Circuit Meeting Assistant with the Node SDK

Andreas Stephan

October 9, 2015

Contents

1 Setup

Get a Node JS Server, upload the source and then type: *npm install*. Next you need to install a MYSQL database. Also you need to install cairo, so graphics can be created.

To configure the application, open *'/assistant/config.json'* and fill in the settings. Start the server with *node index.js*

To run the tests, call *npm test*. There will be two errors printed out, but they only came up, because I wasn't able to configure the test suite correctly.

2 Meeting reminder

The meeting reminder offers the possibility to add meetings, show meetings and delete meetings.
If a meeting won't be deleted, you will get a reminding message 5 minutes before the meeting starts.
They have the following names:

1. Add Meeting
2. List Meetings
3. Delete Meetings

2.1 Add Meeting

To add a meeting for which you want to get reminded, you have to use this format:

```
ma: remind DD.MM.YYYY HH:mm
```

You always have to write *ma:*, to tell the assistant, that he is used. Possibly, you could also start with *meeting assistant:* or *assistant:*. Afterr *remind* is a timestamp format given. For example *DD* stands for days.

A few examples how you can give commands:

```
ma: add meeting 25.11.2015 17:23
```

```
ma: rem 25.11.2015 17:23
```

```
ma: rem tomorrow 17:23
```

```
ma: remind in 2 weeks 17:23
```

```
ma: remind in 18 days 17:23
```

You can also tell the meeting assistant that he shall give you a .ics file for the given date like this:

```
ma: remind tomorrow 17:23 ics
```

3 Text statistics

By adding the meeting assistant to the conversation, he will follow text and make statistics.
You can get a .png file, which is showing general statistics of your conversation with the command

```
ma: statistics
```

If you want to have a greater overview you have to go to the website of the server.

4 Feedback(mini talking engine)

If you want to talk with the meeting assistant, you can keywords and special answers to them. By this, you could simulate a few ask/answer sequences.

First open the the *parsingOptions.json* file in */assistant/files*. To add a new category go to *services* → *feedback* and add a structure looking like this:

{"category name" : {"keywords" : ["keyword1","keyword2","keyword3","keyword4",]}}

Now the assistant recoginzes these words. To specify an answer open */assistant/files/messages.json*, and add a struct to the *feedback* entry looking like this:

{"category name" : {"text" : "example text, which will be sent to the user"}}

Of course the category names have to be equal!

5 Modifying the application

In this section I will tell you firstly how to modify the behaviour of the meeting assistant and secondly how to add functionality to the meeting assistant.

5.1 Modifying the text parsing

In the meeting assistant folder, you find a folder files, where a document named *parsingOptions.json* is located.
In this file, you can modify every keyword array.This are the words the parser is searching for.

For example by adding "friend:" to the "assistantKeywords", you can now speak with the assistant by starting your command with "friend:"
The changes will be effective after restarting the server.

5.2 Modifying the output text

Again there's a file called *messages.json* in the 'assistant/files' directory, where each message text is saved.
If you want to modify the texts, change them there.
Again, the changes will be done after restarting the server.

6 Add new service

To add a new service, you need to do the following things:

1. Detect usage of the service
2. Start and build service logic

6.1 Detection

To add detection, you need to change the optionParser, which is located under *'/assistant/services/optionParser'*.
Add a new file to the folder with the name of your service. This js-file needs to have a *parse(text)* function. The parameter *text* is the command from the user. This function must return an object of the following format:

{
"isInUse": true/false,
"writtenOptionsWrong",
(further options for usage of the service..)
}

If *isInUse* is set to true, the service shall be used. If more than one service is in use, the optionParser returns an error.
If *writtenOptionsWrong* is set to true, the optionParser will return the usage of another service, if another service is in use, or will return that the user has to be informed, that his command format was wrong.

The you have to modify *optionParser.js* by adding your parser to *service* object, located in line 52.

6.2 Service logic

First of all you should add a new service module by adding a folder to *'/assistant/services/'*. Here you can build your service logic.
In *'/assistant/utils'* you will find a few helper classes, like the database and the application communicator to build your service. To be consistent make your masseges generic and add them to *'/assistant/files/messages.json'*.

In *'/assistant/main.js'* is a *parseItem()* function, which gets called when a text item is added. There you have to trigger your service logic.