Multi-agent bot

Tom Rochette <tom.rochette@coreteks.org>

July 24, 2025 — daae079c

0.1 Context

0.2 Learned in this study

0.3 Things to explore

- Better input query format
  - Key/value vs options/arguments
- How dispatcher determines to which agent to dispatch a message
- What response format should be sent back to the dispatcher? To the user?
- What should the user receive as a response?
- Should query processing be asynchronous or synchronous?

1 Overview

The goal of this study is to look into the various ways to construct a network of bots that will interact with users.

A secondary goal is to attempt to create a framework on which developers may offer useful services to their users.

Similar to IFTTT.

2 Requirements

- Bots should be able to
  - communicate with one another
  - synchronize their data with one another (distributed database)
  - find one another (bot discovery/DHT)
- A user should be able to
  - use a bot locally (without internet connection) and yet get useful results
  - send information to the bot
  - query the bot for information
  - interact with the bot through various interfaces (irc, email, slack, web, cli, etc.)
- Only authenticated users may use a bot

3 Query processing

1. A user issues a query

   A query is minimally a string:

   ```
   my query
   ```

However, a query can be very complex as well

```
age =< 36
name = Tom
```

2. An adapter (email, irc, shell, slack, web, etc.) takes the input query and converts it into a standardized format the dispatcher bot expects. The suggested format is similar to the command line arguments and options:

```
my query --age =<36 --name =Tom
weather --city=Montreal
roll --min=1 --max=6
--query="age =< 36 AND name = 'Tom'"
how are you?
what day is it?
```

3. The dispatcher receives the query and dispatches it to any agent that can handle it

    1. The dispatching can either be
        1. to send the received query to all agents
        2. have agents inform the dispatcher what they'd like to be notified about (*this would require a protocol*)

4. All agents **must** reply to the dispatcher, either with an empty response or with a response. The format of the response should be defined by the agent itself (think micro-services).

    An alternative to this specification is that request would have a maximum timeout. If any other agent is unable to reply within the specified timeframe, its reply will be ignored.

    Another possible alternative would be to accept all replies until a second query is formulated. However this would mean that doing multiple queries would either be prohibited or some sort of protocol would have to be defined to do multiple queries at once.

5. The dispatcher sends back the set of responses to the user.

    In some contexts it makes sense to display all the results to the user. However, in some contexts, it does not, for instance when interacting with the bot over irc, we expect to receive a single answer.

    Thus, it would make the most sense to first notify the user that many options are available to him and that he may choose before proceeding.

    In some other cases, we may prefer the bot to act as humans would during a discussion, that is, to start multiple threads of discussion. The user would then be free to engage in the threads he is interested to pursue.

4 Naive architecture

Here I will try to detail an initial architecture by looking at the various components that would be required to make this system work. At first, one will probably recognize the classical flow of an http request/response.

- An `adapter`
    - IRC
    - Slack
    - Email
    - Web (HTTP)
        * API
        * Website with front-end
    - Command line interface

The goal of the adapter is to provide the user with a point of entry to communicate with the bot infrastructure.

- A `query processor`

  The goal of the query processor is to receive a user *command* and convert it into a format that the bot will be able to work with. This generally means converting unformatted commands into keys/values dictionaries (consumable by most web APIs) and/or a CLI arguments/options string.

- A `dispatcher` service

  The goal of the dispatcher is to know about other bots/dispatchers so that he may relay a user query to them and await their response on behalf of the user.

- A `controller`

  The goal of the controller is to act upon a request. If it has to fetch data online or in a database, it may do so. Once the data is acquired, its general next step will be to format/transform it for user consumption.

- A `response formatter/transformer`

  The goal of the response formatter/transformer is to process a response coming from another bot that may not be readily digestible (think of an API resultset for instance).