Combining Non-Expert and Expert Crowd Work to Convert Web APIs to Dialog Systems

Ting-Hao K. Huang
tinghaoh@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA USA

Walter S. Lasecki
wlasecki@cs.rochester.edu
University of Rochester
Rochester, NY USA

Alan L. Ritter
ritter.1492@osu.edu
The Ohio State University
Columbus, OH USA

Jeffrey P. Bigham
jbigham@cs.cmu.edu
Carnegie Mellon University
Pittsburgh, PA USA

Abstract

Thousands of web APIs expose data and services that would be useful to access with natural dialog, from weather and sports to Twitter and movies. The process of adapting each API to a robust dialog system is difficult and time-consuming, as it requires not only programming but also anticipating what is mostly likely to be asked and how it is likely to be asked. We present a crowd-powered system able to generate a natural language interface for arbitrary web APIs from scratch without domain-dependent training data or knowledge. Our approach combines two types of crowd workers: non-expert Mechanical Turk workers interpret the functions of the API and elicit information from the user, and expert oDesk workers provide a minimal sufficient scaffolding around the API to allow us to make general queries. We describe our multi-stage process and present results for each stage.

Introduction

In this paper, we propose a crowd-powered framework to interact with web APIs through a conversation interface. We illustrate two major phases: the on-line phase and off-line phase. In the on-line phase, we propose a crowd-powered system architecture that implements real-time conversational interaction between users and web APIs; in the off-line phase, we propose a crowdsourcing workflow that "translates" one web API to real-world questions, which is the foundation to empower our real-time dialog system.

We leverage prior work in human computation to bootstrap the creation of automated dialog systems. Human computation has been shown to be useful in many areas, including writing and editing (Bernstein and et al. 2010), and image description and interpretation (Bigham, et, and al. 2010; Von Ahn and Dabbish 2004). Crowd responses can also be gathered very quickly. VizWiz elicits nearly real-time responses from the crowd using a queuing model (Bigham, et, and al. 2010). Chorus is a system that enables a real-time conversation with a crowd of workers as if they were a single conversational partner (Lasecki and et al. 2013).

On-line Phase: Working System

In this paper, we illustrate a crowd-powered framework that operates an API through a real-time conversational interface, which is inspired by the slot-filling process of modern dialog systems. The process can be addressed in 3-stages (Figure 1): First, in the "parameter value elicitation" stage, our system asks questions to request for the value of a target parameter. Second, in the "parameter value extraction" stage, our system recruits crowd to extract the value of a target parameter from the user's answer. For all useful parameters, the system is able to run through stages 1 and 2 to collect their values, and the API handler is then able to decide when to trigger the API based on the parameter-filling status. Finally, when we get the results returned by the API, the response format is usually technical, e.g., JSON or XML. In the "response generation" stage, we recruit workers to convert the result into a natural language. From the perspective of modern dialog systems, in our framework, the API acts like a frame, the parameters act like slots, and the API handler is in charge of dialog management.

Stage 1: Parameter Value Elicitation In this stage, questions are presented to the user and their answers are recorded. The key piece of this component is a knowledge-base that maintains the information of all parameters and associated questions, which is prepared in the off-line phase. When the system is running, the parameter value elicitation component selects questions to ask the user based on the current status and the dialog history, the dependency and importance of parameters, and a pre-defined policy. API query parameters are then extracted from the user's response.

Stage 2: Parameter Value Extraction We propose a crowd-powered approach to extract the parameter values from natural language text. There are two underlying tasks in this stage: understanding the semantics of the given parameter by reading its technical descriptions, and extracting the value from a natural language conversation. Unskilled crowd workers are recruited to accomplish these two tasks.

Stage 3: Response Generation The last stage of the dialog system process is to convert the API response object, e.g., JSON or XML data, into natural language text. There are two underlying subtasks in this stage: understand the information (which is in a machine-processable data format), and describe the information in a natural language. We also propose recruiting unskilled workers to perform this task.

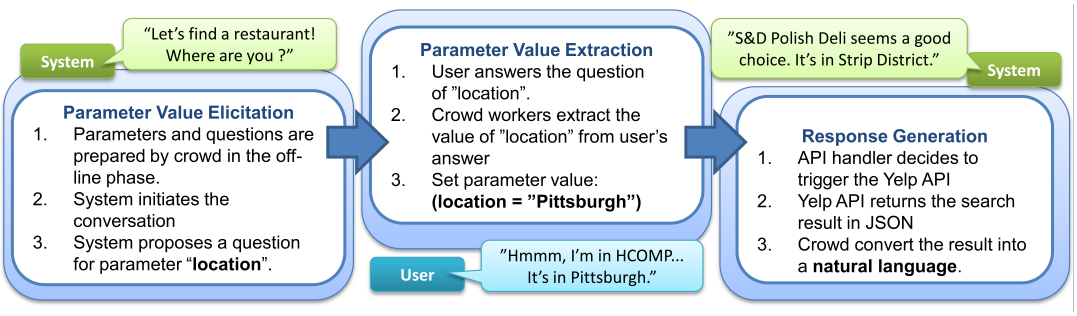

Figure 1: On-line Phase: Crowd-powered Real-time Conversational Interface to Operate API

Off-line Phase: Preparation Process

The on-line system above needs 3 pieces of knowledge: First, the technical details of the API; Second, the parameters that were selected for the dialog system and the relations between parameters; Finally, questions are associated with each selected parameter. In the off-line phase, we design a workflow that allows unskilled and skilled workers to work together to collect these facts. The process is:

Stage 1: Question & Answer Collection First, we collect the questions associated with the task. We ask crowd workers the following: "A friend wants to `TASK_DESCRIPTION` and is calling you for help. Please enter one question you would ask them to help accomplish their task." After each worker asks a question, we ask the worker to write down a possible answer. This process is then repeated to collect the next question-answer (QA) pairs.

Stage 2: Parameter Filtering In Stage 2, we perform a filtering process with an unskilled crowd to reduce candidate parameter set. Scalability is a challenge that often arises when applying general voting mechanisms to API parameters. Our solution is to adopt a filtering step before the voting stage. We show all of the parameters (along with the parameters name, type, and description) to the workers, and ask them to select all of the "unnatural" items which are not very likely to be mentioned in a real conversation, or are obviously designed for computers and programmers.

Stage 3: QA-Parameter Mapping In Stage 3, we map the QA pairs collected in Stage 1 against the remaining parameters from Stage 2. We display one QA pair along with all parameters, and ask workers which parameters' information is provided in the answer. For each QA pair, the workers are first asked to pick the best parameter, and then rate their confidence level. This process not only finds a good set of parameters for the dialog system application, but also find good questions associated with each selected parameter.

Wrapping APIs with Expert Workers Web APIs lack a unified interface to access their parameters and generally require reading and understanding documentation. We therefore recruit expert workers from oDesk to implement an API Handler based on the given API and selected parameters. This kind of work requires basic programming skills, and thus cannot be distributed to unskilled workers. The API Handler should be robust enough for basic errors and provide some level of validation of the input values.

Experiments and Discussion

To explore the feasibility of our framework, we conduct experiments with the well-known Yelp restaurant recommendation API, on both off-line and on-line phase.

For the off-line phase, we collected 40 QA pairs for the task "find a restaurant to eat at", and applied a voting process in which Turk workers filtered out 13 of 22 parameters. The remaining 9 parameters are further matched with the 40 QA pairs by crowd. Finally, we select the top 3 matched parameters, "category_filter", "term", and "location" to create the on-line system; Based on the experimental results of the off-line phase, we run experiments for the on-line phase to test the ability of the non-expert crowds to understand the semantics of the parameters, to extract parameter values, and to convert data into natural text. Most values extracted by crowd were semantically correct and valid Yelp parameters. Around 65% of responses created by crowd mention at least one of the top two restaurants in the Yelp search results.

For oDesk workers, we posted a $35.00 USD job, waited 6 hours for responses, and then chose the least expensive offer ($15.00 USD). We followed up with the worker to describe additional details. The worker returned a completed Python script within 12 hours that implemented the whitelist for the category_filter parameter as an associative array.

Our experimental results demonstrate the feasibility of our approach. In the future, these dialog systems could be generated dynamically, as the need for them arises, making automation a gradual process that occurs based on user habits. Since it might require more than one session to elicit enough information from the user to learn the necessary parameters, the system can learn over time, without having to lock the user into a singe session. Intent recognition can also aid this lazy-loading process by determining a user's goal and drawing on prior interactions, even by others, to collaboratively create these systems.

References

Bernstein, M., and et al. 2010. Soylent: a word processor with a crowd inside. In *UIST*.

Bigham, J. P.; et; and al. 2010. Vizwiz: nearly real-time answers to visual questions. In *UIST*.

Lasecki, W. S., and et al. 2013. Chorus: a crowd-powered conversational assistant. In *UIST*.

Von Ahn, L., and Dabbish, L. 2004. Labeling images with a computer game. In *CHI*.