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

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

ABSTRACT

This paper presents the architecture of Sabrina, an open, crowdsourced, privacy-preserving and programmable virtual assistant for online services and the Internet of Things (IoT). Sabrina is built on top of Thingpedia, a crowdsourced public knowledge base of natural language interfaces and open APIs. 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 address usability by providing a natural language interface, whose capability can be extended via training. We demonstrated the feasibility of a crowdsourced assistant with a fully functional prototype. With natural language users can specify combinations of actions drawn from a crowdsourced set of 187 functions across 45 different kinds of devices.

1. INTRODUCTION

A virtual assistant 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 read, let alone 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 plane 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, such as privacy, interoperability, and generality.

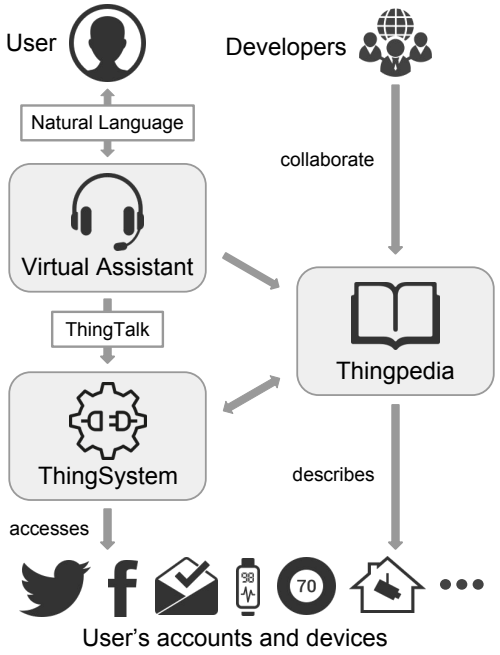


Figure 1: The architecture of Sabrina.

To answer these questions, we propose the architecture for *Sabrina*, shown in Figure 1, an open, crowdsourced, privacy-preserving and programmable virtual assistant. With natural language users can ask Sabrina 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 succinctly 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, Sabrina is the only programmable virtual assistant that lets users specify event-driven commands in natural language. For example, the user can instruct Sabrina to call 911 if motion is detected by the security camera,

WWW 2017 April 3–7, 2017, Perth, Australia

DOI:

whenever the alarm is turned on. Sabrina is programmable: she can translate this sentence into ThingTalk code, as long as the actions "call", "detect motion on the security camera", and "alarm is on" are available in Thingpedia. The closest available system is If This Then That (IFTTT) [15] 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 accounts and personal information. Furthermore, they need to be given permission to interact on their behalf and provide suggestions that are tailored to their personal preferences. However, data like bank accounts, medical records, legal information or security cameras 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 is open for code reviews and checks for malicious behaviors.

Interoperability. Keeping the system open also improves interoperability. Having multiple non-interoperable proprietary standards, such as Google Weave [11] 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, to a LG TV or a Samsung TV, and the different details are handled transparently by the system.

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 Sabrina open-source as an attempt to bring researchers and developers together to create an open system competitive to proprietary undertakings. 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 [18]), etc. By supporting device inter-operability and making the entire virtual assistant code base available, we hope that Thingpedia can attract all service providers, device makers, and hobbyists to contribute their interfaces to help create the largest encyclopedia of things. Having all this knowledge in the public domain can promote open competition and thus innovation.

Usability. While Sabrina can translate natural language into ThingTalk code, its accuracy today is severely limited due to lack of training data. We have created a user interface that lets users train Sabrina with relative ease. Over time, the crowdsourced training information can feed into further machine-learning research to produce the natural language interface of a powerful virtual assistant.

1.2 Contributions

Architecture of an open virtual assistant. Sabrina 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.

A crowdsourced language-to-code platform. We have developed a semantic parser, based on the SEMPRE [28] framework, that translates natural language into ThingTalk code. This platform integrates with the Thingpedia repository so that Sabrina is extended dynamically with each new Thingpedia entry. Our parser is trained on data crowdsourced from Thingpedia contributors and Sabrina users.

Demonstrated feasibility of a crowdsourced virtual assistant with a fully functional prototype of Sabrina. 60 students successfully contributed to the system, resulting in a Thingpedia with 45 devices, with about 200 lines of code each on average, and 705 commands expressed in 1261 sentences. Our semantic parser is found to have an accuracy of 50%. Thanks to our interface that also provides users with choices and templates, our beta users have used our system effectively in a wide range of scenarios.

1.3 Organization

In Section 2, we describe what the virtual assistant Sabrina can do and her user interface. We describe an overview of the underlying system architecture in Section 3 and our algorithm to translate natural language to ThingTalk in Section 4. We then describe the experimentation of Sabrina in Section 5. Finally, we present related work and conclude.

2. SABRINA VIRTUAL ASSISTANT

This section describes what Sabrina can do and how users interact with Sabrina. Sabrina is unique in that she can perform event-driven tasks based on natural language commands. Furthermore, she is fully extensible: her capabilities grow as Thingpedia grows.

2.1 Generality

Sabrina derives its generality from Thingpedia, a repository designed to capture interfaces to all the different web services and IoT devices. While Sabrina can access publicly available services like the weather channel or Uber, her strength is in managing and interfacing with personal web accounts and devices with privacy[1]. For example, Sabrina can notify a user of low bank balances without having the user disclose his credentials to a third party. Sabrina prompts the user for the credential just once, when a user

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

initiates the action, and stores the credential on, for example, the user's phone. Sabrina can also handle physical devices, irrespective of whether they use Bluetooth, WiFi, etc. The user tells Sabrina to initiate discovery of devices, and Sabrina prompts the user for the credentials to each device found.

2.2 Expressiveness

At the basic level, Sabrina lets users access a wide collection of devices with a uniform textual interface. Users can simply tell Sabrina to "tweet a message" or "turn on the light", thus avoiding the need to navigate different interfaces in different apps. Our Sabrina virtual assistant is unique in that she can perform event-driven operations. She can also be "programmed" to connect functions from different devices together.

Class	Type	Examples
primitive	action	send email to bob
	query	get my latest email
	monitor	notify me if I receive an email
	filtered monitor/query	notify me if I receive an email where 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 Sabrina

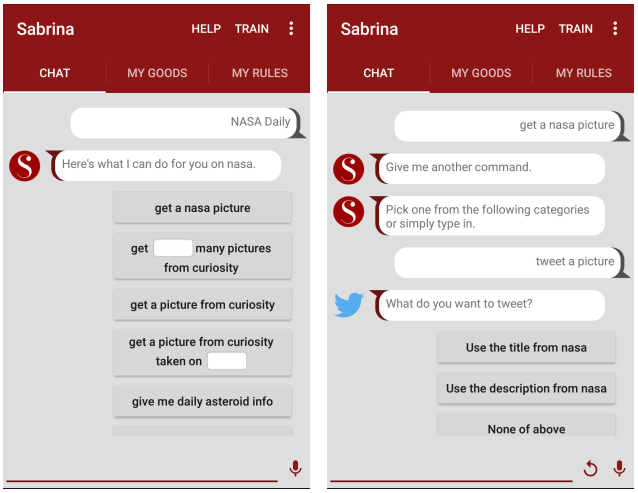
We categorize the commands that Sabrina accepts into *primitive* and *compound* operations. The most basic primitive command is a direct *action* or *query* to a device. Sabrina also supports standing queries or *monitors* to notify the user when an event of interest is triggered. Users can add one or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

2.3 User Interface

Sabrina is a conversational agent with a chat-based interface. Like a real assistant, the user and the virtual assistant needs to spend a period of time to get acquainted. Here, we describe the Sabrina's interface by way of three personas.

Stage 1: Discover. Cody, a novice user, starts by exploring the application by clicking the *help* button, whereupon Sabrina lists the classes of devices supported. Cody picks the device, and Sabrina shows the list of possible commands for that device, ranked by popularity (Figure 3a). Cody picks the command, fills in the blanks, and issues the command. If he leaves any blanks empty, Sabrina will



(a) List of commands for NASA (b) Interactive creation of compound commands

Figure 3: Two screenshots of the Sabrina user interface

prompt him for his answers.

Stage 2: Talk & Train. Familiar with the supported devices and commands, Bob can talk to Sabrina in voice or text instead of clicking buttons. Eventually, Sabrina will have no problem understanding Bob. Before then, Bob has the ability to teach Sabrina when she misunderstands him. By clicking on the *train* button, he can access a list of possible commands that might match his sentence. If Bob still can't find the correct match, he will be asked to rephrase the sentence.

Stage 3: Create. Alice has configured all her devices on Sabrina, and she wants to connect her devices based on her needs. Sabrina has the basic capability of understanding compound commands, even if it is a combination that she has never encountered before. However, at this point, Sabrina is not so accurate with compound commands. For training purposes, Sabrina also provides an alternative interactive method. Sabrina guides Alice through picking the primitives involved, and Sabrina asks Alice follow-up questions, with possible answers, on how the primitives are to be combined (Figure 3b).

3. SYSTEM ARCHITECTURE

In this section, we provide an overview of Sabrina'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 4. Each entry includes information such as a version number,

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, action, queries	set_power(*power*), play_url(*url*), set_volume(*percent*), mute(), unmute()

Figure 4: 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	"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 Sabrina 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)

Figure 5: The natural language entry for a Thingpedia function, with the example of set_power on the LG Smart TV.

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 5, 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:

```
@instagram.new_picture(picture_url, caption, hashtags),
    $contains(hashtags, "cat")
⇒ @facebook.post_picture(text, url),
    text = caption, url = picture_url
```

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:

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

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 query or trigger parameters to a variable name and using them to set the action parameters.

Primitive Sabrina 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 6 summarizes the correspondence between Sabrina 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, Sabrina 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.

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

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 6: The different ThingTalk forms, and how they map to Sabrina commands.

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: 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 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

In this section we present how the natural language commands are converted to ThingTalk code.

4.1 Prior Work

Quirk, et al. [22] were the first to analyze natural language descriptions of trigger-action code. Their goal is to identify the device and function names in the natural language descriptions of IFTTT recipes, each of which contains one trigger and one action. While their training set has 114408 recipes, the descriptions are very imprecise because they are intended to help users find useful recipes.

The most recent efforts [4, 10] use a neural network model to train the system, and obtain an accuracy of 42% for correct function, and 54% for correct device. When they focus on a small subset of 758 high-quality recipe descriptions (obtained by hand using Amazon Mechanical Turk), they obtain an accuracy of 82% to identify the correct function.

4.2 Our Solution

Our problem is harder than previously attempted because we need to generate fully executable code, which requires correct identification of the parameters, and not just the device function names. First, to get better training data, we require contributors of Thingpedia to include a few example sentences to describe each function. Second, we have created a semantic parser that translates natural language sentences into ThingTalk codes as a logical form. Third, our machine-learning based parser is integrated with Thingpedia so it can dynamically improve with the addition of more devices and more sentences from users.

Our parser builds upon the SEMPRE 2.0 framework [28], which uses the generation without alignment algorithm introduced by Berant et al. [6]. This parser uses a formal grammar, based on the ThingTalk grammar in our case, to generate many candidate programs, and then chooses the best one using machine learning. Details of our algorithm are out of the scope of this paper.

A demonstration of the technique is detailed below for better understanding. For example, the natural language command "Play the 'presidential debate' from YouTube on my TV" is parsed as follows:

1. Run a syntactic parser and entity extractor on the input. This returns "presidential debate" as a string parameter.
2. Look up words in the sentence to identify the relevant functions from the *function lexicon* in Thingpedia. The lexicon is built from words in the canonical form of the function after removing stop words like "from" and "on". For this example, the words "play", "youtube", "lg" and "tv" return the following among others. (The first two are the desired functions).

   @youtube.search_videos
   @lg_webos_tv.play_url
   @youtube.search_channel
   @tv.set_power

3. Combine each function with each value exhaustively. In this case, there is only one value, "presidential debate", which generates several candidates, among which:

   @youtube.search_videos(*query*, *video_url*, *title*),
   query = "presidential debate" (1)

   @lg_webos_tv.play_url(*url*),
   url = "presidential debate" (2)

4. Triggers and actions are combined with all choices of parameters to generate the following among other pairwise possibilities:

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 7: Categories of devices available in Sabrina.

$$\$\text{now} \Rightarrow \text{@youtube.search\_videos}(\mathit{query}, \mathit{video\_url}, \mathit{title}), \mathit{query} = \text{“presidential debate”} \Rightarrow \text{@lg\_webos\_tv.play\_url}(\mathit{url}), \mathit{url} = \mathit{video\_url} \quad (3)$$

$$\$\text{now} \Rightarrow \text{@lg\_webos\_tv.play\_url}(\mathit{url}), \mathit{url} = \text{“presidential debate”} \quad (4)$$

5. Featurize and score to choose the program most closely matching the input using logistic regression. In this case, the ThingTalk code in (3) is deemed to have the highest score and is chosen.

5. EXPERIMENTATION

The Sabrina Virtual Assistant app, 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 Java and JavaScript.

To make Sabrina more generally accessible, she is also available as a chat agent on Omlet Chat [21], an open commercial chat app that runs on Android and iPhone. We chose Omlet Chat because it honors privacy and also provides the option to back up data on the cloud.

The Sabrina chat agent, provides similar functionalities as the Android version, except that it cannot discover and configure local devices. Users' data is hosted at https://sabrina.stanford.edu, a service that does not own or monetize it. The entire code base is open source and is available on Github[2].

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

In the following, we assess Sabrina's capability to (1) support crowdsourcing and (2) help with different tasks as a virtual assistant.

5.1 Crowdsourcing

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 7 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 if I receive a new email on gmail”, “notify me if 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.

Can users contribute to the English sentences of Thingpedia? Besides the canonical sentences supplied by the device contributor, users can also write English sentences that translate into new ThingTalk programs. At this point, users of Sabrina have submitted 1261 sentences, mapping to 705 unique ThingTalk programs. These sentences correspond to commands of varying complexity, of which 657 are primitive and 333 are compound. Besides these two

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

Domain	# of sentences	# of programs
Media	304	181
Social Networks	291	182
Home Automation	254	187
Communication	223	144
Data management	55	47
Health & Fitness	8	7
Miscellaneous	203	131

Figure 8: Crowdsourced sentences from the users, classified by domain. (Compound commands are multiple counted)

classes, the other 253 commands are referred to as *bookkeeping*, which handles configuration, discovery, confirmations and follow-up answers. The breakdown of these commands by domain is shown in Figure 8.

This shows that even with a small number of users, people are interested in a wide collection of devices, supporting our motivation to create a general virtual assistant platform. Quite a few compound sentences have been created; we found that while it is harder to create compound commands, it is also more powerful and more attractive to users.

5.2 Natural Language Understanding

Sabrina today does not have enough data to be proficient at natural language. Sabrina's machine-learning algorithm is nonetheless in place to learn as she gathers usage. Here we evaluate the accuracy of Sabrina's semantic parser based on the small training data set we have today.

We separate a part of the dataset to initially train the parser and use the rest as test set to verify the trained parser. Our dataset contains 813 training sentences and 448 test sentences. The test sentences can be categorized as 101 compound sentences, 261 primitive sentences and 86 bookkeeping sentences. The 448 sentences correspond to 293 distinct ThingTalk programs.

We extend our small dataset by generating sentences using the example commands from Thingpedia by providing synthetic values for the unfilled parameters. This technique adds a further 5235 primitive sentences to our dataset that are used just for training.

After training the parser, we use our test set to verify three levels of correctness: correct device, correct function, full match. Correct device checks for the structure of the command and the device match while the correct function additionally checks for the function match. Full match is the gold standard for our parser which tests for correct parameters along with the correct device and function. The first two levels are still useful measures since the parameter extraction can be solved by using follow up questions.

The accuracy results are shown in Figure 9. Overall, our parser achieves 50% accuracy for a full match while we can identify the correct device 77% and the correct function 60% of the time. We demonstrate a better accuracy for the primitive commands (64%), going as high as 78% for the correct function and 90% for the correct device. However, our algorithm performs poorly on compound commands (Full: 14%, Function: 27%, Device: 45%) because the training data is insufficient to accurately cover the space today.

Despite the low accuracy of the learning algorithm, there are several scenarios where the semantic parser is useful today. First, as discussed in Section 2.3, our users typically

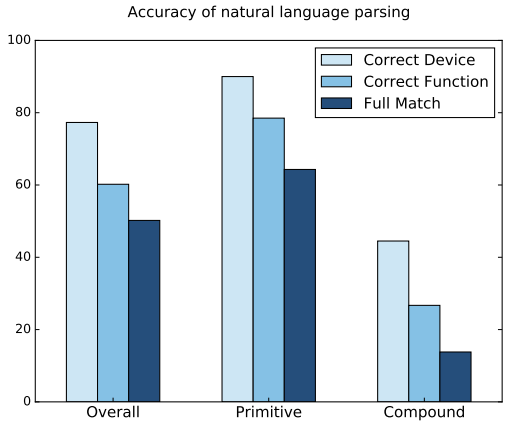


Figure 9: The accuracy of the Sabrina natural language system on our testing set.

start by perusing sentences that Sabrina accepts through the "help" button. The semantic parser understands minor variations of such sentences reasonably well. Second, a user often reuses the same commands. Once a user trains the parser using the interactive interface, subsequent issues of similar commands are likely to succeed. Finally, the coverage of our training set is uneven; commands around the more commonly used devices, such as Twitter and security cameras, have a higher chance of success. If Thingpedia can attract the participation of device makers, they have the incentives to supply more training sentences for their devices. We believe the accuracy of the parser will improve as more training data becomes available.

5.3 Use Case Scenarios

Our Sabrina virtual assistant prototype today can support a large number of use cases. To give the readers a sense of its capability, we describe four use case scenarios assembled from the various rules our beta users have created using Sabrina.

Simple, universal interface. Dan likes Sabrina, because she provides a uniform interface across many devices, without requiring him to share data with a third party.

Here are 7 ways in which Sabrina helps Dan in just the first couple of hours of his day. When he wakes up, he tells Sabrina to "Turn on the lights" and asks "What is the weather like?". While getting ready for work, he asks "Give me an Uber estimate to work". If the price is too high, he will drive. While driving, he uses voice control on Sabrina to text his mom, by saying "Send a text" and answering Sabrina's follow-up questions to compose the text. He arrives at work and says "List events on my calendar". He sees he has to give a presentation, he tells Sabrina to "Show my video presentation from YouTube on the TV". His phone is automatically turned to silent by the time the meeting starts because he asks Sabrina to "Turn my phone to silent every work day at 9am".

Quantified self. Ethan uses Sabrina to monitor his habits, knowing full well that he can switch devices while maintaining a common interface and losing no data.

Ethan likes to keep track of how much he sleeps, eats, and exercises. He uses Sabrina as an interface with his Jawbone UP to find "How much did I sleep last night?", "How many

steps have I taken today?", and "How many calories did I burn today?". By using Sabrina, Ethan can also switch to a Fitbit whenever he wants without having to learn a new interface.

Media Filtering. Fei is a software developer who loves to consume media. She saves time by using Sabrina to stream all the social media content customized to her preferences, without having to open each app many times a day to check for new content.

She links Sabrina to her accounts on Facebook, Instagram, Twitter, Tumblr, and YouTube. She then tells Sabrina to "Monitor @justinbieber on Twitter" and "Send me new XKCD comics". She asks Sabrina to "Post my new pictures from Instagram to Twitter", so she does not have to do it manually. She prefers to see the news headlines in Chinese, so she says "Get new articles from the Washington Post and translate them to Chinese". Sabrina is useful for work too, because Sabrina can "Monitor pushes to Github" and "Monitor Slack updates on channel #urgent".

Home Automation. As a non-technical person, Guy likes to use Sabrina to connect his gadgets easily.

His teenage son has the tendency of turning on the air conditioner even if the weather is cool. To save energy, he asks Sabrina to "Notify me if the temperature is below 70 F and the cooling is on in my thermostat." His wife Hellen loves hummingbirds: he sets up a hummingbird feeder and points a camera at it, and he asks Sabrina to "Send the video to @hellen via the Omlet chat if there is motion on my camera".

Summary. Via a simple and uniform chat-based interface, Sabrina 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 Sabrina's repertoire, an average programmer can extend Sabrina by adding the interfaces of interest into Thingpedia with a reasonable effort.

6. RELATED WORK

Virtual Assistants. Virtual assistants have been developed for education [13], entertainment [12], and elder care [17]. 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. [9] 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 [11] 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 [16, 8, 31, 1, 30]. Berant et al. [5] introduce the SEMPRE framework as a question answering system over the Freebase [7] database, and extend it by proposing the generation without alignment algorithm [6]. 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 [15] 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. [14] 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.

7. 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. Sabrina is an attempt to rally makers, developers, and users to contribute to create the world's first open and most general, inter-operable and powerful virtual assistant that encourages open competition and preserves user privacy.

This paper presents a fully functional prototype of Sabrina. 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 event-driven code. Sabrina is effective today in helping enthusiasts automate their tasks and get a convenient chat interface, while preserving privacy.

We demonstrated that Sabrina supports crowdsourcing and has acquired a rudimentary knowledge base of device interfaces and a small training set of commands in natural language. Sabrina's open and learning infrastructure will hopefully attract enough contributions for her to grow to serve a more general audience over time.

8. 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] I. Beltagy and C. Quirk. Improved semantic parsers for if-then statements. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL-16)*, pages 726–736, 2016.

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

[6] 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.

[7] 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.

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

[9] 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.

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

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[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] Omlet Chat. http://omlet.me.

[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)*, 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*, 2016.

[31] L. S. Zettlemoyer and M. Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*, 2012.