BotMan - An enterprise-grade framework for chatbot development

NLP API Integration Manager (Dialogflow)

Functional Specification

Version 1.0

Sushant Vairagade
sushant.vairagade@sjsu.edu

TABLE OF CONTENTS

Version History 3
Introduction 4
References 4
Requirements 5
Functional Overview 6
  Configuration 8
  External Interfaces 8
  Dependencies 8
  Debug 9
    Logging 9
Implementation 9
Testing 13
Appendix 14

VERSION HISTORY

Version	Changes
1.0	Initial Draft
1.1	Update Dialogflow API version

INTRODUCTION

Natural Languag Processing is one of the crucial requirement for chatbot application development. NLP implementation requires extensive research work. Enterprises don't prefer to spend their resources on building an NLP platform from the scratch. Companies like Google, IBM, Microsoft have already invested a lot of their resources into developing an NLP platform. They have exposed NLP engine as a service which can be used by other applications. Enterprises prefer using these NLP engines which are readily available in the industry over developing their own in-house NLP platform. It also reduces their maintenance cost of the system as these platforms are maintained by the host company.

Dialogflow(api.ai) and Wit.ai are one of the most widely used and accepted NLP platforms. BotMan framework is designed to the user a choice of selecting the most suitable NLP APIs available in the industry. This easy-to-use integration will allow users to save a considerable amount of their development time. Users can start developing their chatbots by only focusing on the business logic without having the need to write integration code.

The NLP platforms currently available in the market use different approaches for solving the problem of NLP, still, they use same building blocks required for developing a chatbot. The same set of configuration requirements for these NLP platforms enables BotMan framework to create a generic interface that can be used to integrate with different NLP platforms. This enables the user to select their choice of NLP platform without having the need to worry about the integration with NLP platform.

This document will cover functional requirements for the NLP API integration manager. The document will discuss technical specifications and requirements for Integration manager.

REFERENCES

1. Dialogflow reference document. https://dialogflow.com/docs/reference/agent-json-fields
2. Code samples. https://dialogflow.com/docs/examples/
3. Sl4j logging. https://www.slf4j.org/apidocs/overview-summary.html
4. Microsoft bot framework: https://docs.microsoft.com/en-us/bot-framework/
5. Workbook II
6. Embedding chatbot: https://miningbusinessdata.com/embedding-api-ai-chatbot-into-your-wordpress-site/

REQUIREMENTS

Below are the functional requirements for Integration manager -

1. The system should allow user to create, retrieve and delete bots.
2. The system should be able to create, update, retrieve and delete intents for a bot.
3. The system should allow user to create, update, retrieve and delete entities for a bot.
4. The system should persist the created bots, intents and entities.
5. The system should expose a generic interface to the user irrespective of the NLP platform choice made by the user.
6. The system should maintain the credentials per NLP platform.
7. They system should maintain detailed logs of all the activities performed by the user while making calls to the NLP API.
8. The system should externalize environment constants.
9. All the components should be Junit tested.

FUNCTIONAL OVERVIEW

Below is the high-level architecture diagram of BotMan framework:

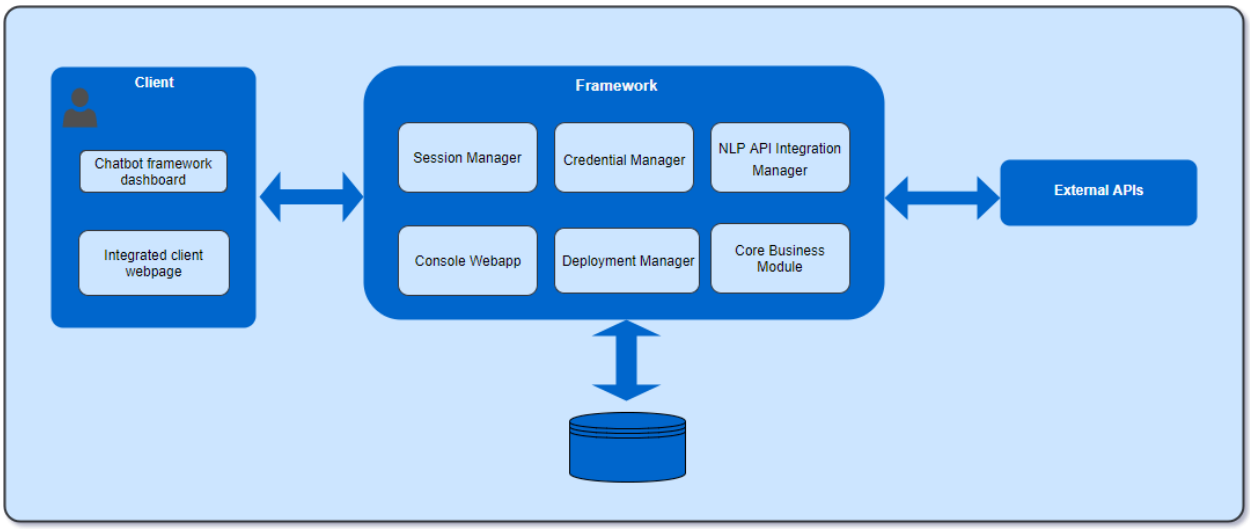

Figure 1 BotMan Framework

BotMan framework is a composite of many components. Each component is responsible for performing a specific task. Together these components work towards the common goal of helping the user in creating and managing chatbots. This document will discuss the Integration Manager, its components, and internal working.

Integration manager provides an interface to the users for creating and managing their bots. It acts as a middleware or an adapter between the external NLP APIs and the other components of the framework. By providing a generic interface it avoids the need for the code changes while switching between the NLP API providers. By implementing factory pattern, IntegrationManagerFactory provides an instance of the implementation of IntegrationManager interface depending on the choice of NLP API provider.

Integration manager should generate detailed system logs. These logs should be sufficient for debugging the system by capturing the series of events occurring in the system. The logs can be examined to get the sequence of actions that a user performed on a chatbot. The logging level can be easily modified by externally configuring a property file.

Below is the class diagram for Integration Manager:

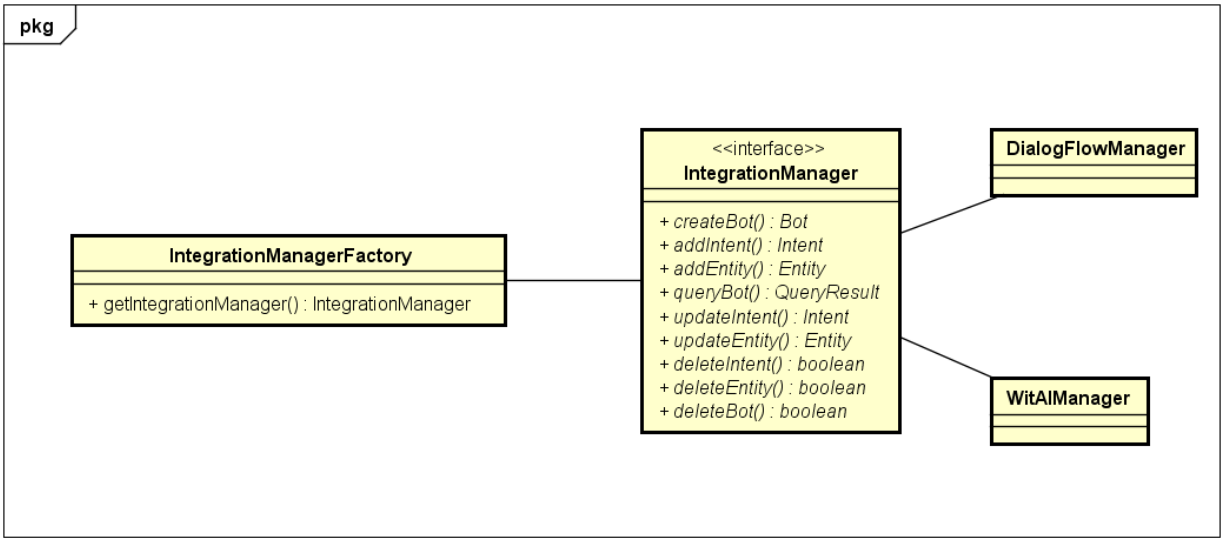

Figure 2 Integration Manager

FUNCTIONS OF INTEGRATION MANAGER

1. **Operations on Bot:**

   The system should allow the user to create, fetch and delete chatbots. NLP providers have a different process for creating new bots (agents) and removing them. Integration manager will take care of these differences and will make the process look like platform agnostic.

2. **Operations on Intents:**

   Intents are responsible for mapping user statements to an action and extracting the entities from the provided sentence. The system should allow the user to create, retrieve, update, and delete intents.

3. **Operations on Entities:**

   Entities are used to identify parameters in a sentence and extract their values. These parameter values are then used to perform the action configured for the intent in context. The system should allow the user to create, retrieve, update, and delete entities.

4. **Query Bot:**

   The system calls the external API to perform NLP. Upon receiving a successful response, the system returns a detailed response to the user with below details:
   a) Intent
   b) Action
   c) Entities (key, value pairs)

Basic Operational steps:

- Validate request – Strings provided for bot name and description are not empty or null, the bot name is not duplicate for the user, description is at least 6 characters long.
- Log the user request.
- Validate user and user's session.
- Call external API to create a bot.
- Persist the response details on successful service call response.
- Provide appropriate response to the client.
- Log the service response.

CONFIGURATION

To keep Integration manager as flexible as possible, the configurable properties are externalized by setting up a set of property files per environment. The system will read these properties at runtime, the externalization and runtime read allow the service to change configuration at runtime without having the need to restart the service, no downtime.

1. Set up base URL for external API in the configuration file.
2. Set external API's version to be used.
3. Set up client authentication tokens.
4. Set timeouts for external calls.
5. Set current environment; DEV, TEST, PROD.
6. Set up Logger implementation for SLF4J.

EXTERNAL INTERFACES

Integration manager will be used directly by core business module in the framework. When the user tries to create a new bot, or perform an action on existing bot, web app sends the request to core business module which further communicates with Integration manager. A client application that has chatbot deployed will also communicate indirectly with integration manager to query chatbot.

1. Core business module to perform bot creation and configuration operations.
2. Client application to query chatbot.

DEPENDENCIES

Integration manager is an independent micro-service that can run on its own without interacting with any other modules in the framework other than Session manager, which it uses to validate user session. Although it has only one internal dependency it is directly dependent on the status of external NLP APIs for below factors:

a. Response time.
b. Failure rates.

DEBUG

The easiest tool to use for debugging a Java application is the plug-in provided by most of the modern IDEs. The debugger can debug a complex system by adding breakpoints. Host application must run in debug mode. Based on the choice of NLP platform user can add a breakpoint for a specific operation on one of the implementations of IntegrationManager interface. Ex. If debugger wants to debug the process of Intent creation while using Dataflow as a choice of NLP platform then a breakpoint can be added in the createIntent method of DataflowManager class. This will allow developers to validate variable values at any given point of time in the execution.

LOGGING

Integration manager uses SLF4J for logging. SLF4J being an abstraction layer gives us a choice of logging implementation to use. This avoids tight coupling between the application and logging implementations like log4j. Logging comes handy when it is not possible to debug the system or if we are unable to recreate the issue. Logs can be configured to be produces at different levels; DEBUG, INFO and ERROR. Debug level needs to be configured in the system before the application starts. Logs are the main source of information for debugging an issue on production as they provide important data like date and time the error occurred, sequence of activities that occurred, and a stack trace.

IMPLEMENTATION

Integration manager is a standalone application that can be added as a dependency to any application that needs to interact with external NLP APIs.

Maven dependency to include Integration Manager:

```
<dependency>
   <groupId>com.botman</groupId>
   <artifactId>botman-integration</artifactId>
   <version>1.0.0-SNAPSHOT</version>
</dependency>
```

Maven Dependency of api.ai SDK:

```
<!-- https://mvnrepository.com/artifact/ai.api/sdk -->
<dependency>
   <groupId>ai.api</groupId>
   <artifactId>libai </artifactId>
   <version>1.6.12</version>
</dependency>
```

Integration manager uses Dialogflow(api.ai) SDK to make client calls to the deployed agent (bot). Below is the sample code for querying chatbot using SDK:

Code Sample

```
private AIDataService getDataService(String apiClientToken) {
        AIConfiguration aiConfiguration = new AIConfiguration(apiClientToken);
        return new AIDataService(aiConfiguration);
 }

 private String queryBot(String query, String apliClientToken) {
        try{
                AIRequest aiRequest = new AIRequest(query);
                AIDataService aiDataService = getDataService(apliClientToken);
                AIResponse aiResponse = aiDataService.request(aiRequest);
                if (aiResponse.getStatus().getCode() == 200) {
                        return aiResponse.getResult().getFulfillment().getSpeech());
                } else {
                        System.err.println(aiResponse.getStatus().getErrorDetails());
                        return null;
                }
            } catch (Exception ex) {
      ex.printStackTrace();
                 return null;
            }

        }
 }
```

While Integration manager uses SDK for client interactions with chatbot, it uses the API services exposed by Dialogflow for maintenance tasks done by the chatbot owner.

Below are sample Rest API calls:
Query Bot to buy a pizza:

```
curl \
-H "Authorization: Bearer ed01c1d554904ae4aa06fbb02e3baca8" \
https://api.dialogflow.com/v1/query?v=20170712&contexts=buy&lang=en&query=pizza&sessionId=5214&timezone=America/Los_Angeles
```

Create entities:

```
curl -X POST \
'https://api.dialogflow.com/v1/entities?v=20170712' \
-H 'Authorization: Bearer ed01c1d554904ae4aa06fbb02e3baca8' \
-H 'Content-Type: application/json' \
```

```
--data '{
  "entries": [{
     "synonyms": ["pizza", "bread slice", “slice”],
     "value": "pizza"
    },
    {
     "value": "burger"
    }
  ],
  "name": "foodItem"
}'
```

Create Intent:

```
curl -X POST  \
'https://api.dialogflow.com/v1/intents?v=20170712' \
-H 'Authorization: Bearer ed01c1d554904ae4aa06fbb02e3baca8' \
-H 'Content-Type: application/json' \
--data '{
  "contexts": [
    "buy"
  ],
  "events": [],
  "fallbackIntent": false,
  "name": "shopping-cart-add",
  "responses": [
    {
      "action": "addShoppingCart",
      "affectedContexts": [
        {
          "lifespan": 2,
          "name": "buy",
          "parameters": {}
        },
        {
          "lifespan": 2,
          "name": "chosen-food",
          "parameters": {}
        }
      ],
      "defaultResponsePlatforms": {
        "google": true
      },
      "messages": [
        {
          "platform": "google",
          "textToSpeech": "How many $food?",
          "type": "simple_response"
        },
      ],
      "parameters": [
        {
```

```
        "dataType": "@food",
        "isList": true,
        "name": "food",
        "prompts": [
          "I'm sorry, I didn't get that. What food did you want?"
        ],
        "required": true,
        "value": "$food"
       }
     ],
     "resetContexts": false
    }
  ],
  "templates": [
    "@food:food ",
    "Add @food:food ",
    "I need @food:food "
  ],
  "userSays": [
    {
      "count": 0,
      "data": [
        {
          "alias": "food",
          "meta": "@food",
          "text": "pizza",
          "userDefined": true
        }
      ]
    },
    {
      "count": 0,
      "data": [
        {
          "text": "Add "
        },
        {
          "alias": "food",
          "meta": "@food",
          "text": "burger",
          "userDefined": true
        }
      ]
    }
  ],
  "webhookForSlotFilling": false,
  "webhookUsed": false
}'
```

TESTING

GENERAL APPROACH

The features of integration manager discussed in this document should be unit-tested. Using a test-driven development each implementation of the IntegrationManager interface will have test cases for both positive and negative scenarios. For integration testing with external APIs, it is necessary to test the consistency of the system's implementation and the external APIs interface. Any change in the external APIs interface could result in improper functioning of the system, it is necessary to validate the current version of the client interface to that of the current version of the external API. For offline testing, Mockito framework can be used to mock the responses of external APIs.

UNIT TESTING

TEST CASE NUMBER	NAME	PURPOSE	EXPECTED RESULT
1	Validate session	To test is a user is valid i.e. authenticated and has a valid session token within TTL.	User has a valid session token and is within its TTL (time to live).
2	Create Entity/Intent/Bot	To test if an Entity/Intent/Bot is created with the specified list of probable values.	Received success response (200) from external API. Persisted the changes in the DB.
3	Update Entity/Intent/Bot	To test if an Entity/Intent/Bot is updated with the specified list of new values.	Received success response (200) from external API. Persisted the changes in the DB.
4	Delete Entity/Intent/Bot	To test if the specified Entity/Intent/Bot is deleted.	Received success response (200) from external API. Persisted the changes in the DB.

5	Intent test	To test the system by querying a chatbot with a sentence.	The system should successfully resolve the query to a valid intent and extract entity values.

APPENDIX

1. Log4j logging framework: https://logging.apache.org/log4j/1.2/manual.html
2. Embed chatbot to HTML: https://discuss.api.ai/t/how-do-you-embed-a-chat-bot-in-html/296