BotMan

An enterprise-grade framework for chatbot development

NLP API Integration Manager (Wit.ai)

Functional Specification

Version 0.1

Nachiket Joshi
(nachiket.joshi@sjsu.edu)

Version History

Version	Changes
V1.0	Initial Draft

Introduction

As the title suggests, the topic of this functional optimization document is An Enterprise-grade framework for Chatbot development. Before we dive into the specifications let us quickly understand some of the terminologies that are used extensively in this area.

Chatbots

From Wikipedia, the definition of chatbots can be understood as follows, A chatbot is a Computer Program which conducts conversation via auditory or textual methods.

Our previous documentation extensively talks about Chatbot so let us move forward to the contents that make up a cognitive chatbot.

The key to understand humans or any language for that matter is the ability to extract intent and the payload attached to that intent in any conversation. These intents then decide the course of relevant activities that can be performed in order to get the entity done.

NLP

A natural language processing is a science of extracting the intention of a text and relevant information from a text or audio. There is a reason why we see all these bot platforms suddenly popping out like mushrooms everywhere and that reason is the ever-increasing power of all the **NLP as a service** platforms.
Connecting to channels and then developing bots was never a problem and we demonstrated that in our previous project work in CMPE 272. The only missing link was a NLP platform which can scale and is easier to work with because you won't like to learn NLP just to make a simple silly bot.

Some of the popular NLP service platforms are,

1. Api.ai - by Google
2. LUIS.ai - by Microsoft
3. Watson - By IBM
4. Wit.ai - By Facebook

Each platform has some characteristics that you would like to be implemented in your service. Comparing them and finding the best will be a completely different aspect out of the scope of this functional spec.

Let us discuss some characteristics of an ideal bot platform should offer,

1. A NLP Service - Usually platforms opt for only one dedicated service but for our project we are going to give the user an option to choose among some platforms.

2. SDK - This will have the ability to handle conversations and their meta-data.

3. A platform to host the bot code

4. A platform to connect the bot logic, the NLP service and different channels.

While NLP as a service platform helps the developer in developing the NLP capabilities in the least amount of time possible, there definitely comes a time when developers find themselves out of wits to understand the basic jargon of NLP and training their NLP as a service to best of its capabilities.

Lets us dive into the NLP terminologies that we are going to use extensively over our development lifecycle,

INTENT

Simply put, intents are the intentions of the end -user, these intentions or intents are conveyed by the user over the chars they perform with the bot over time. The intents can be classified in two parts,

1. Casual Intents

These are mainly the opening or the closing statements in any conversations. We can also call then as **Small Talk** intents.

The casual intents also comprise of Affirmative and negative intents for utterances like "OK", "Yes Please", "NO", "I would like to" phrases.

A simple example in bots perspective could be as follows, let us suppose a bot asks a user a question where it expects **Yes/ No** answers but if the user gives something else and the bot is confused in extracting the intent the logical procession would be to ask the question again till we get expected intent.

2. Business Intents

These are the intents that directly map to the business of the bot. A simple example could be, when a user asks a bot to help order a pizza or when a user asks a movie information bot, "Which movie received Oscars in 2016?" In our cases, we are going to emphasize more on the business intents than the casual ones.

Entities

Business intents usually have metadata regarding the intent that they find and it is called as "Entities". Let's take an example for the Intent **"FindMovieReleaseYearByTitle" —** Sample Utterance is "When was {movie-name} released?" Here "movie-name" is the title of the movie for which the user "intends" to find out the release year. As a user of NLP as a service we don't need to get into the technicalities of knowing how **POS tagging** works but we are only interested in finding the year of the movie release.

Whenever we are thinking about designing our intents the entities must also be identified and labeled accordingly. Entities are like payloads in any API or the additional information which is in some cases a must to carry out the intent. In entities, we can also have general entities labeled for use throughout the intents like metrics (including quantity, area, and count), Dates and most of NLP as service allows you to tag entities of such general types without any big hassle.

We can label some entities as **'composite entities'**. These are the intents that are having more than one entity (component entity) inside it. As a science, it doesn't matter if you don't have this feature with your NLP service as long as you have simple entity labeling. If we want to tackle the multi intent entities we must define component entities before labeling composite entities. All these aspects are provided by the NLP platforms and are free to program. I have tried to execute one simple program to understand everything that the intent/entity relationship represents.

The aspect of NLP that I will be working upon throughout the project is the WIT.ai from Facebook.

WIT.AI

The Google definition of Wit.ai is a NLP service that makes it easy for developers to build Siri like speech interface for their app or device. It's basically an API that turns natural language (speech or messages) into actionable data.

Previously the wit.ai provided an API that takes our text or voice input and returns intents and entities. We then were able to add our own bot APIs behind those intents to perform a logical course of actions. So basically, depending on the user input, your application can take actions or ask more questions to fulfill the user's request.

However, in recent days, the Wit.ai has released something called **"Bot Engine",** with this they have introduced one more aspect called **'Stories' and 'Contexts'**.

So the **contexts** are the background of a certain action that user expects from the bot. this could be session specific. Thus it becomes of foremost important to store the expected context in a successful intent resolution.

References

1. https://medium.com/@brijrajsingh/chat-bots-designing-intents-and-entities-for-your-nlp-models-35c385b7730d

2. https://chunksofco.de/wit-ai-explained-part-1-bot-engine-stories-and-actions-4b5be87ecf38

3. https://www.themarketingtechnologist.co/api-ai-vs-wit-ai/

4. https://www.codementor.io/srijansaxena/how-to-make-a-basic-chatbot-wit-ai-part-1-7g82j81wu

5. https://github.com/wit-ai/wit-api-only-tutorial

6. https://www.quora.com/topic/Wit-ai

Requirements

As shown in the configuration diagram, we will have an integration manager in the framework. A user will have a choice to choose the NLP platform of his choice and his bot template will be generated.

So the basic requirement is that a user must be able to select the template and his bot must use that specific NLP as a service.

Required functions of an Integration Manager are as follows;

1. Operations On Bot: Add, Remove
2. Operations on Intents: Resolve, Query
3. Operations on Entities: Find, ask, enquire, create payload
4. Query Bot: Get the entity payload as input and does the heavy-lifting.

Functional Overview

We are going to train our NLP as a service with some real corpus; this means the chat that our clients will do with the bot will eventually be used to train the bot. If you are the user of our bot framework, these chat messages can be replaced with your clients over Facebook or Skype or whatever channel you work with those messages/utterances can help training for intents, otherwise, you can even think of training for intent with your own "Manufactured" utterances for any intent. We will also try to implement this initial training in some business context so it should not be a very big hassle for a new business to start integrating with us.

Manufactured utterances are the intents that are taught during the inception of the NLP bot. The training with Manufactured utterances helps in bootstrapping the system but one must re-train the NLP service when it starts getting some real utterances. This retaining part will be implemented in the form of a server hosted on the cloud.

The process of re-training the system should keep going on until the error rate reduces. The error rate reduction can be postulated with the help of the confidence of the answers that the bot produces. The more variance of utterances you receive from real conversations the better you can train your NLP service for intents. Ideally, Minimum 5 or optimally 10 utterances per intent are good enough to bootstrap the system.

From Medium's article on Chatbots, following could be some intrinsic training rules for Intents and Entities,
Rules of Training a machine learning model,

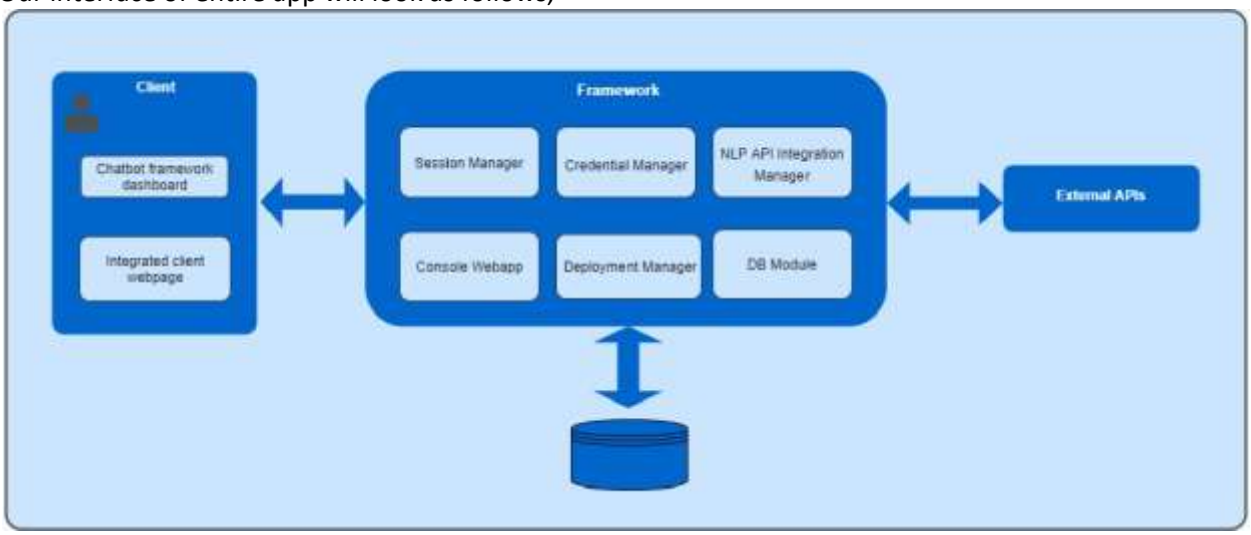
Rule #1: You can never train it enough. Get more data, train it more
Rule #2: You can never train it enough. Get more data, train it more
Rule #3: You can never train it enough. Get more data, train it more

The NLP services including the wit.ai go through a routine of **Supervised** — **Unsupervised** — **Supervised** learning phases. Each supervised learning phase serves as a feedback loop through which the course correction is done for the NLP models. In our case, the bootstrapping phases where we will decide some

of the intents will the supervised modules. The over the time learning scripts that we will develop will contribute to unsupervised learning phase.

Configuration

Our interface of entire app will look as follows,



Briefly describing the image; If we closely look our Framework module, we can find that NLP API integration manager. This manager will consist of different NLP as service platforms. We have decided to use at least two of them viz api.ai and wit.ai.

Debug

The basic debugging is performed with **Atom** and **IntelliJ** inbuilt functionalities. There are multiple add-ons in the **IntelliJ** and atom that save users time by giving optimum debug aspects of the code. When the notification manager runs, programmers can easily set breakpoints, check values of variables and do more.

Logging

In notification manager, we do have options for logging different things on the console. In the atom, we have a **safe mode** where we can see the progress of almost every variable and log status. Atom is pretty good when it comes to handling logging error responses and information.

Implementation

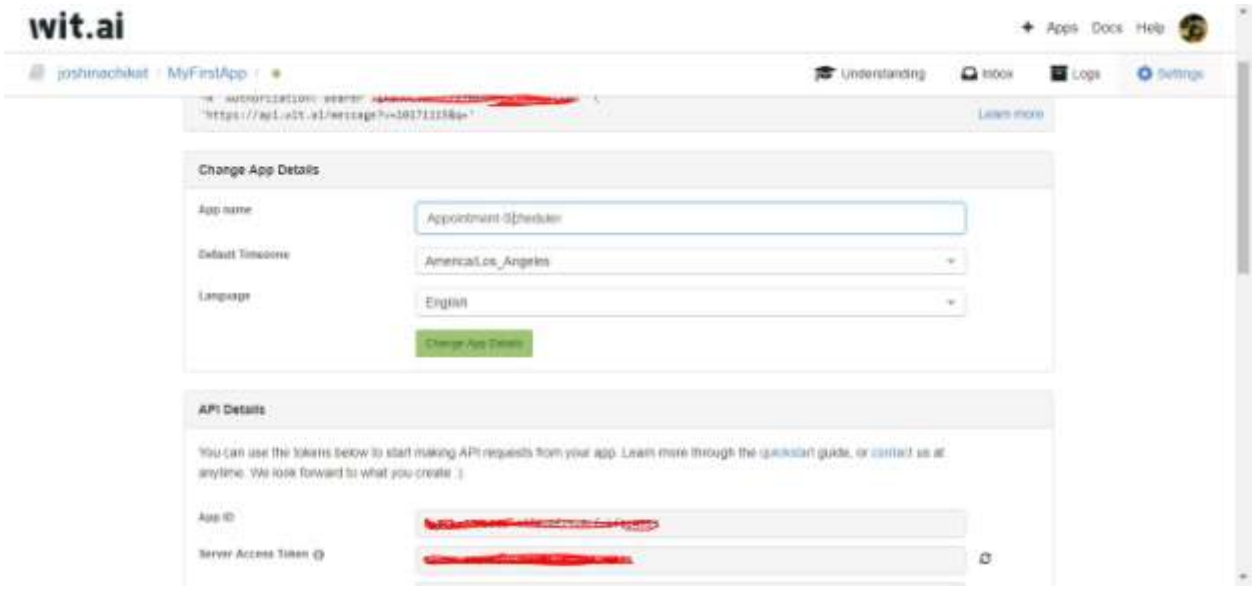
I tried to create one **Appointment Scheduling Bot** which will use everything that we discussed till this point,
The bot will be doing following things,

1. Create the app through API
2. Train app using existing data
3. Add build-in entities
4. Handle ambiguous queries
5. Implement a feedback loop to let me train existing entities
6. Let users (me) expand the scope of your app.

I found a GitHub description that exactly matched with all the research and development requirements and thus decided to follow these steps to understand how wit.ai works and how I can implement the intuitive training in our bot framework,

Steps I followed,

1. Get Your Seed Token from WIT API console window.

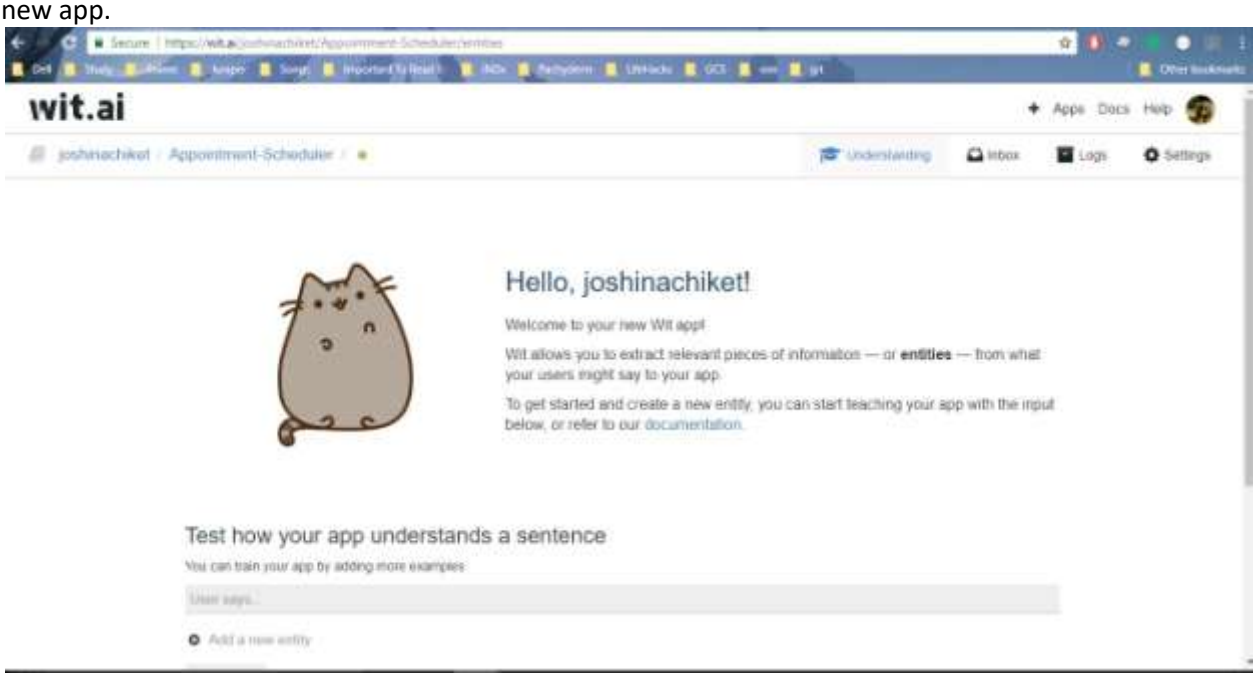


2. Creating the App

I used the following cURL command to create first project. This step could also be done through the console.

```
curl -XPOST 'https://api.wit.ai/apps?v=20170307' \
 -H "Authorization: Bearer $BASE_TOKEN" \
 -H "Content-Type: application/json" \
 -d '{"name":"appointment_bot",
      "lang":"en",
      "private":"false"}'
```

Now the link, `https://wit.ai/{your_username}/appointment_bot` will have my new app.



3. Updating the App

Using the following cURL,

```
curl -XPUT 'https://api.wit.ai/apps/$NEW_APP_ID?v=20170307' \
  -H "Authorization: Bearer $NEW_ACCESS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"timezone":"America/Los_Angeles"}'
```

The access token can be found in the first step.

4. Training with Existing data

The initial training I created was in the form of excel spreadsheet,

The screen shot shows the exact intent mapping that initial training was fed with

1	I want to make a new appointment	appt_make
2	Show my appointments	appt_show

After this csv, we just need a script to read this file and map the intents with their entities and payload management functions.

Following screenshot, I have a small script to just read the SCV for the same,

```
function validateMyCSV(samples) {
  return fetch('https://api.wit.ai/samples?v=20170307', {
    method: 'POST',
    headers: {
      Authorization: `Bearer ${NEW_ACCESS_TOKEN}`,
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(samples),
  })
    .then(res => res.json())
}

validateMyCSV (samples)
  .then(res => console.log(res));
```

5. Data Detection

Wit has the build in entity for this kind of apps which can be explained with the following script,

```
curl -XPOST 'https://api.wit.ai/samples?v=20170307' \
  -H "Authorization: Bearer $NEW_ACCESS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '[{
        "text": "Appointment for tomorrow, please",
        "entities": [
          {
            "entity": "intent",
            "value": "appt_make"
          },
          {
            "entity": "wit$datetime",
            "value": "tomorrow",
            "start": 26,
            "end": 34
          }
        ]
      }]'
```

6. Send Responses

The next question was to map responses to the intents. Above two steps will make sure that we have

proper intents extracted from the CSV. Here we can easily use a simple switch case statement and do the needful,

A sample switch case statement can be as follows,

```
function handleIntentMessage(question) {
  return queryWit(question).then(({entities}) => {
    const intent = firstEntity(entities, 'intent');
    const dateTime = firstEntity(entities, 'datetime') || {};
    if (!intent) {
      console.log(Can't Find Intent, Try Again :)');
      return;
    }
    switch (intent.value) {
      case 'appt_make':
        console.log('making an appointment, please wait',
dateTime);
        break;
      case 'appt_show':
        console.log('here are your appointments', dateTime);
        break;
      default:
        console.log(`  ${intent.value}`);
        break;
    }
  });
}
```

7. Manage Uncertainty

These steps make sure that the app answers according to what we have trained it. The confidence, factor in these cases, is more than 99.99%. But what id the WIT is uncertain in deciding the intent? This can happen for following two reasons,

1. App needs more training, remember the training laws that we learned earlier?
2. The question is genuinely ambiguous.

To solve these we can **N-Best** feature in the app. The approach of N-Best is to narrow down the exact intent by users help,

We can create a script for this. I have used following script from the internet, for the sample appointment scheduling app,

```
const N = 3;
const THRESHOLD = 0.7;
function handleMessage(question, readline) {
  return queryWit(question, N).then(({entities}) => {
    const intents = entities['intent'] || [];
    const bestIntent = intents[0];
    const dateTime = firstEntity(entities, 'datetime') || {};

    if (!bestIntent || bestIntent.confidence < THRESHOLD) {
      console.log('  what would you like to do?');
      intents.forEach(intent => console.log(`\n --
${intent.value}`));
      readline.question('choice > ', choice => {
        console.log(`  okay, running > ${choice}`);
      });
      return;
    }
    // ...
  });
}
```

8. Approach where users can train the App

The concept here is that over the time many messages are generated and the wit will have produced many successfully intents over the time, now the challenge is to formulate these intents in a script so that the users can teach the app itself.

For ex: Some unambiguous reply at first can be confirmed with the user, mapped to the payload and then saved for any future such user query.

We can also write simple scripts for this.

9. Conclusion

This is just an example app that was developed for research, mainly to understand the theory regarding the wit APIs. The actual challenge will be to incorporate this as a detachable module in our framework. That work will definitely not be as easy as writing scripts and analyzing the answers in the project directory in wit.

This example app is just a part of a vast puzzle that needs to be sorted out in a correct order. The pieces need to be put together to form a picture in our case the Framework. In the final version, this sample app and one more app that I am working on which uses sample hospital data to help doctors query patient information will be used to be part of the **NLP integration manager.**

Testing

Software testing is one of the most important processes in any software development project. The testing includes validation and verifications both.

General Approach

Testing is a process that starts when we start writing the first line of code or rather before even that. We are going to follow a Test Driven Approach for development. Our development will be driven by the Unit Tests. A JavaScript framework for writing unit tests will be used called the Chai and Mocha JavaScript.

Unit Tests

Currently, most of the efforts are being done towards research and development in utilizing wit.ai as a standalone app, understanding how intent matching works, entity extraction works

The aspect will test individual units of the code. Intuitively one can assume a unit test as the smallest testable part of any application. We will be using unit testing for API testing and functional testing. We will try to incorporate these aspects into standalone test scripts.

But in future, we are going to write the unit test scripts to test the intent matching, intent comparison, entity comparison.

Basic nature of unit tests will be as follows,

Serial Number	Test Case	Expected Result	Received Result
1	Intent Matching	Given chat resolves to some intent	Wit.ai arrives at the same intent as expected result
2	Intent Comparison	Given chat resolves to some intent	Wit.ai arrives at the same intent compared to expected result
3	Entity Matching	Given intent requires some entity	Wit.ai asks user the same entity as in expected results
4	Story Resolution	Given chat extracts some story	Wit.ai arrives at the same story as the expected results
5	Context Extraction	Given chat extracts some context	Wit.ai arrives at the same context as the expected results.

Appendix

NLP: Natural Language Processing
API: Application Programming Interface