Finite State Machine ChatBot

Dialogue engineering

M2 NLP 2025-2026

This tutorial is inspired from this one, taught at the University of Gothenburg by Vladislav Maraev and Staffan Larsson.

In this exercise, you will work with a finite-state machine (FSM) based dialogue system. Recall that an FSM is a model of computation that consists of states, transitions, and actions. In the context of dialogue systems:

- Each state represents a step in the conversation, such as asking the user for a meeting title or confirming a date.
- Transitions define how the system moves from one state to another, based on user input or automatic actions.
- Actions include things like displaying a message to the user and updating the internal context (e.g. storing the title or date of a meeting).
- Using FSMs allows us to clearly structure dialogues, handle user input in a controlled way, and update the conversation state in a predictable manner.

1. Download the `FSMChatbot` folder on Arche in the section Lab 2. In this folder you will find the three components of your system: the FSM definition, the grammar, and the code.

   - FSM definition (`fsm.py`): This file describe your finite-state machine, e.g. it explicits the different steps the system has to follow and in what order. Each state has a name and contains:
     - A message to display (`say`);
     - Transition(s), each with:
       * The expected entity or control input;
       * The target state (i.e. in what state the system should go after this one);
       * The transition type (`entity` for inputs that update context, `control` for steps that require no update like yes/no).
   - The grammar – `grammar.py`: This file describes the vocabulary of the system. It contains two dictionaries:
     - `task_vocabulary` contains expressions that the system will recognise and associate with a variable in the context. Each entry has:
       * A type, which is the name of the variable we will use in the context (we do not want users to be able to input dates when the name of the meeting is inquired);
       * A list of aliases, which are all the forms the system will recognise and associate with the entry.
     - `control_vocabulary` contains control word which are not associated with variables, like confirmation or refusal tokens (yes/no). Each entry is associated with a list of possible forms the system will recognise and associate with the entry.
   - The code – entity_recognition.py and main.py: Those files contain the core logic of the system:
     - `main.py` is the main program that runs the ChatBot. You should not have to modify it but it is the file you will run.
     - `entity_recognition.py` contains the functions that will match the user's input with the grammar's entries.

   Let's play around with the files to understand the different components:

   (a) Run the file `main.py` and try to answer the bot's questions.

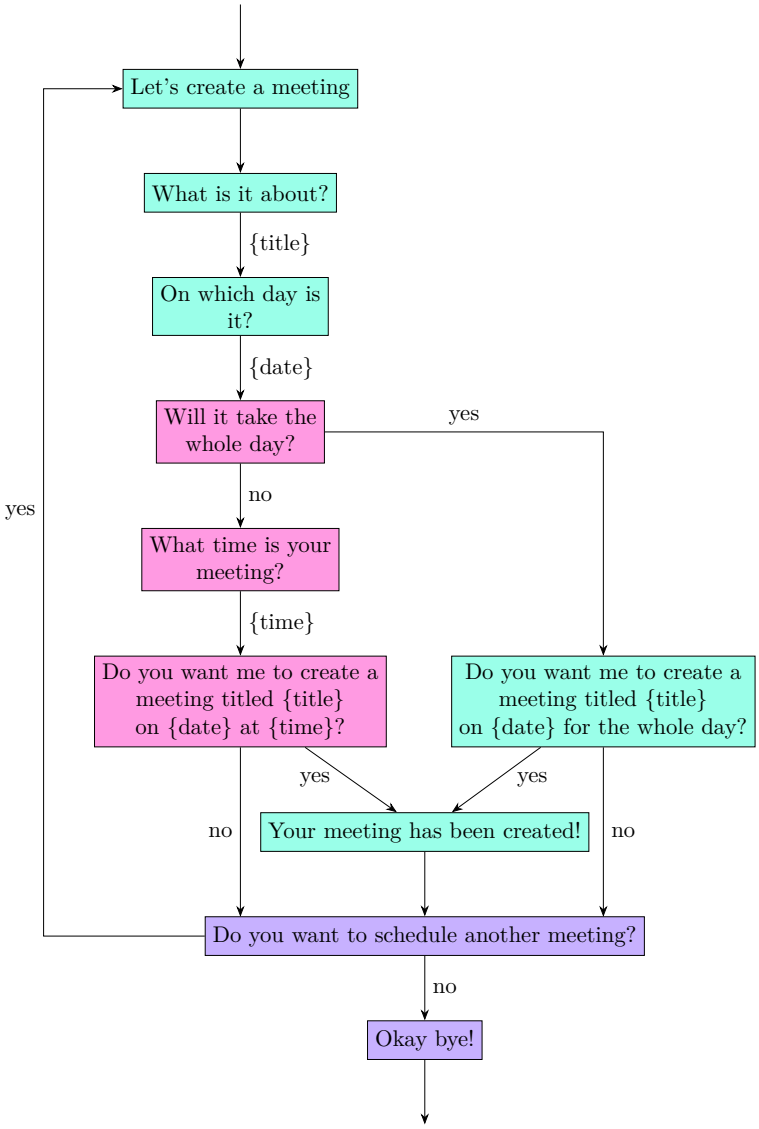
   (b) Open the file `grammar.py` and look at the `task_vocabulary`. Try to answer the bot's questions with the keys and with the aliases. Try to answer the question about the date with a title or vice versa.

   (c) Add a new type of meeting to the grammar (so far you have Dialogue systems lecture and Lunch at the canteen).

   (d) Add more dates to the grammar so that we can create meetings on any day of the week.

2. In its current form our system never stops because there is no `final` state. Create one which the system can enter after having successfully created a meeting or after the user said they did not want to create a meeting.

3. Below is a more complete scheme for a dialogue system that books meetings in your calendar. The original states are in blue and the one added in the previous question in purple. Extend your grammar and FSM description so that your system can follow this pipeline:



4. The pattern-matching we are currently using is not very flexible. Our only leeway right now is to manually write several aliases per entry.
   (a) Think about ways to make the pattern matching more flexible (less susceptible to typos for example) and implement them (you should modify the `match_entity` function in `entity_recognition.py`.
   (b) Increasing the flexibility can make our system more prone to errors. Add a validation state (or several) the model will go through when the pattern-matching was not exact (you may need to modify `main/py` for this question).

5. Think about a second task to include in your model. You can work in small groups to come up with the suitable states/questions or look at existing task-oriented corpora if your task is common enough.
   (a) Draw the related finite-state machine.
   (b) Extend your FSM description file to include this new task.
   (c) Extend the grammar.

6. FSM based dialogue system are quite rigid and brittle because the task must be manually divided into steps the user must follow. Think about some conversational behaviours the model could benefit from and how to implement them.