Almond: The Architecture of an Open, Crowdsourced, Privacy-Preserving, Programmable Virtual Assistant

Giovanni Campagna Rakesh Ramesh Silei Xu Michael Fischer Monica S. Lam
Computer Science Department
Stanford University
Stanford, CA 94305
{gcampagn, rakeshr, silei, mfischer, lam}@cs.stanford.edu

ABSTRACT

This paper presents the architecture of Almond, an open, crowdsourced, privacy-preserving and programmable virtual assistant for online services and the Internet of Things (IoT). Included in Almond is Thingpedia, a crowdsourced public knowledge base of open APIs and their natural language interfaces. Our proposal addresses four challenges in virtual assistant technology: generality, interoperability, privacy, and usability. Generality is addressed by crowdsourcing Thingpedia, while interoperability is provided by ThingTalk, a high-level domain-specific language that connects multiple devices or services via open APIs. For privacy, user credentials and user data are managed by our open-source ThingSystem, which can be run on personal phones or home servers. Finally, we create a natural language interface, whose capability can be extended via training with the help of a menu-driven interface.

We have created a fully working prototype, and crowdsourced a set of 187 functions across 45 different kinds of devices. Almond is the first virtual assistant that lets users specify trigger-action tasks in natural language. Despite the lack of real usage data, our experiment suggests that Almond can understand about 40% of the complex tasks when uttered by a user familiar with its capability.

1. INTRODUCTION

Virtual assistants can greatly simplify and enhance our lives. Today, a two-year old can play her favorite song by just saying "Alexa, play the happy song", before she learns how to use a computer. As the world gets wired up, we can simply use natural language to ask our virtual assistant to interact with social media, purchase movie tickets, and even manage our financial and medical records.

A virtual assistant may ultimately be our interface to all digital services. As an intermediary, the virtual assistant will see all our personal data and have control over the vendors we interact with. It is thus not a surprise that all the major companies, from Amazon, Apple, Facebook, Google, to Microsoft, are competing to create the best virtual assistant. Just as there is a dominant search engine today, will there be a single proprietary company that provides the world's dominant virtual assistant? This raises a lot of questions, including privacy, interoperability, and generality.

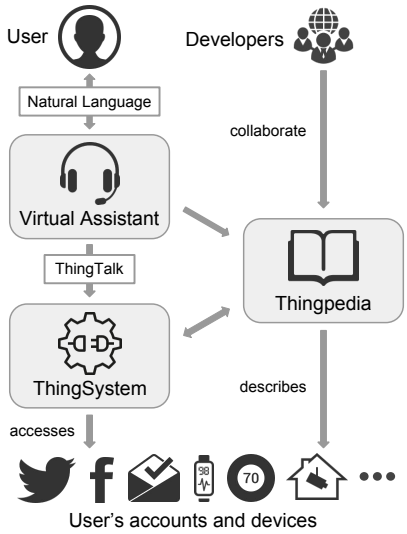


Figure 1: The architecture of Almond.

To answer these questions, we propose the architecture for *Almond*, shown in Figure 1, an open, crowdsourced, privacy-preserving and programmable virtual assistant. Users can ask Almond in natural language to perform any of the functions in the *Thingpedia* knowledge base. Thingpedia is a crowdsourced repository containing natural language interfaces and open APIs for any device or service. For privacy, all the personal data and credentials are stored in the *ThingSystem*, whose code is open-source and can be run on personal phones or home servers. The ThingSystem runs programs written in *ThinkTalk*, a high-level language we developed to connect open APIs from different services together. This high-level abstraction makes it feasible to automatically translate natural language to ThingTalk code.

As far as we know, Almond is the only virtual assistant that lets users specify trigger-action commands in natural language. For example, the user can instruct Almond to "notify him if motion is detected by the security camera, when the alarm is on". Almond is programmable: it can translate such a sentence into ThingTalk code, as long as the functions "notify", "detect motion on the security cam-

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052562

era", and "is alarm on" are available in Thingpedia. The closest available system is If This Then That (IFTTT) [14] which provides users with a web interface to connect two APIs together. However IFTTT does not have a natural language interface, nor a formal language like ThingTalk, and it cannot handle conditions like "when the alarm is on".

1.1 Why Open & Crowdsourced?

We advocate openness and crowdsourcing so all internet of things can interoperate, developers can compete and innovate openly, and users can have their data privacy. Technically, crowdsourcing is also necessary to generate sufficient natural language training data.

Privacy. For virtual assistants to comprehensively serve the user, they need to handle all their personal information and be given permission to interact on their behalf. However, data like bank accounts or medical records are highly sensitive. It is inappropriate to use virtual assistants on services like Facebook Messenger that own their users' data. ThingSystem, on the other hand, is open-source and can be run locally on the phone, on a home server, or as a web service. As open source, ThingSystem can be code reviewed and checked for malicious behavior.

Interoperability. Keeping the system open also improves interoperability. Having multiple proprietary standards, such as Google Weave [10] and Apple Homekit [2], serves neither the device makers nor the consumers. Owners of the standards have too much control over their partners, and consumers are locked into specific brands. In contrast, an open system like ThingTalk is designed to be modular to enable interoperability across many different discovery, configuration and communication protocols. In ThingTalk, the same commands can be applied to a Jawbone wristband tracker or a Fitbit, and the different details are handled transparently. Additionally, because Almond is open-source, multiple competing virtual assistants can be developed on top of a shared codebase.

Generality. Significant research and development is still needed to create a truly usable virtual assistant. Inspired by the open-source BSD Unix and Linux efforts, we make Almond open-source as an attempt to bring researchers and developers together to create an open system competitive to proprietary systems. Similarly, Thingpedia is inspired by the same values that power Wikipedia including community, openness, and ease of use. Besides Wikipedia, crowdsourcing has proven successful for building large knowledge bases in a multitude of different areas like structured content (DBpedia [3]), mathematics (Wolfram Mathworld [29]), genomics (UCSC Genome Browser [17]), etc. By supporting device interoperability and making the entire virtual assistant code base available, we hope that Thingpedia can attract all service providers, device makers, and hobbyists to help create the largest encyclopedia of things. Having all this knowledge in the public domain can promote open competition and thus fuel innovation.

Usability. Ideally, we wish to specify the desired interaction in natural language, and have such input be automatically translated into code. Natural language parsing is typically based on machine learning, which requires a large volume of annotated user data. The SEMPRE system is a semantic parser that addresses the lack of sufficient user data by leveraging crowdsourcing [28]. Although SEMPRE was originally designed to handle semantic queries, the Almond semantic parser extends SEMPRE to handle trigger-action commands to generate executable programs. Further, Almond has the only semantic parser that can generate trigger-action commands that requires composition, i.e. parameters be passed from the triggers to the action. Nonetheless, Almond today is not accurate enough to handle all inputs, and we have created a menu-driven interface that lets users train Almond with relative ease. Over time, the crowdsourced training information can feed into machine-learning algorithms to further improve Almond's natural language capability.

1.2 Contributions

Architecture of an open virtual assistant. Almond is a conversational agent built on top of a system with three main components:

1. Thingpedia: a public knowledge base of natural language interfaces and open APIs to services and devices.
2. ThingTalk: a high-level language that can connect web services and Internet of Things in a single line of code.
3. ThingSystem: an open-source system that manages users' credentials and data and executes ThingTalk programs. It can be run on users' devices to preserve privacy.

Methodology to acquire training data for virtual assistants. We extended the technique introduced by Wang et al. [28] to generate training samples for trigger-action commands from Thingpedia entries for new devices.

The first virtual assistant that understands trigger-action commands in natural language. We have created and released Almond, which currently has a repertoire of 45 devices and 187 functions. With a training set consisting of over 7000 sentences, Almond is found to return the desired program among the top 5 choices for 53% of the valid inputs written by users familiar with its capability.

1.3 Paper Organization

In Section 2, we describe what the virtual assistant Almond can do and its user interface. We describe an overview of the underlying system architecture in Section 3. We present an overview of our algorithm to crowdsource natural language training in Section 4. We describe the results of crowdsourcing Thingpedia in Section 5, and the experimentation of Almond in Section 6. Finally, we present related work and conclude.

2. ALMOND VIRTUAL ASSISTANT

This section describes what Almond can do and how users interact with Almond. Almond is unique in that it can perform trigger-action tasks based on natural language commands. Furthermore, it is fully extensible: its capabilities grow as Thingpedia grows.

2.1 Generality

Almond derives its generality from Thingpedia, a repository designed to capture interfaces to all the different web services and IoT devices. While Almond can access publicly available services like the weather channel or Uber, its strength is in managing and interfacing with personal web accounts and devices with privacy[1]. For example, Al-

[1]In the rest of the paper, we use the term *devices* to refer to physical devices and web services.

mond can notify a user of low bank balances without having the user disclose his credentials to a third party. Almond prompts the user for the credential just once, when a user initiates the action, and stores the credential on, for example, the user's phone. Almond can also handle physical devices, irrespective of whether they use Bluetooth, WiFi, etc. The user tells Almond to initiate discovery of devices, and Almond prompts the user for the credentials to each device found.

2.2 Expressiveness

At the basic level, Almond lets users access a wide collection of devices from a single text-based user interface. Users can simply tell Almond to "tweet a message" or "turn on the light", thus avoiding the need to navigate different interfaces in different apps. Almond can be "programmed" to connect functions from different devices together to create trigger-action tasks.

We categorize the commands that Almond accepts into *primitive* and *compound* operations. The most basic primitive command is a direct *action* or *query* to a device. Almond also supports standing queries or *monitors* to notify the user when an event of interest is triggered. Users can add one or more comparisons to filter the result of the queries or monitors.

Compound commands involve two or more functions. For example, as shown in Figure 2, one can email the daily weather update by saying "every day at 6am get latest weather and send it via email to bob", where "every day at 6am" is a trigger, "get latest weather" is a query, and "send email to bob" is an action.

Class	Type	Examples
primitive	action	send email to bob
	query	get my latest email
	monitor	notify me when I receive an email
	filtered monitor/query	notify me when I receive an email if the subject contains deadline
compound	trigger+query	every day at 6am get latest weather
	trigger+action	every day at 6am send email to bob
	query+action	get latest weather and send it via email to bob
	trigger+query +action	every day at 6am get latest weather and send it via email to bob

Figure 2: Categories of commands accepted by Almond

2.3 User Interface

Almond appears to the user as a conversational agent with a chat-based interface, where users can input their commands in natural language. There are 3 key challenges we addressed when designing Almond: (1) novice users do not know what can be done on Almond, (2) users find it difficult to express complex compound commands in natural language, and (3) users get frustrated when the natural language fails.

Discovery. To show novice users the scope of Almond's capability, we developed a "cheat sheet" of Almond commands. The cheat sheet lists entries by interfaces along with a small icon so users can visually scan the entire document at a glance. Under each interface is a list of commands sorted by triggers, queries, actions listed in their natural form through the keywords WHEN, GET, DO, respectively. Additionally, we also added a list of popular commands to show how Almond could be used. An example cheat sheet with 4 devices is shown in Figure 3.

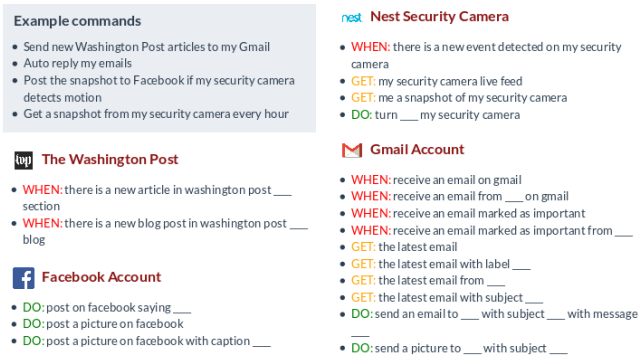
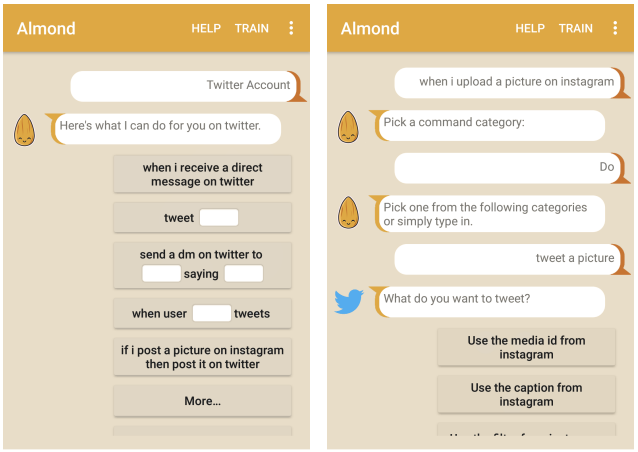


Figure 3: Example cheat sheet for 4 devices



(a) List of commands for Twitter

(b) Interactive creation of compound commands

Figure 4: Two screenshots of the Almond user interface

Besides the cheat sheet, users can also explore the application by clicking the *help* button, whereupon Almond lists the supported devices by categories in the form of a menu. Then, they can simply navigate the menu by selecting the device and choosing the command from the list of popular commands sorted by device (Figure 4a). After selecting the command, they are prompted to fill in the blanks or answer slot filling questions to complete them.

Creation. To help power users maximize the capabilities of Almond, we provided a simple menu-based approach to create compound commands. Users can choose the primitive commands that make up the compound command by clicking on the corresponding WHEN, GET, DO buttons. Almond then prompts the user for the composition (Figure 4b) and asks follow up questions to fill in the rest of the blanks.

Training. To improve the natural language, Almond provides an alternative way to be corrected when it makes a mistake. By clicking the *train* button, users can pick the right command from an exhaustive list of candidates when Almond fails to match in the top 3 choices. The correct match is then stored in Thingpedia and Almond can learn from it. If there is still no correct match, users will be asked to rephrase the sentence.

2.4 Use Case Scenarios

To give the readers a sense of Almond's capability, we describe four use case scenarios assembled from the various rules our beta users have created.

Simple, universal interface. Alice likes Almond because it provides a uniform interface across many devices. When she wakes up, she tells Almond to "Turn on the lights" and asks "What is the weather?", "What's the Uber price to work?". Once she arrives at work she asks Almond to "Set my phone to silent when an event on my calendar begins".

Quantified self. Bob uses Almond to monitor his habits, knowing full well that he can switch devices while maintaining a common interface. He currently uses Almond as an interface with his Jawbone UP to find "How much did I sleep last night?", "How many steps have I taken today?". Because health data is sensitive, Bob cares about privacy and likes that the data is stored only on this phone. Additionally, by using Almond, Bob can switch to Fitbit whenever he wants without having to learn a new interface.

Media Filtering. Carol is a software developer who saves time by using Almond to stream all her social accounts customized to her preferences. For example, she tells Almond to "Monitor @justinbieber on Twitter" so she will only be notified when @justinbieber tweets. Almond lets Carol focus on what she is interested on, across all social media. She also use Almond for work by setting "Monitor pushes to Github" and "Monitor Slack updates on channel #urgent".

Home Automation. Dan likes to use Almond to connect his gadgets easily. To save energy, he asks Almond to "Turn off my HVAC if the temperature is below 70 F." His wife Eve is a bird lover. So, he sets up a hummingbird feeder and points a camera at it, and asks Almond to "Send the video to @eve via the Omlet chat if there is motion on my camera".

Summary. Via a simple and uniform chat-based interface, Almond can help users with many tasks, from simple commands, to customizing interfaces, and programming connections across different devices, all while preserving privacy. If the service or device of interest is not in Almond's repertoire, an average programmer can extend Almond by adding the interfaces of interest into Thingpedia with a reasonable effort.

3. SYSTEM ARCHITECTURE

In this section, we provide an overview of Almond's underlying system architecture.

3.1 Thingpedia

Thingpedia is an encyclopedia of applications for the Internet of Things. Just like how Wikipedia stores knowledge about the world, Thingpedia stores knowledge about devices in the world. Wikipedia is organized around articles; Thingpedia is organized around *devices*, such as Twitter, a light bulb, or a thermostat. Each device has a *entry* on Thingpedia. A Thingpedia entry stores the natural language interface that represent how humans refer to and interact with the device, and the executable specification corresponding to the device API.

A full list of attributes of the entry is shown in Figure 5. Each entry includes information such as a version number, package name, communication protocols, and discovery information. In addition, it has one or more functions, which can be *triggers* (which listen to events), *queries* (which retrieve data) and *actions* (which change state). These functions are implemented by wrapping the low-level device API in a JavaScript package, which can be downloaded by ThingSystem on demand. For each function, the manufacturer also provides the parameter specification, some natural language annotations, which we describe in Figure 6, and a few example sentences to activate it.

Thingpedia also hosts the anonymized natural-language commands crowdsourced from the users, which are used to provide suggestions for other users and to train the assistant.

3.2 ThingTalk

ThingTalk is a high-level language we developed to connect the Internet of Things. It connects APIs of devices together, while hiding the details of configuration and communication of the devices.

For example, here is a ThingTalk program that posts Instagram pictures with hashtags containing "cat" as pictures on Facebook:

$$\begin{aligned}&\text{@instagram.new\_picture}(\textit{picture\_url}, \textit{caption}, \textit{hashtags}),\\ &\quad \text{Contains}(\textit{hashtags}, \text{“cat”})\\ &\Rightarrow \text{@facebook.post\_picture}(\textit{text}, \textit{url}),\\ &\quad \textit{text} = \textit{caption}, \textit{url} = \textit{picture\_url}\end{aligned}$$

The above code reads as "if I upload a picture on Instagram with certain *picture_url*, *caption* and *hashtags*, and the *hashtags* array contains the value 'cat', then post a picture to Facebook, setting the *text* from the *caption* and the *url* from the *picture_url*".

A ThingTalk program is a *rule* of the form:

$$\textit{trigger}\ [, \textit{filter}]^*\ [\Rightarrow \textit{query}\ [, \textit{filter}]^*]^*\ \Rightarrow \textit{action}$$

where each of *trigger*, *query*, *action* is an invocation of a Thingpedia function.

The trigger denotes when the rule is evaluated, the query retrieves data and the action produces the output. The trigger and the query can be filtered with equality, containment and comparison operators. Composition occurs by binding the trigger or query parameters to a variable name and using them to set the action parameters.

Primitive Almond commands are expressed in ThingTalk using degenerate rules with the builtin trigger **now**, which indicates that the rule is to be executed now, and the builtin action **notify**, which indicates that the result is to be reported to the user. Figure 7 summarizes the correspondence between Almond commands and ThingTalk forms.

For now, devices can only be referred to by their type. If the user has multiple devices of the same type, Almond asks the user to choose the one to operate on. In the future, we plan to extend ThingTalk to name devices by location, by user-defined labels, or by using contextual information.

Attribute	Definition	Example
Class and version	A namespaced identifier referring to a specific implementation of the API	com.lg.tv.webos2, version 20
Global name	A unique name that can be used to refer to the device, to configure it or get help about it	"lg webos tv"
Types	Generic category names that can be used to refer to the device	"tv", "powered device"
Configuration method	How is the device discovered and configured, such as OAuth, UPnP, Bluetooth, etc.	UPnP
Discovery descriptors	Low-level identifiers, specific to a discovery protocol, which are used to associate an entry with a physical device	UPnP *serviceType*: "urn:lge-com:service:webos-second-screen:1"
Functions	Triggers, Queries, Actions	set_power(*power*), play_url(*url*), set_volume(*percent*), mute(), unmute()

Figure 5: Attributes describing a Thingpedia entry, with the example of a LG Smart TV.

Annotation	Definition	Example
Canonical form	An English-like description of the function with no parameters	"set power on lg webos tv"
Parameters	The parameters to the function, each with name and data type	*power*: Enum(on, off)
Follow up questions	A question for each required parameter that Almond will ask if it is missing	*power*: "Do you want to turn the TV on or off?"
Example sentences	Full sentence templates that activate the function immediately, with parameters to fill by the user; these sentences are used to bootstrap the natural language learning algorithm	"turn my lg tv $power", "switch my tv $power", "set my tv to $power" (where $power is replaced by on or off depending on what the user chooses)
Confirmation	A richer English description of the function which is presented to the user so he can confirm the command before it's executed	"turn $power your LG WebOS TV"

Figure 6: The natural language entry for a Thingpedia function, with the example of set_power on the LG Smart TV.

Class	Type	ThingTalk
primitive	action	**now** ⇒ *action*
	query	**now** ⇒ *query* ⇒ **notify**
	monitor	*trigger* ⇒ **notify**
compound	trigger+query	*trigger* ⇒ *query* ⇒ **notify**
	trigger+action	*trigger* ⇒ *action*
	query+action	**now** ⇒ *query* ⇒ *action*
	trigger+query +action	*trigger* ⇒ *query* ⇒ *action*

Figure 7: The different ThingTalk forms, and how they map to Almond commands.

3.3 ThingSystem

While Thingpedia contains all public information, each user has their own ThingSystem to store information about their configured devices and commands. ThingSystem is portable and can run on the phone or in the cloud. ThingSystem has two main roles: to help the user configure and manage his devices, and to execute the ThingTalk code corresponding to his commands.

Management of devices. ThingSystem maintains a list of all devices that belong to the user. For each device, ThingSystem stores an instance identifier, the IP or Bluetooth address and the credentials to access it. The list of devices forms essentially a *namespace* for the user, where devices can be recalled by type. The namespace is then used to map the abstract name "twitter" to a specific Twitter account owned by the user, or "tv" to a specific TV and its network address.

Devices are added to the list when the user configures them. This happens explicitly when requested by the user or on demand when used in a ThingTalk command. Configuration involves 4 steps: mapping, loading, authenticating and saving. For example, to configure "twitter" the following actions take place:

1. Mapping: "twitter" gets mapped to its Thingpedia entry ("com.twitter", version: 22, config_type: OAuth).
2. Loading: Code package "com.twitter-v22.zip" is downloaded from the Thingpedia server and loaded.
3. Authenticating: The user is directed to the OAuth login page of Twitter and asked for credentials.
4. Saving: User ID and Access token are added to the namespace and the entry is saved.

Physical devices can also be discovered using general-purpose protocols, such as UPnP or Bluetooth. The ThingSystem listens to the broadcasts from visible devices, collects the discovery descriptors and queries Thingpedia for the corresponding entry. Configuration then proceeds in the same way as before.

ThingTalk execution. ThingSystem contains an evaluation loop that executes ThingTalk code on behalf of the user. It polls the triggers, evaluates conditions and invokes the corresponding queries and actions.

When the user gives a command, the corresponding

ThingTalk	Synthethic sentence	Paraphrase
@builtin.timer(*interval*), *interval* = 1day ⇒ @thecatapi.get(*picture_url*) ⇒ **notify**	Every 1 day get a cat picture	Send me a daily cat picture
@twitter.receive_dm(*author*, *message*) ⇒ @twitter.send_dm(*to*, *message*), *to* = *author*	If I receive a new DM on Twitter send a DM on Twitter to the author saying something	Auto-reply to my Twitter DMs
@washington_post.new_article(*title*, *link*) ⇒ @yandex.translate(*target*, *text*, *translated_text*), *target* = "chinese", *text* = *title* ⇒ **notify**	If a new article is posted on the Washington Post then translate the title to "chinese" with Yandex	Translate Washington Post to "Chinese"

Figure 8: Three examples of ThingTalk programs, their associated synthethic sentence, and a possible paraphrase.

ThingTalk code is first compiled using the type information in Thingpedia. Then the code is analyzed to map each trigger, query and action to a specific pre-configured device in the user namespace. The compiled code is connected to the corresponding JavaScript implementation and then passed to the evaluation loop.

ThingSystem also provides persistent storage of the user programs and their state, so that it can be restarted at any time without losing data or duplicating notifications.

4. LANGUAGE TO CODE

How do we build an extensible, rule-based virtual assistant if no user data is available for training? This section presents the methodology of how we crowdsource training data and an overview of our machine learning algorithm.

4.1 Training Data Acquisition

Previous research to translate natural language into trigger-action code is based on the dataset obtained from the IFTTT website, which comprises over 114,000 trigger-action programs, using over 200 different devices [9, 18, 22]. Unfortunately, associated with each program is only an imprecise and incomplete English description that helps users find useful recipes, and not a specification of the code. For example, "Foursquare check-in archive" should map to "save any new check-in to Foursquare in Evernote", but this is not immediately obvious. Information on the parameters is often simply unavailable. Dong et al. [9] reported an accuracy of 40% to identify the correct function and 55% to identify the correct device on the IFTTT dataset. None were able to derive correctly the parameters for the trigger-action code, which is clearly necessary for completeness.

An extensible virtual assistant needs to provide a natural language interface for new devices, before any user data is available. Our solution is to derive a training set by crowdsourcing, with the help of the natural language entries supplied along with each Thingpedia function, as shown in Figure 6. This training set will give Almond rudimentary natural language understanding to allow it to further learn from users.

It is unreasonable to ask a Mechanical Turk worker to create a correct training sample, as it entails creating (1) a ThingTalk program, complete with parameters, and (2) its complete specification in natural language. Borrowing from the methodology proposed by Wang et al. [28], our approach is to use information in each Thingpedia entry to create a sample of ThingTalk programs and their corresponding descriptions in natural language; we then crowdsource paraphrases to arrive at more natural-sounding sentences.

First, we generate a set of random candidate programs using the ThingTalk grammar, sampling uniformly pairs of supported functions and assigning the parameters randomly. Every entry in the Thingpedia contains a confirmation string, with which the virtual assistant asks the user before performing any action. We combine the confirmation strings associated with each of the functions in the generated program to create a *synthetic sentence*. Note that such sentences are typically very clunky and hard to understand; we use a set of heuristics to make them more understandable.

Next, we ask the workers to provide three paraphrases for each synthetic sentence. Using 3 different paraphrases ensures variability in the training set and expands the vocabulary that the machine learning can observe during training. In addition, because these programs are randomly generated, they may not be meaningful. Thus, we allow workers to answer "no idea" when the sentence is hard to understand and we drop these programs from the training set.

Figure 8 shows 3 ThingTalk programs, their corresponding synthetic sentence, and a possible paraphrase. As seen from these examples, having just a parameter is already hard to understand. We sample programs in a way such that it is exponentially less likely to have many parameters, and we rely on machine learning to handle more complicated programs. Furthermore, we introduced heuristics to ensure that parameters have meaningful values. For example, the *target* parameter for @yandex.translate is given strings like "Italian" and "Chinese", even though it is of generic String type.

4.2 Machine Learning Algorithm

Once we acquired the training data, we use a semantic parser built upon the SEMPRE 2.0 framework [21], which uses the generation-without-alignment algorithm introduced by Berant et al. [5].

Like the training-set generation algorithm, our parser uses the ThingTalk grammar to generate many candidate programs. It uses the *canonical form* in the Thingpedia entries to generate a *canonical sentence* for each program. It then uses the model learned from the training to find the top 5 matches between the input and the canonical sentences, returning the corresponding ThingTalk programs as the result. Details of our algorithm are out of the scope of this paper.

5. CROWDSOURCING THINGPEDIA

Thingpedia is hosted as a web service at https://thingpedia.stanford.edu. Developers can open an account on Thingpedia and submit entries for their devices. Once an entry has been reviewed by an administrator, it is publicly available. Users can also go to

Domain	In this category	# of devices	# of functions	# of sentences
Media	Newspapers, web comics, public data feeds	13	38	100
Social Networks	Facebook, Instagram, Twitter, etc.	7	26	70
Home Automation	Light bulbs, security cameras, etc.	6	38	54
Communication	E-mail, calling, messaging	5	29	57
Data management	Cloud storage, calendar, contacts	4	19	38
Health & Fitness	Wearables, smart medical sensors	2	10	26
Miscellaneous	Weather, time, Uber, Bing, etc.	8	27	59
Total		**45**	**187**	**404**

Figure 9: Categories of devices available in Thingpedia.

Thingpedia to build commands by typing sentences and choosing the correct interpretation.

Can developers contribute to Thingpedia? We asked 60 students in a class to contribute entries to Thingpedia. These are mostly computer science graduate level students, with a few undergraduates. We provided a list of suggested devices for students to choose from but students also came up with some on their own. They wrote a total of 57 entries, of which 45 were found working. These devices span across a wide variety of domains, from media, social networks, home automation, communication, data management, health, and other miscellaneous services. Figure 9 shows the categories of devices submitted, and the number of devices, functions and sentences in each category.

The number of primitive commands supported for each device varies, ranging from 1 to 10, with an average of 4.2 commands per device. The Nest thermostat [20] provides a relatively large set of APIs from reading the temperature to making changes to different settings, whereas a scale can only measure weight. Sportradar [24], an app that provides updates for various sports, has the largest number of commands to access results of different sports and teams.

Each Thingpedia entry has 1 to 26 example sentences, with an average of 9 sentences per entry. Monitors use more sentences because they can be constructed by applying multiple filters (for example, @gmail.receive_email can filter by subject, author, label or snippet, which results in different commands) and by paraphrasing (e.g. "notify me when I receive a new email on gmail", "notify me when there is a new email in my inbox", "monitor my emails").

How hard is it to write a Thingpedia entry? Most of the work lies in finding the documentation, choosing the right APIs and mapping them to useful sentences. This process can be greatly aided by the varied expertise of different people by crowdsourcing.

For the majority of devices, the Thingpedia functions map easily to the device APIs. Most of the complexity is in the discovery and configuration code especially for physical devices that use non-standard authentication protocols. In addition, triggers are challenging because they can use different notification styles, such as polling, streaming or web hooks.

On the whole, it takes about 42 to 1198 lines of code to write a Thingpedia entry, not counting comments and metadata, with an average of 195 LOC per entry and 47 LOC per function.

6. VIRTUAL ASSISTANT EXPERIMENT

The Almond Virtual Assistant app, which we recently released in the Google Play Store, has all the functionality described in this paper. It helps users discover their local devices and lets them supply personal accounts information, which are then stored on their personal devices. Everything runs locally on the phone except for the language-to-text translation, which is provided as an anonymous web service to overcome the memory limitations on mobile devices. The Android app is built with a mix of Java and JavaScript. The entire code base is open source and is available on Github[2].

6.1 Training Almond

To train our machine learning algorithm, we use 3 data sets: a *Base* set, a *Paraphrase* set and an *Author* set. All together, 7488 sentences are collected, of which 3511 are primitive and 3977 are compound.

The *Base* set is automatically generated from the Thingpedia example sentences, by replacing each parameter placeholder with representative values (e.g., given the Thingpedia example "turn $power my tv" we generate "turn on my tv" and "turn off my tv"). The base set provides useful training for primitive commands and guarantees a basic level of extensibility to new devices. The Base set for our 187 functions consists of 2394 primitive sentences.

The *Paraphrase* set was generated by the Mechanical Turk workers, as described in Section 4.1. The goal of the paraphrase set is to provide coverage for filters and parameter combinations, as well as learning the paraphrases of compound commands. Our paraphrase set consists of 717 primitive sentences and 3749 compound sentences.

The *Author* set was a small set of realistic commands written by the authors of this paper to guide machine learning towards useful programs. This also counterbalances the uniform sampling used in the Paraphrase set. This set consists of 400 primitive sentences and 228 compound sentences.

Whenever Almond is not sure if it can understand the natural language input, it returns multiple answers for the users to pick from. Thus, to evaluate Almond, we measure if the correct answer is among the top 1, 3, and 5 matches.

To provide a baseline for comparison, we measure how well Almond can handle new unseen paraphrases. We collected an additional test set of 1874 paraphrases, of which 301 are primitive and 1573 are compound. As some of these sentences map to the same code, the whole test suite contains 993 distinct ThingTalk programs.

The accuracy results for the paraphrase test set on our trained parser is shown in Figure 10. Our parser obtains a top 1 accuracy of 71%, top 3 of 88% and top 5 of 89% for primitive sentences, and 51% at top 1, 61% at top 3 and 63% at top 5 for compound sentences. The parser performs well for primitive sentences since the test sentences

[2] https://github.com/Stanford-Mobisocial-IoT-Lab

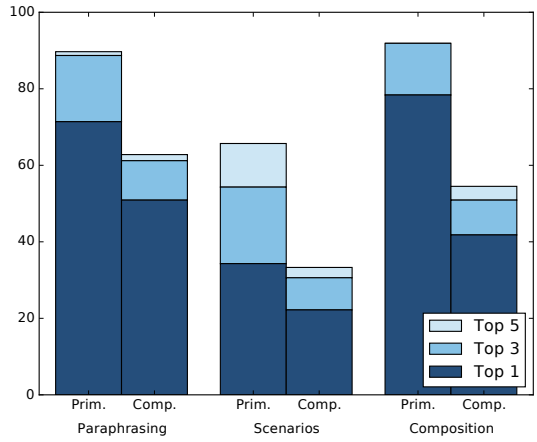


Figure 10: The accuracy of the natural language parser, on our different testing sets.

have significant lexical overlap with the training sentences. The compound commands are significantly harder to parse since there are many more possible choices for parameter compositions and values.

6.2 Scenario-Based Sentences

To gather a realistic test set, we create some real-life scenarios to inspire the Mechanical Turk workers to come up with virtual assistant commands in their own words. First, we provide workers with the context of the experiment by showing them the cheat sheet. Next, we present them with a scenario, describing one day in the life of a person, such as a pet shop owner, an office worker, a dog walker, or restaurant manager. We ask them to envision that they are the person in the scenario and to come up with a command that would make their life easier. These scenario descriptions are written in a way that does not bias the Mechanical Turk workers towards words used in the cheat sheet.

Using this scenario method, we collected sentences that are more natural but open-ended since the workers were not given any sentences to paraphrase. We found that among the 327 sentences obtained, 131 sentences are either unintelligible or irrelevant. Furthermore, 109 sentences refer to operations outside the scope of the device APIs available. Out-of-scope sentences will remain a problem for general-purpose virtual assistants; we believe users will eventually learn the capabilities of the virtual assistant through trial and error. 16 of the sentences have the wrong format, such as having strings that are not quoted; it may be possible in the future to relax this constraint. In the end, we are left with 71 meaningful sentences, 35 primitive and 36 compound. We manually translate these 71 sentences into their corresponding ThingTalk programs. This illustrates how hard it is to gather realistic training data for Almond.

The parser obtained a top 1 accuracy of 34%, top 3 of 54% and top 5 of 65% on primitive sentences. On compound sentences, it achieves a top 1 accuracy of 22%, top 3 of 31% and top 5 of 33%. The accuracy drops significantly compared to the baseline accuracy obtained by testing with paraphrases. This illustrates why it is inadequate to test the parser on paraphrases and the need to obtain more organic data for training. Our approach of providing a menu-driven interface along with the natural language interface provides a means to obtain more training data as time goes on.

6.3 Capable Users

In practice, we expect users to rely on menu-driven interface until they learn what the virtual assistant can do and adapt to its natural language capabilities. To approximate the experience of a user familiar with the virtual assistant, in our next experiment we first ask the workers to read through the cheat sheet. We then remove the cheat sheet, and ask them to pick two functions from their memory and combine them in their own words to form a compound command. We ask each worker to come up with five such sentences that use different functions, assuming that he will remember less of the exact wordings in the cheat sheet as he constructs more sentences. Workers are not allowed to do this task more than once.

In total, we collected 200 sentences, which we then annotated with the corresponding programs manually whenever possible. Of these 200 sentences, we found that 108 were out of scope or unsupported. For the remaining 92 sentences, mapping to 78 distinct programs, the parser obtained an accuracy of 80% at top 1, 94% at top 3 and top 5 for primitives, and 41% at top 1, 50% at top 3 and 53% at top 5 for compound commands. The results are also shown as the *composition* experiment in Figure 10.

Although our parser is not accurate enough to satisfy an end user today, it is nonetheless the first that can translate any compound commands with complex parameters into executable code. The parser achieves a 40% accuracy despite having no real user data; we are hopeful that better results can be obtained as users experiment with our system and generate more real data.

6.4 Language vs. Menu-Driven Interface

To understand if users prefer the natural language or the menu-driven interface, we conducted a user study with 7 female and 6 male computer science students. Users in this study are first given a description of how Almond works and the WHEN, GET, DO architecture. They are given 3 to 5 minutes to review the menu of possible functions as well as a list of example commands. They are then shown a list of fourteen scenarios (similar to the ones explained in Section 6.2) and asked to choose three to come up with a useful command for each. They are allowed to try a few different sentences to get Almond to understand.

Almond translated 40% of the user input, including parameters, into correct ThingTalk programs. 24% of the inputs require capabilities outside the scope of Almond; 9% the inputs cannot be understood because of missing quotation marks, and for the remaining 27%, Almond got the function right but did not identify the right parameters. Notwithstanding that it is frustrating for users when they try to issue unsupported commands, we observe that Almond achieves a 60% accuracy for sentences that can be understood.

Users find it easier to type what they want in English instead of having to scroll through the menu to find the device of interest. However, even though Almond asks the user to confirm an action before it is taken, the confirmation sentences are sometimes confusing and the user does not know for sure if their natural language commands are fully under-

stood. Thus, they prefer to use natural language for "low risk" situations such as setting reminders and notifications. They prefer to use the menu-driven interface for "high risk" commands, such as posting to social media or programming their home security camera or locks.

7. RELATED WORK

Virtual Assistants. Virtual assistants have been developed for education [12], entertainment [11], and elder care [16]. Each of these is domain specific though: a user has to interact with a different assistant for each request. Amazon's Alexa is the only system that can interact with third-party services through "skills" provided by the third-parties themselves. The other commercial systems only respond to a fixed number of queries defined by the company that makes them.

In Alexa, commands are limited to the form "Tell/Ask *service name* to ...", with semantic parsing only for the "..." part. The advantage of the fixed structure is that natural language parsing only needs to recognize the intent among a small set of service capabilities. The disadvantage is that each command can involve only one service at a time and there is no programmability.

IoT platforms. Dixon et al. [8] introduces HomeOS, "an operating system for the home". The goal of the project, which has since been abandoned, was to build a collection of interfaces to home automation smart devices. Unlike Thingpedia, HomeOS was not open source and did not allow open contributions, limiting itself to the use case of research in home automation.

Mayer et al. [19] make the interesting observation that a rule or process based system is necessarily static, while a home automation system should be dynamic. They propose a goal-oriented system based on an RDF model of the different devices and an RDF solver to derive the right connections.

On the commercial side, several companies have attempted to build their own IoT stack, including Samsung SmartThings [23], Google Weave [10] and Apple HomeKit [2]. These systems are vertically designed, together with the virtual assistant and cloud stack, are closed and not interoperable with each other.

Natural language parsing. The body of previous work in semantic parsing is abundant [1, 7, 15, 30, 31]. Berant et al. [4] introduce the SEMPRE framework as a question answering system over the Freebase [6] database, and extend it by proposing the generation without alignment algorithm [5]. Wang et al. [28] added the ability to build a "semantic parser overnight", and allowed extensions of SEMPRE to a new domain with minimal work.

Trigger-Action programming. The first notable attempt to build a trigger-action programming system is CAMP [25]. They include a limited "natural language" system to describe the rules based on small sentence pieces that are glued together in a visual way like fridge magnets.

More recently, IFTTT [14] is a website that lets the user connect services in a pair-wise fashion, using a menu-driven user interface. Ur et al. [26] did user testing by extending IFTTT with filters, and found the trigger action metaphor to be familiar to the users. Huang et al. [13] corroborates their findings with an analysis of the pitfalls of trigger-action programming, and investigate what triggers are understandable by humans.

Walch et al. [27] make the argument that while rules are easy to understand, they are not appropriate in the home automation domain because conditions become too complex. They propose a process based system, and then use a graphical user interface to configure the processes, but their user testing does not show convincing results.

8. CONCLUSION

Virtual assistants are likely to revolutionize how we interact with machines. Thus, major companies are vying to become the dominant virtual assistant that sees all users' data and intermediates all services. Almond is an attempt to rally makers, developers, and users to contribute to creating the world's first open and most general, interoperable and powerful virtual assistant that encourages open competition and preserves user privacy.

This paper presents a fully functional prototype of Almond. The key concepts presented include the open Thingpedia knowledge base, the open-source ThingSystem for managing user data, the high-level ThingTalk language for connecting devices, and finally a machine-learning based translator that can convert natural language commands into trigger-action code.

Almond, available on the Android play store, is the first virtual assistant that can convert rules with parameters expressed in natural language into code. It has some basic natural language understanding capability that is extensible to new devices added to Thingpedia. Based on just a few sentences in the entry, Almond acquires training data for such devices by generating trigger-action programs that use these devices and crowdsources the corresponding paraphrases.

With a combination of a language and menu-driven interface, Almond is ready to be used by enthusiasts to automate nontrivial tasks. Almond's open and learning infrastructure will hopefully attract enough contributions for it to grow to serve a more general audience over time.

9. ACKNOWLEDGMENTS

The authors thank students who contributed the Thingpedia devices, and especially Yushi Wang, Sida Wang, Panupong Pasupat and Prof. Percy Liang for their help with SEMPRE. Support for this work was provided in part by the Stanford MobiSocial Laboratory, sponsored by HTC and Samsung, and a Siebel Scholar Fellowship for Giovanni Campagna.

10. REFERENCES

[1] J. Andreas, A. Vlachos, and S. Clark. Semantic parsing as machine translation. In *ACL (2)*, pages 47–52, 2013.

[2] Apple HomeKit. http://www.apple.com/ios/home.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

[4] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, volume 2, page 6, 2013.

[5] J. Berant and P. Liang. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual*

Meeting of the Association for Computational Linguistics (ACL-14), pages 1415–1425, 2014.

[6] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

[7] D. L. Chen and R. J. Mooney. Learning to interpret natural language navigation instructions from observations. In *AAAI*, volume 2, pages 1–2, 2011.

[8] C. Dixon, R. Mahajan, S. Agarwal, A. Brush, B. Lee, S. Saroiu, and P. Bahl. An operating system for the home. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 337–352, 2012.

[9] L. Dong and M. Lapata. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL-16)*, pages 33–34, 2016.

[10] Google Weave. https://developers.google.com/weave.

[11] M. Gordon and C. Breazeal. Designing a virtual assistant for in-car child entertainment. In *Proceedings of the 14th International Conference on Interaction Design and Children*, pages 359–362. ACM, 2015.

[12] P. H. Harvey, E. Currie, P. Daryanani, and J. C. Augusto. Enhancing student support with a virtual assistant. In *International Conference on E-Learning, E-Education, and Online Training*, pages 101–109. Springer, 2015.

[13] J. Huang and M. Cakmak. Supporting mental model accuracy in trigger-action programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 215–225. ACM, 2015.

[14] If This Then That. http://ifttt.com.

[15] R. J. Kate, Y. W. Wong, and R. J. Mooney. Learning to transform natural to formal languages. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 1062, 2005.

[16] P. Kenny, T. Parsons, J. Gratch, and A. Rizzo. Virtual humans for assisted health care. In *Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments*, page 6. ACM, 2008.

[17] W. J. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and D. Haussler. The human genome browser at UCSC. *Genome research*, 12(6):996–1006, 2002.

[18] C. Liu, X. Chen, E. C. Shin, M. Chen, and D. Song. Latent attention for if-then program synthesis. In *Advances in Neural Information Processing Systems*, pages 4574–4582, 2016.

[19] S. Mayer, N. Inhelder, R. Verborgh, R. Van de Walle, and F. Mattern. Configuration of smart environments made simple: Combining visual modeling with semantic metadata and reasoning. In *Internet of Things (IOT), 2014 International Conference on the*, pages 61–66. IEEE, 2014.

[20] Nest. https://developer.nest.com.

[21] P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53nd Annual Meeting of the Association for Computational Linguistics (ACL-15)*, pages 1470–1480, 2015.

[22] C. Quirk, R. Mooney, and M. Galley. Language to code: Learning semantic parsers for if-this-then-that recipes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL-15)*, pages 878–888, 2015.

[23] Samsung SmartThings. http://www.smartthings.com.

[24] Sportradar. http://sportradar.us.

[25] K. N. Truong, E. M. Huang, and G. D. Abowd. Camp: A magnetic poetry interface for end-user programming of capture applications for the home. In *International Conference on Ubiquitous Computing*, pages 143–160. Springer, 2004.

[26] B. Ur, E. McManus, M. Pak Yong Ho, and M. L. Littman. Practical trigger-action programming in the smart home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 803–812. ACM, 2014.

[27] M. Walch, M. Rietzler, J. Greim, F. Schaub, B. Wiedersheim, and M. Weber. homeblox: making home automation usable. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, pages 295–298. ACM, 2013.

[28] Y. Wang, J. Berant, and P. Liang. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL-15)*, pages 1332–1342, 2015.

[29] E. Weisstein et al. Wolfram mathworld, 2007.

[30] C. Xiao, M. Dymetman, and C. Gardent. Sequence-based structured prediction for semantic parsing. *Proceedings Association For Computational Linguistics, Berlin*, pages 1341–1350, 2016.

[31] L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 658–666. AUAI Press, 2005.