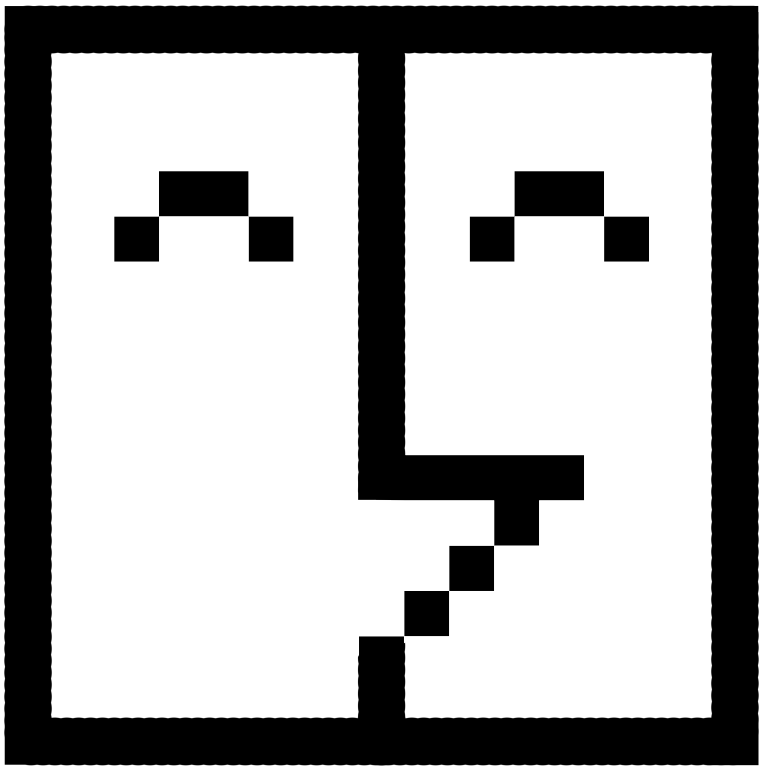

# Conversational Chatbots

*AMOS Project WS 2017/18 - Technical University Berlin in cooperation with Actano GmbH*

Documentation

**Table of Contents**

# 1. Introduction and Objective

This Conversational Chatbot project was part of the AMOS project course at the Technical University of Berlin during the Winter semester of 2017-2018. The project was carried out in cooperation with Actano GmbH. The aim of the course is to make software development process more flexible and smoother to participants, by following agile software development methods. In addition, as the course aims to provide participants with real world conditions facing software development processes, the projects carried out in the course are offered from industrial partners.

The main objective of the developed conversational bot project is to deliver a smalltalk, in alignment with chat bots and the RASA framework, based on natural language processing. The conversational bots were designed to develop dialogues with a user about agile software development methods, involving different aspects of Scrum and Kanban. For both conversational bots, wherever applicable, a more detailed description is offered after having introduced a certain topic. This, for instance, is the case when the scrum bot has introduced the roles of scrum, so that the bot gives the user the opportunity to know more about the Product Owner, Scrum Master or Software Developer roles. However, not all topics are described in detail.

The Team consisted of 7 contributors, who have developed the conversational bot in a total of 13 sprints, each sprint lasting one week. Every team member has played the role of being a product owner or a scrum master at least one time, while for the rest of the weeks contributing as a software developer. In addition, the team participated in different workshops with the team of Actano GmbH, where both parties agreed on the intermediate and the final requirements that the conversational chatbot project is expected to deliver.

This documentation aims to give an overview at the pre-requisites that you need to take into consideration, if you were interested in using the conversational chatbot, as well as introducing the technical architecture on which the bot has been constructed operating, and the providing the role for each used technology during development in more detail.

The github repository for this project can be found at:
https://github.com/amos-ws17/amos-ws17-proj1

# 2. User Manual

If you would like to use the conversational bots, please follow the instructions that are provided by this section.

## 2a) Pre-requisites

### i) Python - version 3.6.

For more Information, please refer to:
https://www.python.org/downloads/release/python-363/

### ii) venv - Virtual Environment for executing Python code

Install a virtualenv in a desired location

```
sudo pip install virtualenv
# cd to our desired location
virtualenv -p python venv
```

Start your venv

```
source venv/bin/activate
```

**Please note:** your venv has to be active while executing the bots. The following step is intended only as a general note.

While you are still in the terminal, you may deactivate venv

```
deactivate
```

### iii) Install RASA Framework

Install Rasa NLU (Version 0.10.16)

```
pip install rasa_nlu
```

For more Information regarding the installation of RASA NLU, please refer to:
http://rasa-nlu.readthedocs.io/en/latest/installation.html

Install Rasa Core (Version 0.7.8)

```
pip install rasa_core
```

For more Information regarding the installation of RASA NLU, please refer to:
https://core.rasa.ai/installation.html

**iv) Install Python libraries:**

In order to be able for the python code to be fully operational, the following libraries are required to be installed: **spacy, scikit-learn, flask and pytest.**

```
pip install -U spacy
python -m spacy download en
pip install -U scikit-learn scipy sklearn-crfsuite
pip install flask
pip install -U pytest
```

## 2b) Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

First you need to deploy the project on a live system, in order to be able to fully operate the bots. After having performed these steps, you will be able to operate the conversational bots locally without the use of the Redis database.

### *Deployment*

To deploy the system make sure you have docker and git installed. If you do, clone this repository with:

```
git clone https://github.com/amos-ws17/amos-ws17-proj1.git
```

Then navigate to the newly created *amos-ws17-proj1* folder and type in the following command to start deploy the bots:

```
docker-compose up
```

Then, we need to train and start the HTTP service, through which a request and a response between the client and the server can be established.

***Train and run the service***

*Start your venv*

```
# cd to our desired location
source venv/bin/activate
```

*Train the models*

```
# cd to amos-ws17/workstreambot
python -m train -d scrum+kanban -n nlu_training_data_full.json
```

It is recommended to train both models, scrum and kanban, in order to be able to run both bots, hence the command included `scrum+kanban.` The same also applies for the next step, when running a local HTTP service, without the use of Redis database. For running this step, you also need to use `scrum+kanban.`

*Run the service*

```
python -m http_service -d scrum+kanban
```

After having successfully ran the service, you have established an HTTP service, to which you can send a request on the web.

Optionally, you can run the tests that have been implemented in order to automatically verify the correct responses on the sent requests for the conversational bots. In order to achieve this, please note that both bot models have to be trained first, before performing the next step.

*Run the tests*

```
python -m pytest test/ # execute all component tests
```

## 2c) How to use the bot

Having trained the NLU models and established an HTTP service, the bot can now be accessed at the following URL on any web browser:

*http://localhost:5000/service/converse/session?query= <your_query>*

Assuming that the user would like to start the Scrum bot, the best way to do this would be to ask the bot "What is Scrum?". The bot is expected to give an introduction about scrum and to guide the user through different aspects of Scrum, and at the same time give him the opportunity to go into detailed topics to the shown aspects. The example below shows an example of the response that the user is expected to get, upon entering the following URL:

*http://localhost:5000/service/converse/session?query=What%20is%20Scrum*

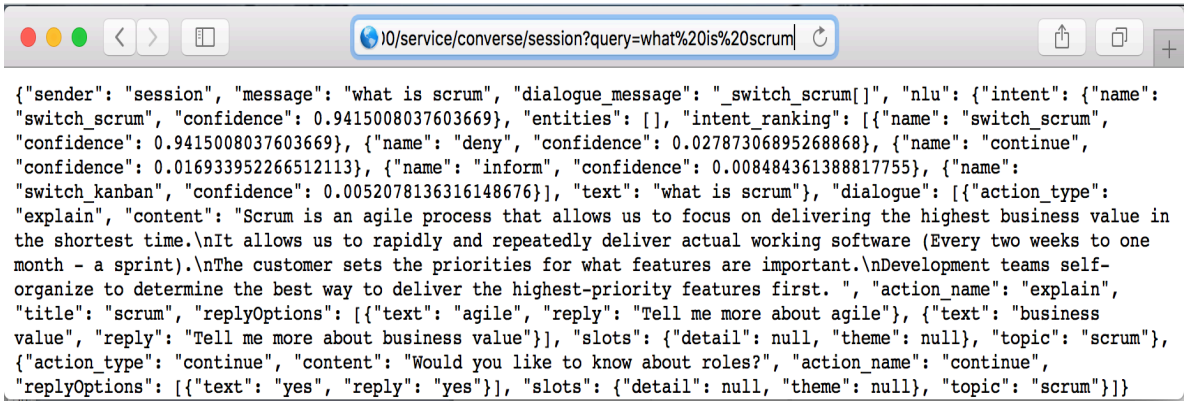


Image a): Introduction to scrum after having ran the bot

As shown in Image a) the bot has provided a JSON response to what Scrum is, which is basically the text included in the content of the JSON string, which is the direct response to the user's request. At the end the bot provides some reply options, in the following order:

- If the user is interested to know more about "agile", they should enter the following query:
  *http://localhost:5000/service/converse/session?query=Tell%20me%20more%20about%20Agile*
- If the user is interested to know more about "business", they should enter the following query:
  *http://localhost:5000/service/converse/session?query=Tell%20me%20more%20about%20Business*
- If the user is interested to continue to the next scrum aspect, which is located under the "action_type": "continue", they should enter the following query:
  *http://localhost:5000/service/converse/session?query=yes*

This mechanism can be applied to every response that the user receives from the bot, upon entering a query. However, the next section 2d) Further Features describes all special functionalities about the conversational bot, and how to perform them.

## 2d) Further Features:

### *Resetting conversational bot*

For clearing the conversation dialogue and starting everything all over, use the following query:

*http://localhost:5000/service/converse/session?query=reset%20dialogue*


Image b): Resetting a bot conversation

Upon successfully resetting a conversation, you should get the message shown in image b) displayed, informing you that the conversation history has been successfully deleted.

### *Switching to the Kanban bot*

If you would like to switch to the Kanban bot, you can start doing that, preferrably by the following query "What is Kanban?":

*http://localhost:5000/service/converse/session?query=What%20is%20Kanban*

### *Switching back to the point where you had left the Scrum bot or vice versa (Paused conversation)*

If you are conversing with either the Scrum or the Kanban bot, you may switch to the other bot, as follows:

- While you are conversing with the Scrum bot and wish to switch to the Kanban, enter:
  *http://localhost:5000/service/converse/session?query=What%20about%20Kanban*
- While you are conversing with the Kanban bot and wish to switch to the Scrum bot, enter:
  *http://localhost:5000/service/converse/session?query=What%20about%20Scrum*

Upon switching from one bot to the other, you have the option to either continue the other bot at the question where you had left it before switching to the other bot, or to reset it and start from its first question. The options you have upon running the above referenced requests while already conversing in the other conversational bot can be shown in image c).

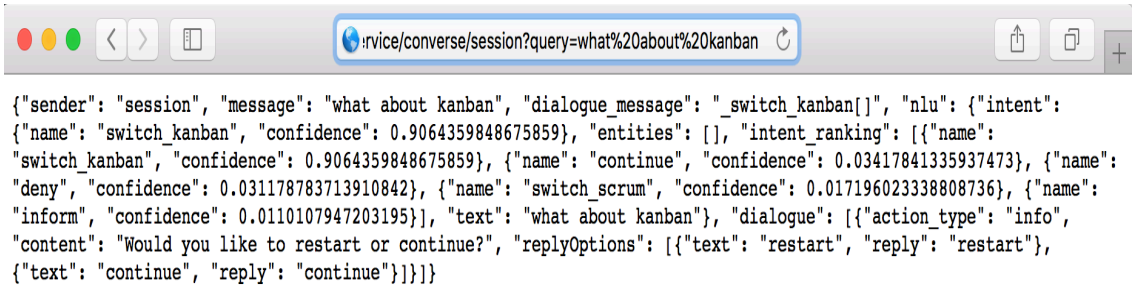


Image c): Upon switching from one bot to the other, users can either continue where the bot at the point they had left it or restart the bot

As shown in image c), the dialogue content informs the user that they are either able to restart the bot by typing "restart" or continue the bot by typing "continue".

# 3. Architecture

This section provides an overview to the structure of the conversational bot system.

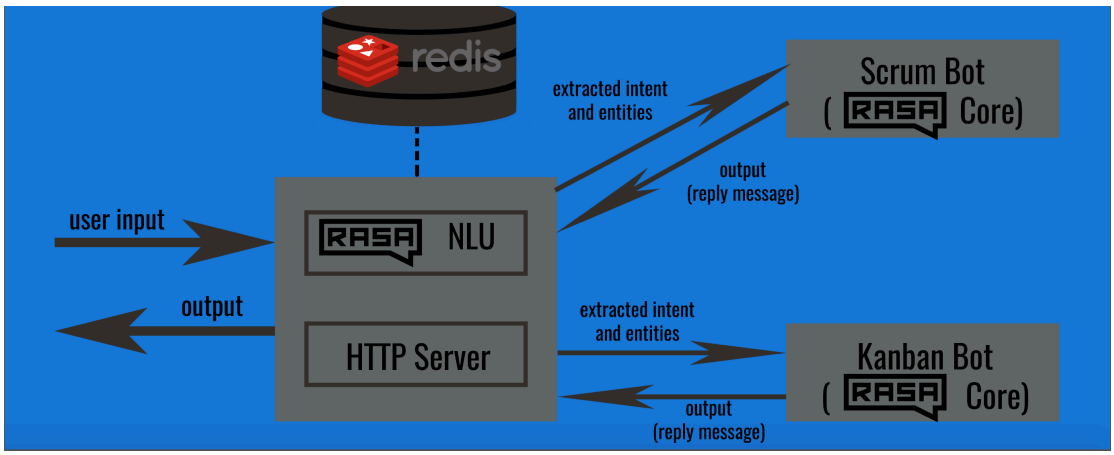


Image d): The overall architecture of the conversational bot project

The development of the conversational bot has comprised a variety of technologies and tools, which can be classified as follows:

- **RASA NLU:** An open source tool used for intent classification and entity extraction. Mainly used for training available data based on probabilistic models.
- **RASA Core:** An open framework that is used in constructing bots. A bot is composed of a domain file, which specifies intents and entities, as well as stories that define the conversation flow of the bot.
- **Amazon Web Services:** Deploying the bot to be used through an API by Actano, the industrial partner of the project.
- **Travis:** Testing code that is committed to the GitHub repository, as well as continuous integration, and deployment.
- **Python:** Programming language that has been used in order to define various features regarding server/client connections (HTTP-service), session handling, and ensuring correct functionality and behavior of the conversational bot.
- **Docker:** Running the conversational bot with a few steps through HTTP-service to deploy and test the bots locally, as well as on Amazon Web Services.
- **Redis DB:** Storing session data, so that the bot can recognize conversation dialogue context.

As a frontend for this conversational bots, Workstreams on Slack was used. The Team of this project has provided Actano with an API, which is capable of conversing and communicating with the bot. However, the Team was not involved in the development scope of this frontend API. Hence, it was not included among the scope of the shown architecture in image d). Nevertheless, as discussed in Section 2c) of the User Manual, it is possible to use the conversational bot through HTTP communication on a web browser.

The conversational bot basically starts working when the user enters a query, asking about a particular scrum or kanban aspect. The first framework that is used to interpret and process that query is the RASA NLU. This NLU's job is extracting the entity and the intent from the entered query, evaluating these units based on the learned training data, and then passes that information to the RASA core.

In the image d), two RASA cores are used for the purpose of serving both conversational bots independently from each other. As each of the RASA cores includes the domain of the allowed and known entities and intents, as well as the stories, which define the sequence of possible actions that the bot can undertake, a response/reply is generated to the user, based on the information that was provided by the NLU. The selection of the output of the Core hence, depends on the output and the response of the NLU with respect to the query that was initially entered from the user. Finally, an output reply is generated from the RASA Core and is then displayed to the user in the form of a JSON response.

Attached to the web server that is operating the HTTP service on which the interaction with the bot is ongoing, is a Redis database. The purpose of this database is storing all the data flow that has occurred within a single session. This helps the bot in understanding the context of the ongoing dialogue and in order to be able to return to the point where a conversation has been previously left (paused conversation), based on the session information that is stored in Redis. More importantly, through the use of Redis, the feature of clearing the history of an ongoing conversation is enabled.

# 4. Used Technologies and Tools

This section gives an overview on all technologies and tools that have been used throughout the development of the conversational bot project. Wherever applicable, it also guides users to reproduce customized options that the developers of the conversational bots have set while using a certain development tool.

## 4a) RASA Framework

RASA is the main framework through which the conversational chatbot dialogues have been implemented. With RASA, users have the ability to adjust dialogues and configure bots about any topic they wish to teach to the bots. The RASA framework consists of the Natural Language Understanding (NLU) component, along the Core component.

The NLU possesses a main role to the success and the relevance of a response that is delivered to a query, as its processing solely depends on how well available data has been trained and interpreted by RASA. In a similar fashion to all systems containing machine learning algorithms, this framework requires as much data as possible to the addressed topics, in order for the bot to be able to learn about topics as much as possible. In addition, the RASA NLU's main function is extracting the intent, that is the action name that is provided from the user's request, and the entity/entities, which involve all keywords that match between the user's query and the training data. In order for the intent and entity extraction to be successful, developers are required to 'chop' all possible categories and scenarios that are possible during dialogue execution, for the purpose of an efficient language processing.

On the other hand, the RASA core is the component that drives the flow of a dialogue of a conversational chatbot. RASA core includes the domain file, where all allowed actions, intents, and entities are located. The bot will not recognize any data entered from the user not matching the domain data. In addition, the RASA core also includes the stories, which are based on a constructed flowchart. The flowchart is graphical diagram, which provides all possible paths from a starting point to an ending point of the dialogue. A flowchart is typically designed by the developers of the bot. Based on a query from the user, the intent, an action name, and the entity, a parameter that user enters, the further flow of a dialogue is decided. The stories of the RASA core list all these possible scenarios during a dialogue flow, so that the RASA core can interpret the next dialogue path to take upon receiving NLU data. Based on that path a response is created and output to the user.

The creation of a new topic dialogue requires the creation of several files. As a first step it is required to set up a folder structure for the new dialogue topic.

- *workstreambot*
  - *data*
    - **<nlu_training_data.json>**
  - **<your_new_dialogue>**
    - **data**
      - **stories.md**
    - **domain.yml**
    - **<action.py>**

**Required:** The name of *your_new_dialogue* is not allowed to consists of any white spaces.
**Required:** Inside *your_new_dialogue* folder a *data* folder must exist.
**Required:** The correct naming of *stories.md* and *domain.yml* is required.
**Note:** Creating a dialogue flow chart to define the inputs of a user and the responses of the service is recommended but not required.

### Adding NLU training data

The NLU training data needs to be a file with the combined training data of all dialogues you want to use. For this you need to create a file (*nlu_training_data.json*) in the location *data/*. If you want to use all dialogues add your data to *data/nlu_training_data_full.json*.

**Note:** Creating a file which contains only *your_new_dialogue* NLU training data is recommended for testing.

Combining all training data of dialogues to use is required because the input given by the user needs to be analysed regarding the topic dialogue it belongs to before even choosing the topic itself. This is done by the Rasa NLU which at the moment is not able to load multiple training files.

A training data sample is composed of text, intent and entities. A more detailed explanation you can find here:
http://nlu.rasa.ai/tutorial.html#preparing-the-training-data

```
{ "text": "show me chinese restaurants", "intent": "restaurant_search", "entities": [
{ "start": 8, "end": 15, "value": "chinese", "entity": "cuisine" } ] }
```

The decision which dialogue the user input belongs to is done on basis of the intent, it distinguishes between two kinds:

- Topic switching intent
- General intent

### Topic switching intent

**Required:** The training data must contain a topic switching intent for *your new dialogue* otherwise the user cannot use it.
A topic switching intent has a specific naming pattern: switch_<your_new_dialogue>
It is important that the name of the folder, your dialogue is located in, is repeated in the topic switching intent.

### General intent

General intents can exist in different dialogues and don't cause topic switching, because of that the naming is not limited by any constraints except the one: They are not allow to begin with switch_, because this could provoke unwanted behavior like topic switching.

### Domain and stories

The creation of domain and stories you can find the documentation:

- Define a domain: https://core.rasa.ai/tutorial_basics.html#define-a-domain
- Define stories: https://core.rasa.ai/tutorial_basics.html#define-stories
- Story - Checkpoints: https://core.rasa.ai/tutorial_basics.html#define-stories

### Customize your response

**Required:** The customization of your response is required because the expected response is JSON format.
By default the responses of the service are defined in the *domain.yml* as templates, to customize these it is necessary to create a python file (*action.py*).

For more information take a look into Defining Custom Actions: https://core.rasa.ai/domains.html#defining-custom-actions
**Note:** The message given to dispatcher needs to be JSON format.

## 4b) Docker

Docker was mainly used in order to facilitate the deployment of the conversational bot project in just few steps, on a local machine, as well as on Amazon Web Services (AWS). In order to use multiple Docker containers for running several dockers, we have used Docker-Compose.

**For Installation:**

- Docker: please refer to: https://docs.docker.com/install/
- Docker-Compose: please refer to: https://docs.docker.com/compose/install/
- Configuring Docker on AWS: please refer to: https://docs.docker.com/docker-cloud/cloud-swarm/link-aws-swarm/

Upon configuring the conversational bots on AWS, **please note**: The team had to make 3 of their ports public, in order to enable the visibility of their API.

***How to dockerize a conversational bot***

~/Desktop/amos-ws17-proj1/workstreambot/docker-compose.yaml

```
version: '3'
services:
  workstreambot:
    container_name: workstreambot
    build: .
    ports:
      - 5012:5000
    expose:
      - 5012
    environment:
      - PYTHONUNBUFFERED=0

networks:
  default:
    external:
      name: redis-network

```

Image e): Structure of a docker-compose.yml file to run the conversational bots

Image e) shows how conversational bots that are located in the workstreambot directory of the project files, are dockerized, so that they can be accessed at port 5000. Furthermore, this dockerization also deploys a Redis database store in order to save session based information.

***How to run the docker***

Assuming that you have already installed Docker, Docker-compose, and have created a docker-compose.yml configuration file of your wish, you are able to run the conversational bots with the following steps:

*Start the Redis database:*

```
# cd to amos-ws17-proj1
bash redis/run_redis.sh
```

*Run docker-compose.yml in 'workstreambot' directory*

```
# cd to workstreambot
docker-compose up
```

Having performed these steps successfully, you now be able to establish a connection between the Workstreamsbot application and the API that has been provided by this project. In the case of the development involved in the scope of this project, as agreed, our industrial partner, Actano GmbH, has implemented the required technical steps to fully achieve the connection of the Workstreams application to the API, and hence, this is not included in this documentation.