BITS Pilani
Pilani Campus

Design of Conversational Experiences

Dr. Bharathi R
Off Campus- CSIS, BITS-Pilani

Slides by :Dr. Shreyas Rao
Associate Prof. (Off Campus), CSIS, BITS-Pilani

BITS Pilani
Pilani Campus

Lecture No. 6

Agenda

- Conversation Scripting
- Designing a Conversation using “RASA” Open Source Tool
  - Intents
  - Entities
  - Utterances
  - Stories
  - Actions

innovate achieve lead

BITS Pilani
Pilani Campus

RASA

RASA

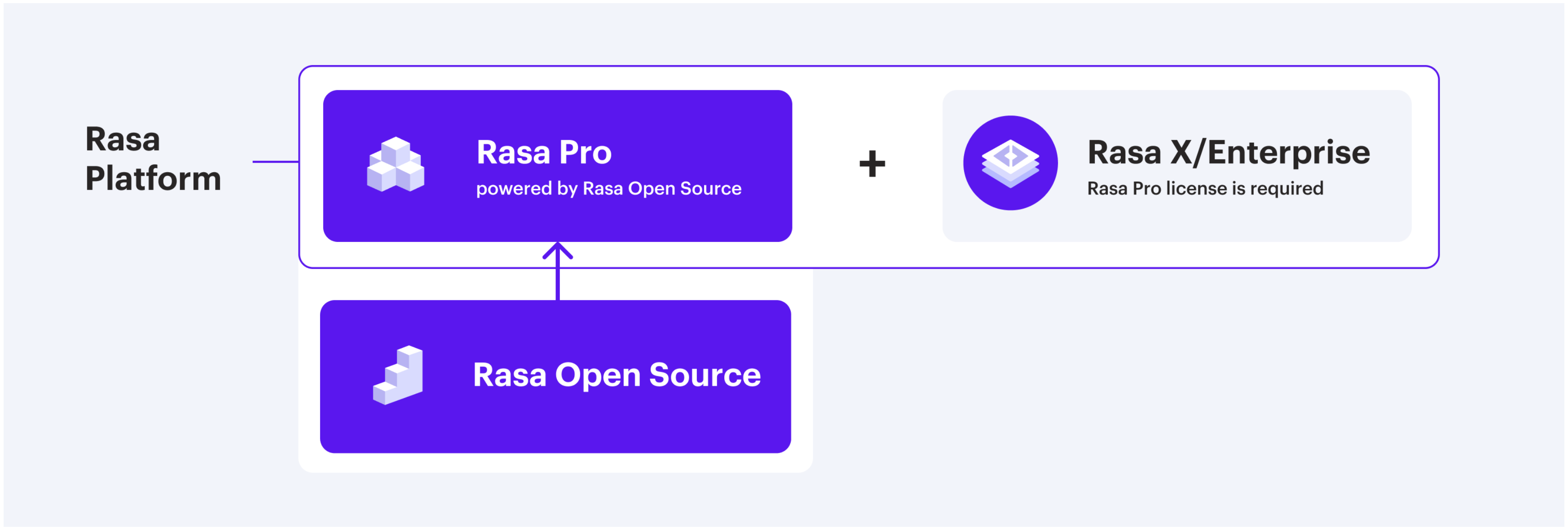
- **Rasa** is an open-source framework for building conversational AI, including chatbots and virtual assistants.
- It is designed to help developers create robust, context-aware conversational experiences that can understand natural language, manage dialogues, and perform actions based on user inputs.

RASA

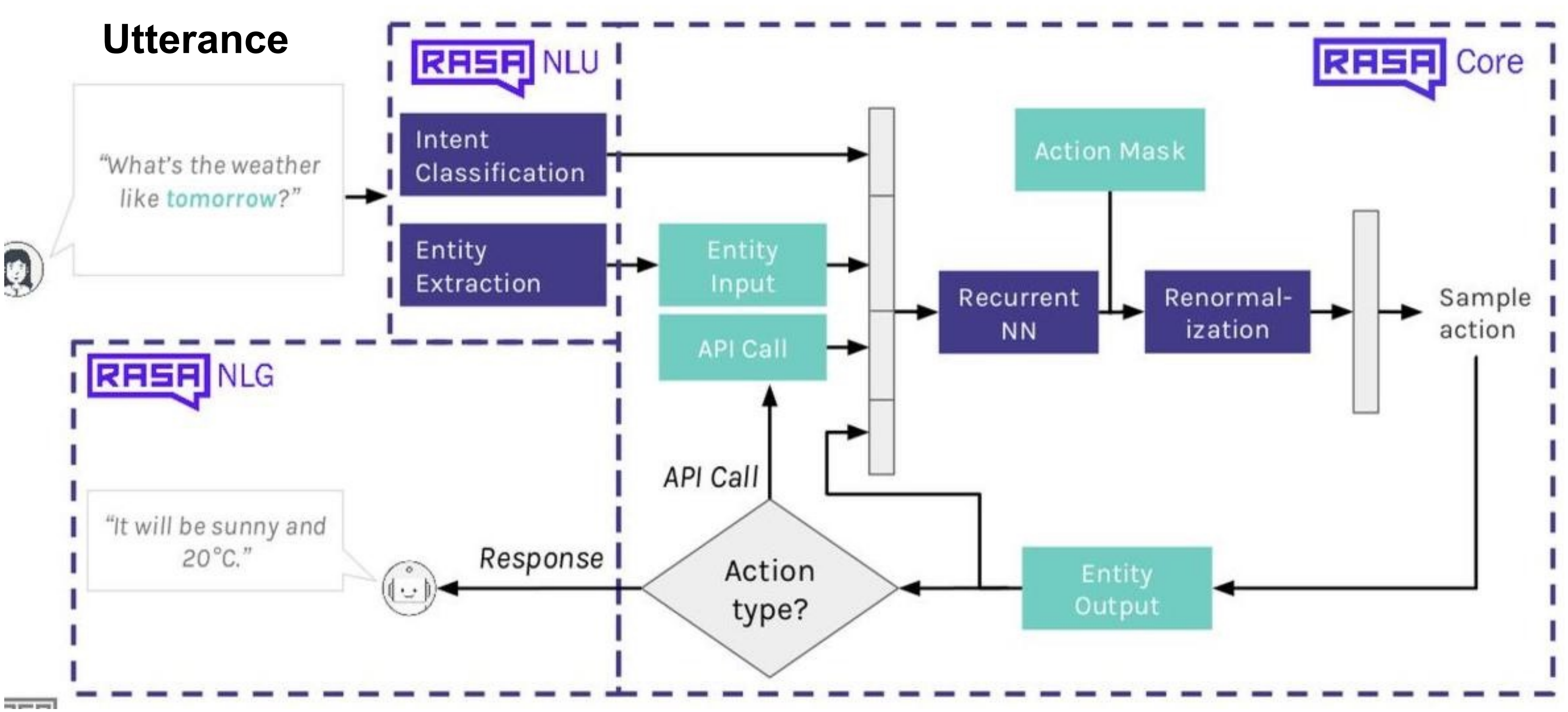
- **Rasa Open Source**: The core framework for building, training, and deploying machine-learning-based chatbots and assistants. It includes tools for natural language understanding (NLU) and dialogue management.

- **Rasa X (Rasa Pro):** A toolset that helps developers improve and manage their conversational AI over time. Rasa X is designed for reviewing conversations, annotating data, and deploying models. It provides a user-friendly interface for developers and non-developers to refine and improve the assistant’s performance continuously.

RASA


Rasa Platform
Rasa Pro
powered by Rasa Open Source
+
Rasa X/Enterprise
Rasa Pro license is required
Rasa Open Source

RASA Architecture



RASA Installation

- **URL -** https://rasa.com/docs/rasa/installation/installing-rasa-open-source/
- Open Command Prompt

➢ d:

➢ mkdir rasa

➢ cd rasa

➢ pip install --upgrade pip

➢ pip install rasa

RASA - Run Basic Bot

- rasa init
- rasa train [train and build the model]
- rasa shell [test the chatbot]
- rasa run --port 5006 OR rasa run [run server; default is localhost:5005]

Optional Commands

- rasa interactive [to interact via web interface]

View the visualizations at URL -> http://localhost:5006/visualization.html

- rasa run actions [If python functions written for actions]

RASA Basics Explained

- You can create a chatbot model using the Rasa framework by providing the necessary files:

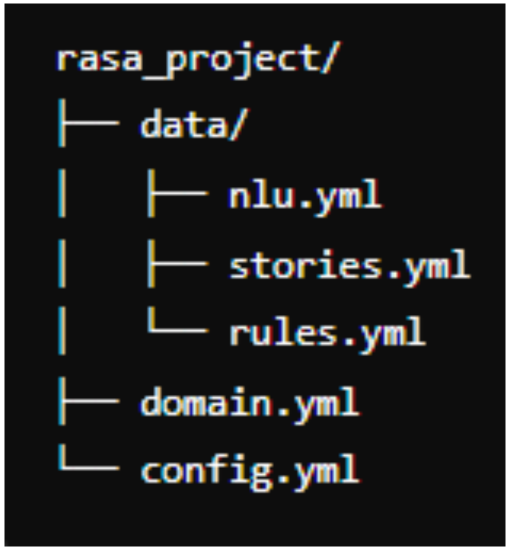
1. nlu.yml (intents)
2. stories.yml (conversation paths – dialogue management)
3. rules.yml (conditional paths)
4. domain.yml (actions, entities, slots, responses).

These files define the essential components needed to train a Rasa model that can understand user inputs, predict intents, and respond appropriately

RASA File Structure

Set Up Rasa Files: Place your configuration files in the correct subdirectories

- data/nlu.yml: Contains the intents and example utterances.
- data/stories.yml: Contains the stories defining the conversation flows.
- data/rules.yml: Contains rules that define specific behaviors.
- domain.yml: Defines the actions, entities, slots, responses, etc.



```
rasa_project/
├── data/
│   ├── nlu.yml
│   ├── stories.yml
│   └── rules.yml
├── domain.yml
└── config.yml
```

Example of **nlu.yml** with Entities

```
version: "3.1"
nlu:
- intent: greet
  examples: |
    - Hi
    - Hello
    - Hey there!

- intent: ask_weather
  examples: |
    - What’s the weather like in [Bangalore](city)?
    - Tell me the forecast for [tomorrow](date).
    - Will it rain in [Delhi](city) next week?
    - How's the weather in [Chennai](city) today?

- intent: book_flight
  examples: |
    - I want to book a flight from [Mumbai](source) to [Dubai](destination).
    - Can you find flights from [New York](source) to [London](destination)?
    - Book me a ticket from [Paris](source) to [Berlin](destination) for [Friday](date).
```

Example **rules.yml**

```
version: "3.1"

rules:
- rule: Greet the user
   steps:
   - intent: greet
   - action: utter_greet

- rule: Say goodbye
  steps:
  - intent:goodbye
  - action: utter_goodbye

- rule: Handle FAQs
  steps:
  - intent: ask_weather
  - action: utter_weather
```

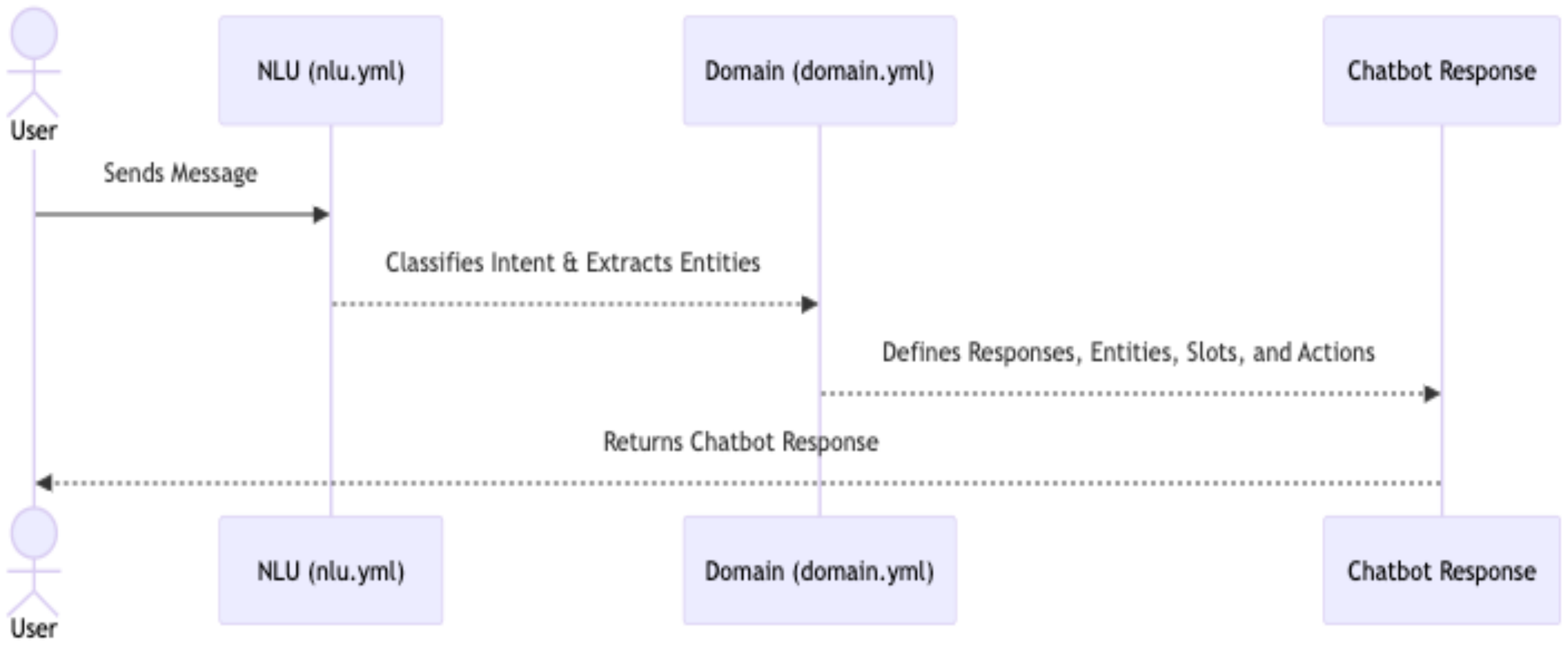
Example **stories.yml**

```
version: "3.1"
stories:
- story: User asks for weather
  steps:
  - intent: greet
  - action: utter_greet
  - intent: ask_weather
  - action: utter_weather
  - intent: goodbye
  - action: utter_goodbye
```

Integration of **nlu.yml** with **domain.yml** in Rasa

The nlu.yml file helps the chatbot **understand user inputs**, while the domain.yml file **defines responses, entities, slots, and actions** that the bot will use.

Integration of **nlu.yml** with **domain.yml** in Rasa



nlu.yml (Understanding User Input)

```
version: "3.1"
nlu:
- intent: ask_weather
  examples: |
    - What’s the weather like in
      [Bangalore](city)?
    - Tell me the forecast for
      [tomorrow](date).
    - Will it rain in [Delhi](city) next
      week?
    - How's the weather in
      [Chennai](city) today?
```

domain.yml (Defining Responses & Entitie

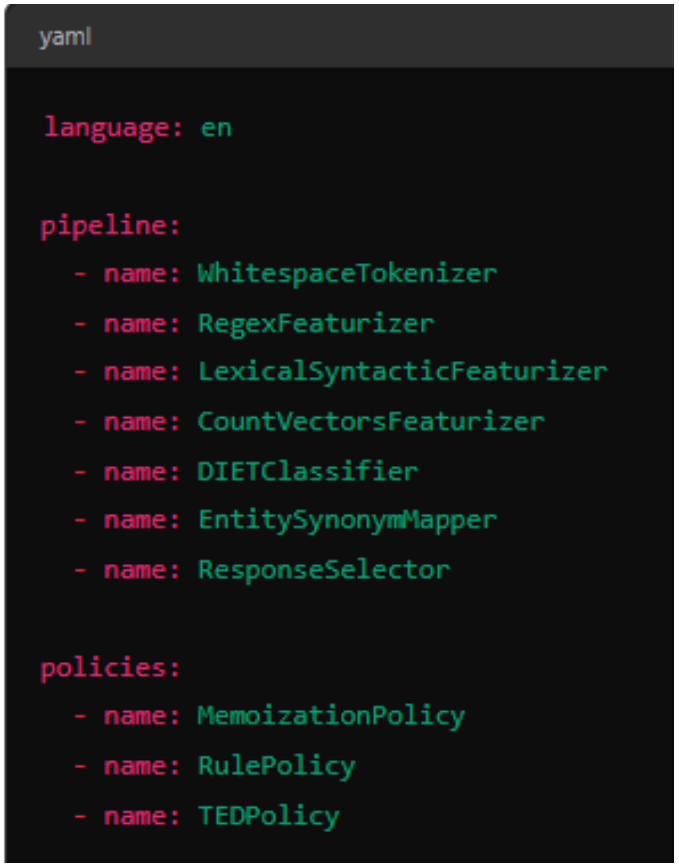
```
version: "3.1"
intents:
  - ask_weather

entities:
  - city
  - date

slots:
  city:
    type: text
  date:
    type: text

responses:
  utter_ask_weather:
    - text: "Fetching weather details for
      {city}..."
```

Comparison Table

File Name	Purpose	Example Use Case
nlu.yml	Defines intents, entities, and training examples	Understanding "What's the weather like?" as an ask_weather intent
rules.yml	Specifies fixed responses for certain intents	Always replying "Hello!" when user says "Hi"
stories.yml	Trains conversation flows	Handling multi-turn conversations (greeting → asking weather → saying goodbye)

Config.yaml

This file is used to define the machine learning pipeline (e.g., tokenizers, featurizers, and policies) that will be used to train your model.



yaml

```yaml
language: en

pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: LexicalSyntacticFeaturizer
  - name: CountVectorsFeaturizer
  - name: DIETClassifier
  - name: EntitySynonymMapper
  - name: ResponseSelector

policies:
  - name: MemoizationPolicy
  - name: RulePolicy
  - name: TEDPolicy
```

RASA train

After setting up your files, you can train the Rasa model by running the following command

- **rasa train**

This command will read all the data from the nlu.yml, stories.yml, rules.yml, and domain.yml files, use the configuration in config.yml, and train a model. The trained model will be saved in the models/ directory.

RASA shell

Test the model

You can test your trained model in interactive mode to ensure it performs as expected.

- **rasa shell**

This will launch an interactive shell where you can type messages, and the model will respond based on the training data.

RASA run

Run the bot

- rasa run [default is localhost:5005] OR rasa run --port 5006

- This command is used to run the Rasa server and expose your assistant's endpoint, making it accessible for external applications or users.

- When you run rasa run, the assistant will start an HTTP server that can listen for incoming messages or requests from clients (such as a web interface, messaging platform, or REST API client).

RASA interactive [Optional]

Interact with the bot

- ➢ rasa interactive

- Provides a way to interact with your assistant in real-time while simultaneously training it.
- It allows you to simulate conversations with your bot, observe how it responds to various inputs, and make adjustments if necessary.
- This method of interactive learning helps refine both the NLU model (understanding intents and entities) and the dialogue management model (handling conversation flows)

RASA run actions [Optional]

Run Actions

- rasa run actions

- Used to run a custom action server for your Rasa assistant.
- This server handles custom actions that are not covered by predefined responses or simple utterances.
- Custom actions allow your assistant to perform complex operations such as database queries, API calls, calculations, or any other Python logic needed to fulfill a user's request.

RASA X (Not free - Subscription)

- **Rasa X** is a companion tool to Rasa Open Source that provides additional features such as:
- Reviewing and annotating conversations.
- Managing training data.
- Sharing your assistant with testers.
- Deploying your bot and integrating it with messaging platforms.

➢ pip install rasa-x --extra-index-url https://pypi.rasa.com/simple

➢ rasa x

This command will start the Rasa X server, which can be accessed via a web browser at http://localhost:5002.

Exercise I

1. Run the default chatbot application loaded with RASA [Do you want to train a model -> YES]
2. Observe the folder structure
3. Understand the stories, intents, utterances, actions
4. Make minor changes to the structure, and re-run the experiment
    a. In the nlu.yaml - add a new intent, with sample utterances.
    b. In the stories.yaml – add a new story
    c. In the domain.yaml – add the intent and response
    d. Train the bot using command “rasa train”
    e. Test the bot using command “rasa shell”

Exercise II – Adding Entities

1. nlu.yaml

```
# Change No. 1
- intent: provide_name
  examples: |
    - My name is [John](user_name)
    - I am [Alice](user_name)
    - You can call me [Michael](user_name)
    - I go by [Sarah](user_name)
    - It's [David](user_name)
```

Exercise II – Adding Entities

2. domain.yaml

```
intents:
  - greet
  - goodbye
  - affirm
  - deny
  - mood_great
  - mood_unhappy
  - bot_challenge
  - provide_name  #Change No. 2
```

Exercise II – Adding Entities

2. domain.yaml

```
# Change No. 3
entities:
  - user_name
```

Exercise II – Adding Entities

2. domain.yaml

```
# Change No. 4
slots:
  user_name:
    type: text
    influence_conversation: true
    mappings:
      - type: from_entity
        entity: user_name
      - type: from_text
```

Exercise II – Adding Entities

2. domain.yaml

```
# Change No. 5
 utter_acknowledge_name:
 - text: "Nice to meet you, {user_name}!"
```

Exercise II – Adding Entities

3. stories.yaml

```
# Change No. 6
- story: capture user name
  steps:
  - intent: greet
  - action: utter_greet
  - intent: provide_name
  - slot_was_set:
    - user_name: "{user_name}"
  - action: utter_acknowledge_name
```

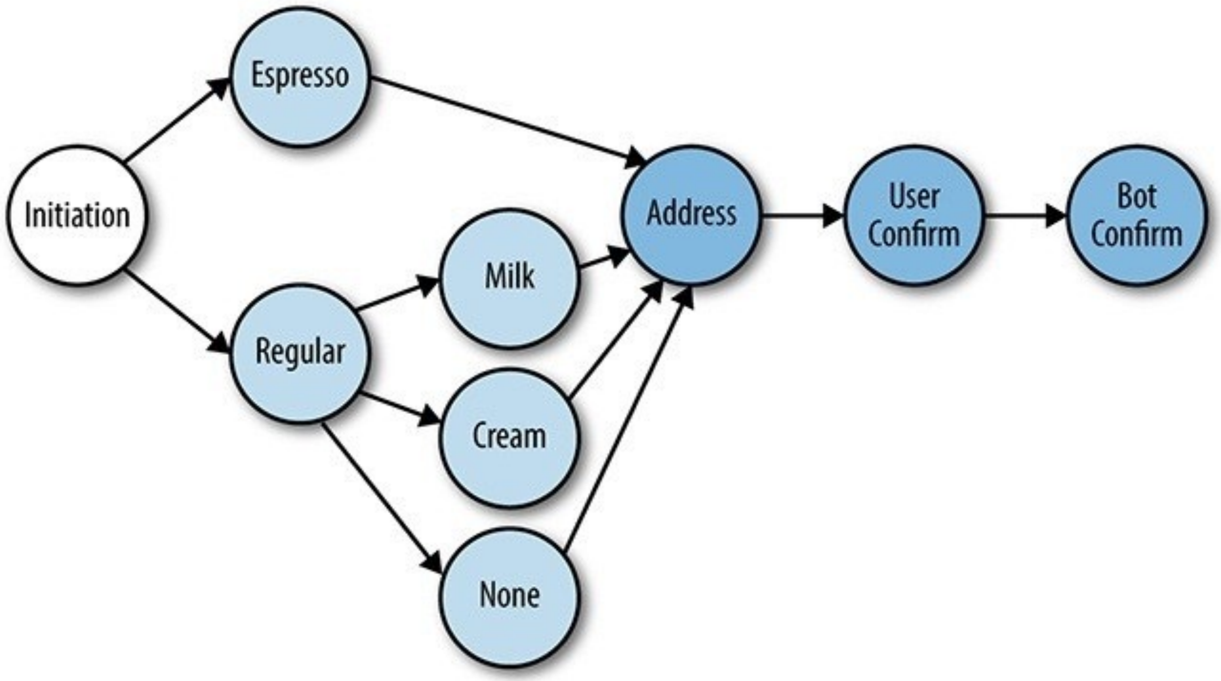
Exercise II

4. Train the Model and run the application

- rasa train
- rasa shell OR
- rasa interactive

Exercise III [15 minutes]

Design the conversation flow for “Coffee Bot” using RASA open source tool



File Structure

- Intents: data/nlu.yml
- Entities: Defined in both data/nlu.yml and domain.yml
- Slots: domain.yml
- Responses: domain.yml
- Stories: data/stories.yml
- Custom Actions: actions.py

Step 1: Create a Blank RASA Project

- d:
- mkdir rasa-coffeebot
- cd rasa-coffeebot
- rasa init [choose to create empty repository]
- Do you want to train a model -> NO

Step 2: Design Intents

- File: nlu.yml
- Location: data/nlu.yml

- This file stores all the intents that your bot should recognize, along with example user inputs for each intent.

Step 3: Design Entities

- Entities are specified within the nlu.yml file using the format shown above. In addition, entities are referenced in the domain.yml file to inform Rasa of which entities it should track.

- File: domain.yml
- Location: domain.yml

Step 4: Design Slots

- File: domain.yml
- Location: domain.yml

- Slots should be added to the domain.yml file, where you define the slots that store contextual information throughout the conversation.

Step 5: Design Responses

- File: domain.yml
- Location: domain.yml

- Responses are defined in the domain.yml file. This file specifies the messages the bot sends in response to certain user inputs or actions.

Step 6: Design Stories

- File: stories.yml
- Location: data/stories.yml

- Stories are defined in the stories.yml file, which describes different conversation paths based on user input.

Step 7: Design Custom Actions [Optional]

- File: actions.py
- Location: actions.py

- If you have custom actions that the bot should perform (e.g., making an API call, database operation), define them in actions.py.

Step 7: Design Custom Actions [Optional]

```
# actions.py

from rasa_sdk import Action
from rasa_sdk.executor import CollectingDispatcher
from rasa_sdk.events import SlotSet

class ActionConfirmOrder(Action):
    def name(self):
        return "action_confirm_order"
```

```
def run(self, dispatcher: CollectingDispatcher, tracker, domain):
        coffee_type = tracker.get_slot("coffee_type")
        extras = tracker.get_slot("extras")
        address = tracker.get_slot("address")

        confirmation_message = (
            f"You have ordered {coffee_type} coffee with {extras}. "
            f"Your delivery address is {address}. Please confirm."
        )
        dispatcher.utter_message(text=confirmation_message)
        return []
```

Steps to Run

- rasa train
- rasa shell

BITS Pilani
Pilani Campus

YAML

YAML

- "Stands for “YAML Ain't Markup Language”
- A human-readable data serialization language that is often used for writing configuration files

YAML format

Comments	Start with the # symbol
Mapping (key-value pairs)	Represented with a colon (:)
Sequences (lists/arrays)	Denoted by hyphens (-) for each item

YAML Examples

Comment:

```yaml
# This is a comment
person:
  name: John Doe  # Inline comment
  age: 30
```

Mapping (key-value pairs):

```yaml
person:
 name: John Doe
 age: 30
 city: Manchester
```

Sequences (lists/arrays):

```yaml
people:
  - John Doe
  - Jane Smith
  - Mary Jones
```

YAML Examples

```yaml
# Web Application Configuration
app:
  name: MyWebApp
  version: 1.2.0
  environment: production
database:
  type: PostgreSQL
  host: localhost
  port: 5432
  username: admin
  password: secret
```

Thank You!