse utterances. Moreover, bot developers (manually) specify mappings between intents and corresponding API calls through the development of Webhooks (i.e, intent-action rules that trigger API calls upon detection of associated intents).

Although single-domain/application conversational bots (e.g. flight booking) are useful, the premise of our research is that the ubiquity of such bots will have more value if they can easily integrate and reuse concomitant capabilities across large number of evolving and heterogeneous devices, data sources and applications (e.g. flight/hotel/car bookings all-in-one bot). With the number of different APIs growing and evolving very rapidly, we need bot development to "scale" in terms of how effectively they can be integrated to APIs.

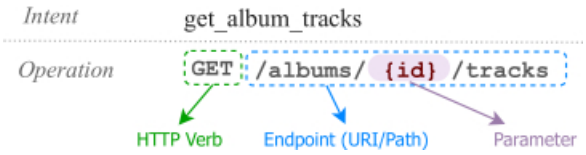
Motivated by the above concerns, we developed the REST2Bot system for the rapid and semi-automated integration of a potentially large and evolving set of intents and APIs [11, 12, 13]. At the heart of the REST2bot system is the idea of providing a middleware that includes knowledge and processing techniques useful to train and automate various activities in bot development life cycle. In a nutshell, this middleware is a set of micro services (e.g., automated canonical utterance generation, automated paraphrasing, automated generation of Webhook functions) for automating bot development process.

2 SYSTEM OVERVIEW

The REST2Bot architecture (Figure 2) consists of a pipeline of components that automate several tasks in the bot development process. First, *API Parser* obtains elements of the APIs (e.g., descriptions, operations, parameters) by parsing *API specifications*. Second, *Canonical Sentence Generator* automatically generates canonical utterances (e.g., "get the tracks of the album with id being 1") for each operation. Third, canonical utterances are paraphrased by *Paraphraser* to obtain diverse utterances to train NLU models by the *Conversation Manager Generator* component on a given bot development platform (e.g., Dialogflow). Finally, *Webhook Generator* generates executable Webhook functions to invoke appropriate APIs based on the intents and entities detected by the chosen bot development framework at runtime.

2.1 API Parser

The *API parser* parses OpenAPI specifications to extract API elements (e.g., operations, parameters). Our prototype is an adaptation of Swagger-Parser in python[1]. OpenAPI specification is an industry standard for REST API documentation. Figure 3 shows an excerpt of an OpenAPI specification for Spotify's API[2]. OpenAPI specification includes operation description, summary, and information about its parameters (e.g., data types, description, and examples). Given that each operation is designed to perform a specific task, each of the extracted operations can be considered a single intent[3] (e.g., getting the list of tracks for a given album):




```
basePath: /v1
host: api.spotify.com
info:
  title: Spotify
  version: v1
paths:
  /albums:  ...
  /albums/{id}:   ...
  /albums/{id}/tracks:
     get:
        description: get an album's tracks
        parameters:
        - {
            name: id,
            in: path,
            description: the Spotify ID for the album,
            required: true,
            type: string
          }
```

Figure 3: Excerpt of Spotify's OpenAPI Specification.

2.2 Canonical Sentence Generator

Canonical Sentence Generator (CSG) automatically translates an operation to canonical templates (e.g., *"get the tracks of the album with id being <<id>>"*)[4]. Generating canonical sentences is essential since they represent corresponding expressions for operations, and later they are paraphrased by the *Paraphraser* component to generate training samples required for training NLU models. The following approaches are used to create canonical sentences:

- **API-Description Transformer.** CSG relies on a set of handcrafted heuristics to automatically transform a description of a given operation to a canonical template [11]. The description of an operation usually consists of several sentences (e.g., *"gets an album's tracks by a given album id. See the following page:"*) describing the functionality of the operation. In short, the description of the operation is split into its sentences (e.g., "gets an album's tracks by a given album id.", "See the following page:"). Then the first sentence starting with a verb (e.g., *"gets an album's tracks by a given album id"*) is chosen as a potential canonical sentence, and its verb is converted to imperative form (e.g., *"get an album's tracks by a given album id"*). Next parameters are injected into the candidate sentence (e.g., *"gets an album's tracks by album id being <<id>>"*) based on a set of hand-crafted rules [11] (e.g., replacing entity mentions like *"given album id"* with a phrase with placeholders like *"album id being <<id>>."*).
- **Neural Translator.** Canonical sentence generated by *API-Description Transformer* are of high quality, but many operations lack descriptions [11]. Since not all operations contain proper descriptions, REST2Bot relies also on the *Neural Translator* proposed in [11]. *Neural Translator* is based on encoder-decoder architecture to directly translate an operation (`GET /customers`) to a canonical template (e.g., "get the list of customers"). REST2Bot uses the same encoder-decoder architecture[5] and training process (e.g., training dataset, hyper-parameters, resource-based delexicalization [11]).

Having generated canonical templates, the placeholders (entities) in the canonical templates are replaced with values. Values are sampled using various methods proposed in [11, 14]. Examples includes parameter example values extracted from Swagger specification. Moreover, for entity types (e.g., restaurants), REST2Bot uses knowledge graphs (e.g., Wikidata [10]) to obtain sample values (e.g., KFC, Domino's).

2.3 Paraphraser

Given that human language is rich and an intent can be expressed in countless ways, having a *diverse* set of utterances is of paramount importance [13]. A lack of variations in training samples may result in bots making incorrect intent detection or entity resolution, and therefore perform undesirable (even dangerous) tasks (e.g., pressing the accelerator instead of the brake pedal in a car) [13]. Existing solutions for paraphrasing involve either automated or crowdsourcing techniques [9, 12, 13]. REST2Bot is integrated with popular crowdsourcing platforms (e.g., figure-eight[6], MTurk[7]), and can automatically create paraphrasing tasks on the specified platform, and train bots based on collected paraphrases. Moreover, *Paraphraser* relies on three paraphrasing components to obtain diverse variations of each canonical sentence utterance:

- **Common Prefix/Postfix Concatenation.***Paraphraser* relies on a set of handcrafted common prefixes/postfixes (e.g., *"can you please", "please", "I want to"*) which can be concatenated with a canonical sentence (e.g., *"create a new album"*) without changing its meaning (e.g., *"can you please create a new album"*).
- **Statistical Paraphrasing**The proposed system also relies on a statistical paraphrasing system called Apache Joshua [7]. Apache Joshua is a statistical machine translation decoder for phrase-based machine translation, and it relies on Paraphrase Dataset (PPDB) to generate paraphrases [3].
- **Neural Paraphrasing.**Inspired by language pivoting in (Neural Machine Translation) NMT systems , *Paraphraser* uses two pre-built NMT models proposed in [8]: one for translating English sentences to German, and the other one for translating from German back to English.

2.4 Conversation Manager Generator

Conversation Manager Generator (CMG) is used to instantiate a conversation manager in a third-party bot development platform (e.g. *Dialogflow*). To support this functionality, we develop extensions called *generator plugins* embedded inside *CMG* component. Each plugin is a program that takes as input the generated training data (e.g. annotated utterances) by *Paraphraser*, and exports them to the conversation manager in order to train included NLU model. This is done by exploiting the REST APIs provided by the bot development platform (e.g. *Dialogflow*).

2.5 Webhook Generator

Webhook Generator (WG) automatically generates webhook functions written in developer's selected programming language (e.g. Python[8]). For each detected user intent and its associating parameter values, the generated Webhook invokes corresponding APIs and returns the response. The output of *Webhook Generator* is a fully-functional and ready-to-deploy source code for a webhook server. This source code can be reused and customized by bot developers.

3 DEMONSTRATION SCENARIO

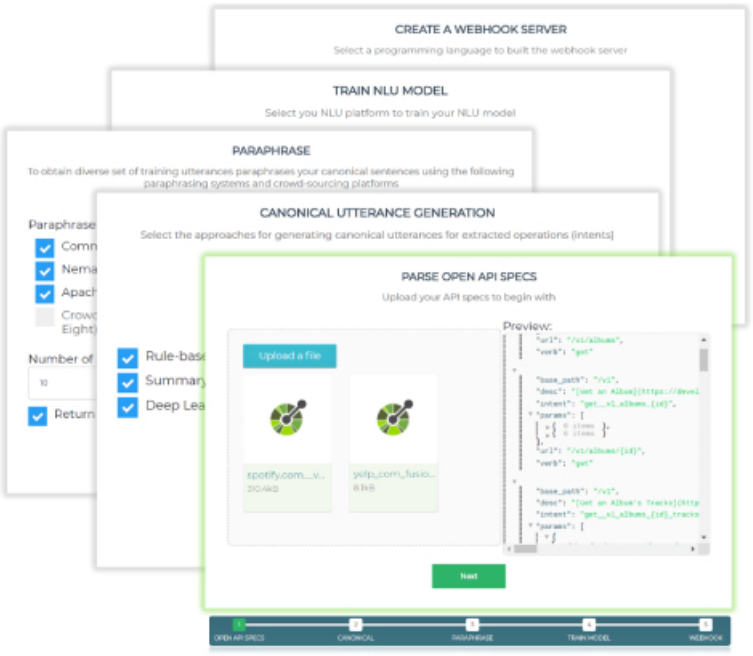


Figure 4: REST2Bot User Interface.

In this demonstration, we will show how REST2Bot facilitates the bot development process in existing bot development platforms. REST2Bot is offered as both a RESTful API and web-based interface. REST2Bot gives the user control over the settings of the different components. This includes selecting algorithms for creating canonical sentences (e.g., deep learning approach, rule-based approach), paraphrasing models (e.g., Apache Joshua, appending common-prefix), bot development platform (e.g., Wit.ai, Dialogflow), and programming framework (e.g., python flask, spring boot) to build the webhook server. The demonstration scenario consists of the following part:

- **Training Dataset Generation.** In REST2Bot, developer starts with providing a list of API specifications (e.g. Spotify, Yelp) chosen for building the bot. REST2Bot extracts all operations from the given set of APIs. Next, based on selected approaches for translating (*API-Description Transformer* and *Neural Translator*), operations are translated to canonical sentences using Canonical Sentence Generator (CSG) component. Then, paraphrasing methods (*common prefix, statistical, and neural paraphrasing*) are selected by bot developer to diversify the training dataset which is handled by Paraphraser component. Since paraphrases are generated automatically, REST2Bot also scores the generated paraphrases (using Paraphraser component) to only keep high quality paraphrases based on the user provided score threshold (similarity of a canonical utterance and a given paraphrase is calculated by the method proposed in [2]). The outcome is a dataset containing natural language utterances to train conversation-enabled services such as bots (e.g. on-premise development platforms) and IoT devices (e.g. smart home appliances, aged-care gadgets).
- **Automate Bot Development.** Using the generated training dataset, REST2Bot trains NLU model (as the main component within conversation manager) by defining intents (e.g. PlayMusic, FindRestaurants), entity types (e.g. album name, restaurant location) in a supported bot platform (e.g., DialogFlow) chosen by bot developer. Next, REST2Bot automatically creates Webhook functions to map intents to APIs in a given

based programming framework (e.g., python flask). This process is handled by CMG and WG components from REST2Bot architecture. Finally, in the demo, the bot users will be able to interact with the generated bot.

ACKNOWLEDGMENTS

This research was supported fully by the Australian Government through the Australian Research Council's Discovery Projects funding scheme (project DP1601104515).

4 CONCLUSION AND FUTURE WORK

This paper aimed at addressing an important shortcoming in current approaches for building bots, namely automatic training data generation and building Webhook functions to invoke APIs. Future work includes adding new plugins (e.g., more bot development platforms, programming frameworks for webhook servers). Moreover, extensions of REST2Bot will include developers sharing of training datasets to minimize the cost of paraphrasing in case of using crowdsourced paraphrasing. Another feasible extension is deploying the webhook server, which can be done automatically using cloud computing platforms (e.g., Microsoft Azure, Google Cloud).

REFERENCES

[1] Giovanni Campagna, Rakesh Ramesh, Silei Xu, Michael Fischer, and Monica S Lam. 2017. Almond: The architecture of an open, crowdsourced, privacy-preserving, programmable virtual assistant. In *WWW'26*. 341–350. Navigate to

[2] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, *et al.* 2018. Universal sentence encoder. (2018). Navigate to

[3] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. 2013. PPDB: The paraphrase database. In *NAACL-HLT*. 758–764. Navigate to

[4] Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. 2005. Bidirectional LSTM networks for improved phoneme classification and recognition. In *International Conference on Artificial Neural Networks*. Springer, 799–804. Navigate to

[5] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780. Navigate to

[6] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*(2015). Navigate to

[7] Matt Post, Yuan Cao, and Gaurav Kumar. 2015. Joshua 6: A phrase-based and hierarchical statistical machine translation system. *The Prague Bulletin of Mathematical Linguistics* (2015). Navigate to

[8] Rico Sennrich, Alexandra Birch, Anna Currey, Ulrich Germann, Barry Haddow, Kenneth Heafield, Antonio Valerio Miceli Barone, and Philip Williams. 2017. The University of Edinburgh's Neural MT Systems for WMT17. In *Proceedings of the 2nd Conference on Machine Translation*. Navigate to

[9] Yu Su, Ahmed Hassan Awadallah, Madian Khabsa, Patrick Pantel, and Michael Gamon. 2017. Building Natural Language Interfaces to Web APIs, In *In Proceedings of the 26th ACM international conference on Information and knowledge management (CIKM '17)*. Navigate to

[10] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85. Navigate to

[11] Mohammadali Yaghoubzadeh and Boualem Benatallah. 2020. Automatic Canonical Utterance Generation for Task-OrientedBots from API Specifications. In *Advances in Database Technology-23rd International Conference on Extending Database Technology (EDBT)*. Navigate to

[12] Mohammadali Yaghoubzadeh, Boualem Benatallah, Fabio Casati, Moshe Chai Barush, and Shayan Zamanirad. 2020. Dynamic Word Recommendation to Obtain Diverse Crowdsourced Paraphrases of User Utterances. In *IUI*. Navigate to

[13] Mohammadali Yaghoubzadeh, Boualem Benatallah, Moshe Chai Barush, and Shayan Zamanirad. 2019. A

Study of Incorrect Paraphrases in Crowdsourced User Utterances. In *NAACL'19*. Navigate to

[14] Shayan Zamanirad, Boualem Benatallah, Moshe Chai Barukh, Fabio Casati, and Carlos Rodriguez. 2017. Programming bots by synthesizing natural language expressions into API invocations. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 832–837. Navigate to

FOOTNOTE

[1] **https://github.com/Trax-air/swagger-parser (https://github.com/Trax-air/swagger-parser)**

[2]Spotify Web API endpoints provides access to music artists, albums, and tracks

[3]current implementation does not support intents requiring API compositions

[4]a canonical template is a sentence in which entities are replaced with placeholders

[5]two layers of Bidirectional Long-Short Term Memory (BiLSTM) [4] for encoding, and two layers of Long-Short Term Memory (LSTM) [5] for the decoder using attention mechanism [6]

[6] **https://www.figure-eight.com (https://www.figure-eight.com)**

[7] **https://www.mturk.com/ (https://www.mturk.com/)**

[8]The current implementation this stage supports template codes written in python.

WWW '20 Companion, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7024-0/20/04.
DOI: **https://doi.org/10.1145/3366424.3383551 (https://doi.org/10.1145/3366424.3383551)**