Skill for Amazon Alexa using Eventbrite API and Schema.org vocabulary

Samuel Frontull, Clemens Degasper, and Kamal Zakieldin

University of Innsbruck, Department of Computer Science
Semantic Technology Institute

Abstract. This paper describes the implementation of an Alexa Skill to provide a chatbot that uses the Eventbrite API and Schema.org vocabulary.

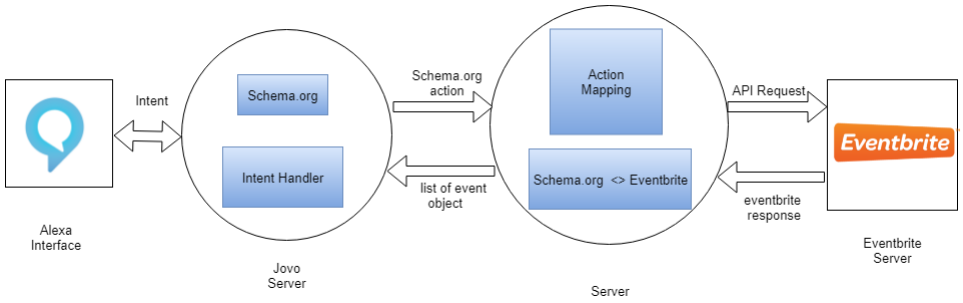
Keywords: Amazon Alexa · Eventbrite API · Schema.org.

1 Introduction

This paper is a documentation of the Alexa Skill implemented as project in the Online Communication Seminar of the STI Innsbruck research group in 2018. It describes the parts needed when implementing a Skill for Amazon Alexa that consumes data provided through an API using Schema.org for communication for developing a chatbot for Eventbrite. Eventbrite is a website for events where you can easily find and search for events around you in specific location and date, you can also create your own event and share it among others.

2 Architecture

Our design architecture consists of building 2 servers, the first one is under Jovo framework containing the intents handler of our Alexa skills, and the second server contains our main server that understands Schema.org and Eventbrite APIs.



3 Amazon Alexa

The Amazon Alexa is a virtual assistant developed by Amazon (Amazon Alexa) which is able to interact with users using voice, giving an intelligent framework to develop voiced-based chatbots. The Voice recognition algorithm is provided by Amazon and the Alexa Developer Team. For this project we had to create a Skill, which would help the user to create / view / alter / delete Events via an API. Creating and using an Amazon Alexa Skill is free and only requires and Amazon account. The online Amazon Developer Console contains also an Alexa Emulator where one can test the Skill during implementation, without the need for an physical Alexa Device.

3.1 Skills

A Skill is a way to teach Alexa to understand specific tasks in a smart way, the Skill should be designed to fulfill users' requests to perform their needs. A Skill consits of Intents, small phrases which Alexa recognizes and then executes given tasks and slots, which act like placeholders in Intent-phrases. Alexa can be extended with any desired Skills.

3.2 Intents

An Intent represents an action that fulfills a user's spoken request. Intents can optionally have arguments called slots (Amazon Developer). Eventbrite API contains 4 actions that we can perform (add event, modify event, search for event and delete event), therefore we have implemented the following intents to assist to perform smoothly these 4 actions.

3.3 Slots

Amazon has implemented a number of smart slots, which are not just a list of words to look up. The slots AMAZON.DATE or AMAZON.TIME for example understand the date and time multiple different ways it is spoken and translate it to one normed form.

Also AMAZON.SearchQuery is special, in that it doesn't do any substitutions in the phrase, it just recognizes all words the users said. This is useful for Google search queries, for example. The only restriction the AMAZON.SearchQuery has is, we cant use any other slots when we already use a AMAZON.SearchQuery.

We use the following listed slots in our Model.

{name}: slot for event name of type AMAZON.SearchQuery

{city}: slot for event city of type AMAZON.AT_CITY

{street}: slot for event street of type AMAZON.StreetAddress

{date}: slot for event date of type AMAZON.DATE

{price}: slot for event price of type AMAZON.NUMBER

{description}: slot for event description of type AMAZON.SearchQuery

{tags}: slot for event tags of type AMAZON.SearchQuery

{id}: slot for event id of type AMAZON.NUMBER

{time}: slot for event time of type AMAZON.TIME

CreateEventIntent To further illustrated the working of Intents and Slots, here the Create Event Intent:

"The Event Date is the {date} and the Event Location is {location}" {date} and {location} are the placeholder slots in this Intent. When the Amazon Alexa recognizes that the user is saying this phrase, it then inserts the correct data at the placeholders. The {date} is of type AMAZON.DATE which understand phrases like "Today", "Next Week" or "Friday the 13th, July 2018". The Slot {location} is of type AMAZON.AT_City, which is just a list of all cities in Austria and Alexa picks the one the user used.

4 Eventbrite

We choose to use the Eventbrite for this project. Eventbrite is the world's largest event technology platform (Eventbrite) and has a very good documentation. Moreover the authentication process for protected operations is simplified which makes the API easy to use.

4.1 Eventbrite API

The Eventbrite API is the best way to integrate and extend Eventbrite for your event or organizing needs (Eventbrite API). Eventbrite has a designed API to communicate and connect your applications with them as a third party using REST-APIs. It uses OAuth2 for authentication and returns responses as JSON. The Eventbrite API comes complete with the needed endpoints and required actions, and it has many different endpoints, from querying ticket types to placing orders and getting recent ticket transfers. In the following we will describe the endpoints we used.

GET /events/search/ Retrieve public event objects from Eventbrite, regardless of which user owns the event. Result can be restricted by specifying date and location.

POST /events/:id/ Updates the event with given id.

DELETE /events/:id/ Deletes the event with given id if the user is allowed to do that.

POST /events/ Makes a new event with all given attributes.

GET /users/me/owned_events/ Returns all events the user owns.

5 Schema.org Vocabulary

We use the Schema.org vocabulary for the communication between Alexa and our Server. Schema.org is a collaborative, community activity with a mission to create, maintain, and promote schemas for structured data on the Internet, on web pages, in email messages and beyond (Schema.org Website). Using Schema.org vocabulary has the advantage that it is not bounded to a specific data structure. In our case for example Alexa doesn't know that the Server communicates with the Eventbrite API because it send and gets back the result in Schema.org format. This makes our system loosely coupled. The Schema.org vocabulary defines also an Event Class, Add-, Update-, Modify- and Delete-Actions.

5.1 Event

We user `Schema.org/Event` to represent an event in Schema.org. It contains `Schema.org/Location` and `Schema.org/PostalAddress` to describe the location of the event. Listing 1.1 shows an example how we represent an event with `Schema.org/Event`.

Listing 1.1. Schema.org/Event Object

```
{
  "@context": "http://schema.org",
  "@type": "Event",
  "location": {
    "@type": "Place",
    "address": {
      "@type": "PostalAddress",
      "addressLocality": "Innsbruck",
      "addressRegion": "Tirol",
      "postalCode": "6020",
      "streetAddress": "Technikerstrasse 21a"
    },
    "name": "ICT Building"
  },
  "name": "OC Seminar - Presentations",
```

```
  "startDate": "2018-06-20T13:15:00Z",
  "endDate": "2018-06-20T15:00:00Z"
}
```

5.2 AddAction

We use `Schema.org/AddAction` when we want to create a new event. This action contains a `Schema.org/Event` as object (as shown in Listing 1.1). Listing 1.2 shows an Example of a `Schema.org/AddAction`.

Listing 1.2. Schema.org/AddAction

```
{
        "@context": "http://schema.org",
        "@type": "AddAction",
        "object": {
                "@type": "Event",
                ...
        },
        "object-input": "required",
        "result": {
                "@type": "Event",
                "name-output": "required",
                "startDate-output": "required",
                "endDate-output": "required",
                "location-output": {
                        "@type": "Place",
            ...
                }
        }
}
```

5.3 SearchAction

We use `Schema.org/SearchAction` when we want to search for events. We can define the attributes `location`, `startDate` and `endDate` and therefore search for event in a given location at within a given time slot. Listing 1.3 shows an Example of a `Schema.org/SearchAction`.

Listing 1.3. Schema.org/SearchAction

```
{
        "@context": "http://schema.org",
        "@type": "SearchAction",
        "query": {
                "@type": "Event",
                "location": "Innsbruck",
```

```
                "startDate": "2018-06-20T09:00:00Z",
                "endDate": "2018-06-20T20:00:00Z"
        },
        "query-input": "required",
        "result": {
                "@type": "Event",
        ...
        }
}
```

5.4 UpdateAction

We use `Schema.org/UpdateAction` when we want to modify an event. This action is basically the same as the `Schema.org/AddAction` but here we have the `identifier` as additional attribute of the `Schema.org/Event` object. Listing 1.4 shows an Example of a `Schema.org/UpdateAction`.

Listing 1.4. Schema.org/UpdateAction

```
{
        "@context": "http://schema.org",
        "@type": "UpdateAction",
        "object": {
                "@type": "Event",
                "identifier": "47621348721",
                ...
        },
        "object-input": "required"
}
```

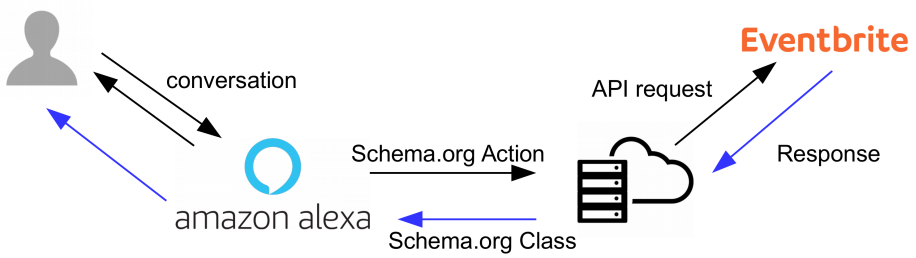
5.5 DeleteAction

We use `Schema.org/DeleteAction` when we want to delete an event. This action has only the `identifier` as attribute. Listing 1.5 shows an Example of a `Schema.org/DeleteAction`.

Listing 1.5. Schema.org/DeleteAction

```
{
        "@context": "http://schema.org",
        "@type": "DeleteAction",
        "object": {
                "@type": "Event",
                "identifier": "47621348721"
        }
}
```

6 Alexa Skill



Alexa communicates communicates with the user and creates a Schema.org Action. This Action is sent to the server that maps the Action to a Eventbrite API operation. The server makes then the request and gets a response. It maps the Eventbrite eesponse back to a Schema.org object which is sent back to Alexa. Alexa reads the response and tells it to the user.

6.1 User

The user communicates with Alexa and activates the defined Intents.

6.2 Alexa

Alexa executes the code for the activated Intents and communicates with the Server using Schema.org vocabulary.

6.3 Server

The server processes the Schema.org Actions and makes the requests to the Eventbrite API. It has to job to map Schema.org Actions and Objects to Eventbrite API operations and objects and back.

6.4 Eventbrite API

The Eventbrite API provides endpoints that allow to execute operations like add, update and delete events on Eventbrite. After executing this operations it sends back a response in JSON format to the server which maps it back to Schema.org.

7 Challenges and limitations

During designing and implementing our application, we have faced some challenges and limitations that affected our implementation process, one of the most vital challenges was solving the synchronization issue between Alexa and our server, in the beginning we have designed our backend using Node.js to perform

our Eventbrite API calls, however we have changed our backend implementation to use a PHP connected with Node.js to solve this synchronization issue.

Another main challenge we had, was performing creating event and deleting events, first we were working with Eventful (http://eventful.com/) as our event API source. We were able to perform search events accurately, however we have faced many limitations with having full permission to access post actions like creating, modifying and deleting events. After solving all constraints we found out that Eventful does not provide full access because of the new GDPR rules, thats why we have changed our event source website to use Eventbrite instead, as we could have the full access credentials and permissions we need to perform users requests.

Another limitation is the processing time of a request. Alexa waits only for 5 seconds and if the task do not respond within this time then Alexa says: "The requested skill took too long". When querying events from Eventbrite it can happen that it needs more than 5 second, especially when searching for events in big cities where there are many of them. Eventbrite does not provide a possibility to limit the data we get as response. A possible solution to this limitation is a Progressive Response, documented in Send the User a Progressive Response.

If Alexa is setup in English (US) it is not able to understand the main part of Austrian cities and street names – we do therefore not recommend to mix these languages.

Finally, the last limitation we can just create our events locally as a draft, because Eventbrite requires selling tickets for any created event to be confirmed.

8 Videos

We recorded examples of features we implemented and made the videos are available on YouTube under the following links:

Search for Events https://www.youtube.com/watch?v=wNbniBFqmzc

Create Event https://www.youtube.com/watch?v=qkaMfKySjWI

Delete Event https://www.youtube.com/watch?v=0fboYjQeaoU

9 Conclusion

We have managed to design and implement a voice-based chatbot to assist users to interact with Eventbrite remotely using the Alexa Skill in a simple and smart way. We may increase our platforms developing different interfaces using Facebook Messenger, Skype bots or AI.API, we left this point as a future work could be done further.