Dokumentation Mycroft-Calendar-Skill

Inhalt

Aufgabenstellung

Die Aufgabe des Mycroft-Projekts war die Erstellung eines Calendar-Skills, der den nächsten Eintrag bzw. Termin des verbundenen NextCloud-Kalenders über Sprache ausgibt. Die Interaktion zwischen dem User und Mycroft sollte dem folgenden Beispiel ähneln:

> *User: „Hey Mycroft, what's my next appointment?*
>
> *Mycroft: "Your next appointment is 'cinema' on June 22nd, 2022 at 4pm"*

Daneben sollten zusätzliche Funktionen umgesetzt werden, die

- die Erstellung eines neuen Kalendereintrags
- das Abfragen von Terminen an einem bestimmten Tag
- das Ausgeben der n-nächsten Termine
- das Löschen eines Termins
- das Umbenennen eines Termins

ermöglichen.

Organisatorisches

Für die Umsetzung des Calendar-Skills stand folgende Hardware zur Verfügung:

- LABIST Starter Kit Raspberry Pi 4
- USB-Lautsprecher Mifa
- Logitech c270 Webcam (als Mikrofon)

Dabei war die SD-Karte des Raspberry Pis bereits mit einem auf Picroft basierten Image bespielt, welches Coqui STT, einen Speech-To-Text Service und SI Mycroft enthielt.

Daneben erforderte das Projekt Grundkenntnisse in der Programmiersprache Python, im Umgang mit dem Versionierungstool Git sowie zum Linux-Betriebssystem.

Aufsetzen des Projekts

Für das Mycroft-Projekt wurde uns ein Basisprojekt unter dem Link

Heisler Marcel / SiCalendarSkill · GitLab (hdm-stuttgart.de)

zur Verfügung gestellt, welches wir forkten und auf diesem aufbauten. Das Basisprojekt enthielt bereits das Skill Template, welches alle benötigten Dateien wie __init__.py, manifest.yml etc. beinhaltet. Außerdem empfanden wir auch die offizielle Anleitung zur Skill-Erstellung von Mycroft-AI (Your First Skill - Mycroft AI (gitbook.io)) als sehr hilfreich, da hier auch die Funktionsweise eines Skills

ausführlich beschrieben wird. Sollte das Forken des obigen verlinkten GitLab-Repositorys nicht möglich sein, kann auch durch das interaktive Skript, welches durch den Befehl ***mycroft-msk create*** ausgeführt wird, ein eigenes Template erstellt werden. Dieses Template wird ebenso Schritt für Schritt in der Anleitung aufgeführt.

Bevor wir uns mit den Skill relevanten Themen wie Dialogs, Intents etc. beschäftigt haben (welche später erläutert werden), haben wir uns zunächst mit der Schnittstelle zwischen dem NextCloud-Kalender und dem Skill bzw. der dahinter liegenden Logik befasst. Dafür waren die Bibliotheken Caldav und iCalendar von zentraler Bedeutung, da diese die Verbindung zum NextCloud Kalender ermöglichen.

Erste Schritte der CalDav-Implementierung

Für die Implementierung der Logik erstellten wir eine eigene Klasse, CalDavCalendar, welche die benötigten Methoden wie bspw. den Login in den NextCloud-Calendar, das Parsen der Termine etc. enthält. Da wir zunächst die eigentliche Verbindung mit Mycroft außen vorließen, war es notwendig, den Usernamen und das Passwort für den NextCloud-Account, der verbunden werden sollte, in den Run-Konfigurationen zu setzen. Dadurch war es möglich, die Funktionen zu testen, bevor wir überhaupt die Ausgabe in Form von Dialogen über den Pi erstellt hatten.

Demnach befassten wir uns zunächst mit der Verbindung zu NextCloud und den zum Account zugehörigen Kalendern, welche mithilfe von CalDav umgesetzt wurde. Die zugehörigen Methoden sind

Create_client (self, url, user_name, password)

Fetch_calendars (self, client)

Anschließend lag unser Fokus auf der Basisaufgabe, der Abfrage des nächsten Termins. Hierfür ist die Methode

Fetch_events (self, start_time, end_time)

zentral, da diese in den anderen Methoden ebenfalls verwendet wird. Diese Methode erwartet eine Start- und eine Endzeit und gibt die vorhandenen Termine in diesem Zeitraum zurück. Bei den Daten handelt es sich um sogenannte .ical Strings, welche anschließend so verarbeitet werden, dass wichtige Informationen wie bspw. der Name des Termins, das Datum sowie Start- und Endzeit ausgelesen werden können. Diese Methode wurde im Laufe der Zeit erweitert, worauf später noch eingegangen wird.

Nachdem wir diese ersten Schritte umgesetzt hatten, teilten wir die Aufgaben für die verschiedenen Funktionen, die unser Skill bereitstellen sollte, auf, sodass eine Person für die Erstellung eines neuen Termins, die nächste für das Umbenennen eines Termins etc. verantwortlich war. Zwischenzeitlich hatten wir uns auch mit der Usereingabe und der Dialogausgabe des Skills beschäftigt.

Erste Schritte der __init__.py -Implementierung

Damit Mycroft auf spezifischen User-Input wie beispielsweise „What's my next appointment?" mit der Ausführung der implementierten Methoden reagiert, sind, wie bereits erwähnt, .intent- und .dialog Dateien nötig.

- .intent-Dateien
    - In diesen Dateien ist der User-Input definiert. Demnach haben wir bspw. eine „calendar.si.next.appointment.intent" Datei erstellt, die exakt den Satz „What's my next appointment?" beinhaltet.
- .dialog-Dateien
    - Diese Dateien beinhalten die Antwortmöglichkeiten von Mycroft. Ein Beispiel hierfür ist „Your next appointment (…)".

In diesen beiden Dateien ist es zudem möglich, mehrere Wörter als Auswahlmöglichkeit mithilfe von runden Klammern „()" und „|" darzustellen. Auch können Platzhalter bzw. Variablen mithilfe von geschweiften Klammern „{}" definiert werden und anschließend im Code verwendet werden.

What (is|are) my next (appointment|event)

Your next appointment is {event}

Auf Basis dieser intent- und dialog-Dateien haben wir in der __init__.py mehrere file-Handler definiert, bei welchen jeweils auf einen bestimmten User-Input reagiert wird. Dafür werden diese file-Handler mit der Annotation „@intent_file_handler" versehen, in der auch die jeweilige Intent-Datei verlinkt ist. Innerhalb der Handler werden die Methoden, die in der caldav_code.py implementiert sind, aufgerufen und deren Output als Variablen zusammen mit den Dialog-Dateien ausgegeben. Die zu schreibenden Dialoge haben wir ebenfalls in unserer Gruppe aufgeteilt, sodass die Teammitglieder, welche die zugehörige Logik implementiert hatten, auch die Integrierung in der Mycroft-Schnittstelle umsetzten.

Schwierigkeiten während der Entwicklung und Learnings

Während des Entwicklungsprozesses stießen wir im Umgang mit der Verknüpfung zwischen dem von uns in der caldav_code.py geschriebenen Code und den definierten file-Handlern auf einige Probleme. Da sich diese Schwierigkeiten im Gesamten auf unser Vorgehen bei der Implementierung der verschiedenen Dialogprozesse ausgewirkt haben, werden sie zunächst genauer erläutert, bevor die einzelnen Methoden und file-handler ausführlicher beschrieben werden.

Problem: Umgang mit Zeitzonen

Zeitstempel beim Parsen der ICal Strings immer zwei Stunden hinterher

Zu Beginn der Entwicklungsphase hatten wir das Problem, dass der Start- und der Endzeitpunkt eines Events beim Parsen falsch interpretiert wurde. Die Zeit wurde in der Textausgabe immer zwei Stunden

hinterher angegeben. Um dieses Problem zu lösen, sahen wir uns die ICal Dateien eines Termins genauer an.

Ursache

Grund für den Fehler war, dass die Start- und Endzeitpunkte der Termine in der grafischen Oberfläche (GUI) des NextCloud Kalenders, zwar als Zeitzone „automatisch (Europe/Berlin)“ angegeben werden, die Information dieser Zeitzone jedoch nicht in dem ICal String des gefetchten Events über den Caldav Code zu finden ist.

Das ist der exportierte ICal String eines über die GUI angelegten Events:

```
BEGIN:VCALENDAR
PRODID:-//IDN nextcloud.com//Calendar app 3.2.2//EN
CALSCALE:GREGORIAN
VERSION:2.0
BEGIN:VEVENT
CREATED:20220614T160414Z
DTSTAMP:20220614T160423Z
LAST-MODIFIED:20220614T160423Z
SEQUENCE:2
UID:61da059e-234c-4d71-8548-9aeaab0e6d10
DTSTART;TZID=Europe/Berlin:20220609T100000
DTEND;TZID=Europe/Berlin:20220609T110000
STATUS:CONFIRMED
SUMMARY:Test
END:VEVENT
BEGIN:VTIMEZONE
TZID:Europe/Berlin
BEGIN:DAYLIGHT
TZOFFSETFROM:+0100
TZOFFSETTO:+0200
TZNAME:CEST
DTSTART:19700329T020000
RRULE:FREQ=YEARLY;BYMONTH=3;BYDAY=-1SU
END:DAYLIGHT
BEGIN:STANDARD
TZOFFSETFROM:+0200
TZOFFSETTO:+0100
TZNAME:CET
DTSTART:19701025T030000
RRULE:FREQ=YEARLY;BYMONTH=10;BYDAY=-1SU
END:STANDARD
END:VTIMEZONE
END:VCALENDAR
```

Die wichtigen Zeitstempel sind rot markiert. An diesen Zeitstempeln ist interessant, dass sie vorangestellt zwar die Information „TZID:Europe/Berlin“ enthalten, diese jedoch nicht in den nachfolgenden Zeitstempel „20220609T100000“ integriert ist. Da in der Methode `get_title_and_time_of_events()` der caldav_code.py die Zeitstempel nur über das reine Datum geparsed werden, bestand zu Beginn der Fehler darin, die Zeitzonen- Informationen zu verlieren. Da die Zeitzone (Europe/Berlin) einen Offset von UTC +2 hat, wurden die gefetchten Zeitstempel immer mit zwei Stunden Abzug angezeigt.

```
for line in event.data.split('\n'):
    if "DTSTART" in line:
        event.start = line.split(":", 1)[1].strip()
    elif "DTEND" in line:
        event.end = line.split(":", 1)[1].strip()
    elif "SUMMARY" in line:
        event.summary = line.split(":", 1)[1].strip()
```

get_title_and_time_of_events

Lösung

Im Code wurde das Problem gelöst, indem in der Methode `generate_output_date_string()` der caldav_code.py der entsprechende time_offset für die gewünschte Zeitzone darauf gerechnet oder abgezogen werden kann, je nachdem ob sich dieser mit einer positiven oder negativen Abweichung zur UTC verhält. Damit wurde das Ziel erreicht, die richtigen Zeitstempel aus den abgerufenen Event-Dateien auszulesen.

```
if tmp_start == date_start_datetime:
    response_string = time.strftime(
        '%dth of %B', time.localtime(time.mktime(date_start) + 60 * 60 * time_offset))
    response_string = self.check_ordinal(response_string)
    event.date_response = f'on {date_start_datetime.strftime("%A")}, {response_string}'
    event.time = None
else:
    response_string = time.strftime(
        '%dth of %B', time.localtime(time.mktime(date_start)))
    start_time = time.strftime(
        'from %I:%M%p', time.localtime(time.mktime(date_start) + 60 * 60 * time_offset))
    end_time = time.strftime(
        ' to %I:%M%p', time.localtime(time.mktime(date_end) + 60 * 60 * time_offset))
    response_string = self.check_ordinal(response_string)
    event.date_response = f'on {date_start_datetime.strftime("%A")}, {response_string}'
    event.time = start_time + end_time
```

generate_output_date_string

Zeitstempel beim Anlegen der Events über den Code zwei Stunden voraus

Als direkte Folge des zuvor beschriebenen Problems des falschen Auslesens der Zeitstempel für die über die GUI angelegten Events ergab sich das weitere Problem, dass es nun nicht mehr möglich war, die über den Code angelegten Events richtig auszulesen.

Ursache

Die Ursache dieses Problems ließ sich ebenfalls über den Vergleich der exportierten ICal Dateien eines über die GUI angelegten Events und eines über den Code erstellten Events feststellen. Vergleicht man die beiden folgenden Auszüge der verschiedenen ICal Dateien, fällt auf, dass die Zeitstempel verschieden gesetzt wurden. Das Problem daran ist, dass diese somit mit Zeitzonen- Informationen angelegt wurden. Beim Fetchen der Events wurden deswegen auf die sowieso schon mit timeoffset

generierten Zeitstempel erneut zwei Stunden addiert. Dadurch wurden sie nun mit zwei Stunden Zeitverzug ausgegeben.

```
BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//Sabre//Sabre VObject 4.3.0//EN
BEGIN:VEVENT
UID:0f7d700e7f3cc6db647dbfc42f005529eace9d39
DTSTART:20220601T110000Z
DTEND:20220601T150000Z
DTSTAMP:20220531T072504Z
SUMMARY:Test
END:VEVENT
END:VCALENDAR
```

1. *Version ICal-Datei*

```
BEGIN:VCALENDAR
PRODID:-//IDN nextcloud.com//Calendar app 3.2.2//EN
CALSCALE:GREGORIAN
VERSION:2.0
BEGIN:VEVENT
CREATED:20220614T160414Z
DTSTAMP:20220614T160423Z
LAST-MODIFIED:20220614T160423Z
SEQUENCE:2
UID:61da059e-234c-4d71-8548-9aeaab0e6d10
DTSTART;TZID=Europe/Berlin:20220609T100000
DTEND;TZID=Europe/Berlin:20220609T110000
STATUS:CONFIRMED
SUMMARY:Test
END:VEVENT
```

2. Version ICal Datei

Lösung

Das Problem wurde gelöst, indem wir die übergebenen Zeitstempel im Code ohne Angabe der Zeitzone übergeben haben, um das Event im NextCloud Kalender anzulegen. Durch den individuell setzbaren Time-Offset beim Auslesen der gefetchten Events wird die Information der gewünschten Zeitzone erst an dieser Stelle benötigt und nicht wie von uns gedacht beim Anlegen des Events.

Problem: Umgang mit Caldav Event Objekten und Mycroft Sprachausgaben

Während der Entwicklung stellte sich die Frage, wie wir am besten mit der Kombination an gefetchten Caldav Event Objekten und den zu implementierenden Antwortmöglichkeiten von Mycroft umgehen sollen. Problematisch war beispielsweise, ein Caldav Event in „schöne“ Textformate umzuwandeln,

welche in die Sprachausgabe einfließen können und gleichzeitig das tatsächliche Eventobjekt nicht zu verlieren, welches im NextCloud Kalender verändert werden sollte.

Ursache

Die Ursache hierfür war, dass wir zu Beginn eine eigene Parsed-Event-Klasse angelegt hatten, um diese Objekte in den Dialogen nutzen zu können. Diese Objekte enthielten die geparsten Ausgaben in der Instanz, was jedoch nicht ausreichte.

Lösung

Wir entschieden uns, die Caldav Methoden grundsätzlich so zu designen, dass sowohl eine Liste an Caldav Event Objekten als auch der geparsten Event Objekte übergeben werden. Hierbei nutzten wir die Möglichkeit Pythons aus, dass mehrere Werte zurückgegeben werden können. Im Fall einer Umsortierung der Termine wurde eine gekoppelte Sortierung vorgenommen, da nur nach dem Startdatum der Events sortiert werden konnte.

Beispiel einer Anwendung in der Caldav Methode fetch_events():

```
events = events_fetched
events = self.get_title_and_time_of_events(events)
parsed_events = self.create_parsed_date_objects(events, 2)
sorted_pairs = sorted(zip(parsed_events, events), key=lambda i: datetime.strptime(i[0].start, '%Y%m%dT%H%M%SZ'),
                      reverse=reverse_sorted)
tuples = zip(*sorted_pairs)
if len(sorted_pairs) > 0:
    parsed_events, events = [list(tuple) for tuple in tuples]

    self.log.info(f"{str(len(events))} events fetched")
    return parsed_events, events
```

fetch_events()

Problem: Ganztägige Events (andere Datumsangaben)

Ein weiteres Hindernis während der Entwicklung war, dass es zunächst nicht möglich war, die Zeitstempel einheitlich in Python Datetime Objekte zu parsen. Es kam immer wieder zu Fehlermeldungen, dass die Zeitstempel nicht im richtigen Format vorliegen würden.

Ursache

Der Grund für das beschriebene Problem lag darin, dass je nach Art des Events (ob es ein ganztägiges Event ist oder nicht) ein unterschiedliches Zeitstempel Format vorliegt. Dies ist im Folgenden dargestellt:

Zeitstempel eines ICal String für ein ganztägiges Event:
DTSTART;VALUE=DATE:20220601
DTEND;VALUE=DATE:20220602

Zeitstempel eines ICal String für ein nicht ganztägiges Event:

DTSTART;TZID=Europe/Berlin:20220609T100000
DTEND;TZID=Europe/Berlin:20220609T110000

Lösung

Das Problem wurde gelöst, indem in der Methode `parse_dates()` überprüft wird, welche Länge der erhaltene Zeitstempel bei Abfrage eines Events aus dem NextCloud Kalender hat.

Entspricht die Länge der Anzahl acht, so handelt es sich um die Zeitangabe eines ganztägigen Events, welche noch um eine Zeitangabe ergänzt werden muss, um beispielsweise eine Sortierung zu ermöglichen.

```
def parse_dates(self, events):
    """
    Formats each begin and end string in the event object to a consistent format.
    Full-day-events still need a given time to be able to sort
    :param events: list of events
    :return list of events with formatted begin and end date
    """
    for event in events:
        if (len(event.start) == 8) and (len(event.end) == 8):
            event.start = time.strftime('%Y%m%dT%H%M%SZ', time.strptime(event.start, '%Y%m%d'))
            event.end = time.strftime('%Y%m%dT%H%M%SZ', time.strptime(event.end, '%Y%m%d'))
            print(event.start)
            print(event.end)
    return events
```

parse_dates

Problem: ICal String Formatierung

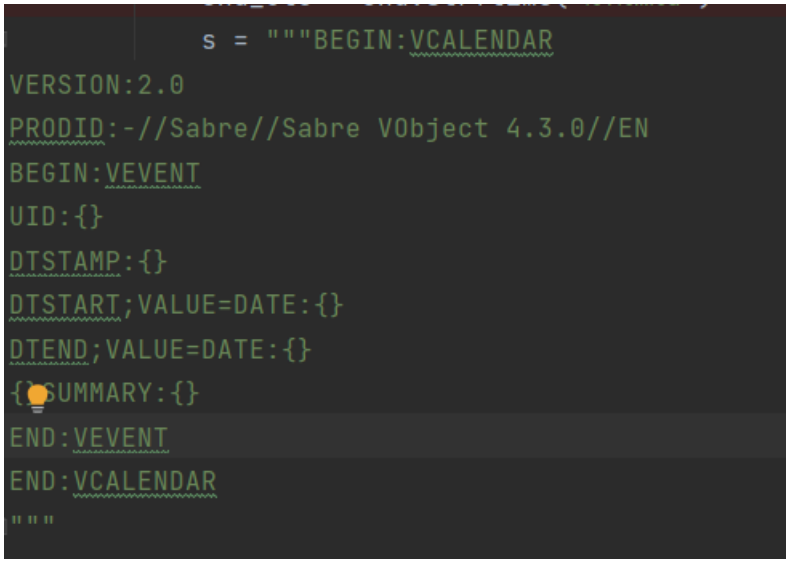
Eine weitere Problematik war die Tatsache, dass es zu Beginn nicht möglich war, ein Event über den Code anzulegen. Es kam zu der Fehlermeldung „END:VCALENDAR was never closed“.

Ursache

Die Ursache des Problems ließ sich darüber erklären, dass wir den ICal String nicht richtig eingerückt hatten. Somit wurde nicht erkannt, dass das Attribut VCALENDAR korrekt geschlossen wurde. Dieser Fehler war schwierig aufzuklären, da es keine gute Dokumentation zum Umgang mit ICal Strings und CalDav gab.

Lösung

Die Lösung war es, den ICal String im Code auf eine ganz bestimmte Weise einzurücken, welche zwar nicht der eigentlichen Formatierung einer Python Datei entspricht, es aber ermöglicht die Events über den Code korrekt anzulegen.



ICal String Einrückung

Problem: Bei remove wurde in die falsche Richtung sortiert

Nachdem die Sortierung mithilfe des Parsen funktionierte, entstand das Problem, dass, wenn man das letzte Event mit `remove_last_n_events(1)` löschen wollte, immer das erste Event aus dem Kalender gelöscht wurde; demnach das am weitesten in der Vergangenheit liegende.

Ursache

Die Ursache dieses Problems lag darin, dass in `fetch_events()` immer so sortiert wurde, dass das startDate der Events aufsteigt. Von diesem sortierten Array wurden die ersten n-Elemente zurückgegeben. Im Falle von `remove_last_n_events(1)` wurde also das erste Element zurückgegeben, also das Element, das am weitesten zurücklag.

Lösung

Die Lösung für dieses Problem war relativ leicht, da die Sortiermethode einen Parameter `reverse` hatte. Dieser Parameter war standardmäßig auf „false“ gesetzt. Zur Lösung haben wir diesen Parameter auf „true“ gesetzt, falls vergangene Events aus NextCloud geholt werden.

Problem: Bei der Mycroft Methode ask_yesno() kann nur mit Spracheingabe gearbeitet werden

Während der Entwicklungsphase testeten wir unsere Skills hauptsächlich ohne Spracheingabe über das Mikrofon und triggerten die einzelnen Intents über die Tastatureingabe, da diese Vorgehensweise schneller war. Dabei fiel jedoch auf, wenn Mycroft den User nach einer Confirmation fragt, erwartet

er demnach als Antwort entweder „Yes“ oder „No“. Über die Tastatur, wurdendiese Eingaben meist jedoch nicht erkannt. Oft antwortete Mycroft mit „Sorry, I didn’t catch that“, wodurch das Testen des eigentlichen Skills besonders zeitaufwendig wurde. Letztendlich haben wir keine Ursache für dieses Problem gefunden.

Lösung

Um das Problem zu umgehen, haben wir als Confirmation-Möglichkeit zusätzlich „I confirm“ als Eingabe implementiert, welches die gleiche Funktion wie „Yes“ hat.

Problem: Allgemeine Schwierigkeiten

Insgesamt zeigten sich während der Entwicklungsphase einige allgemeine nicht Code spezifische Probleme, welche die Entwicklungsgeschwindigkeit deutlich verlangsamten. Ein häufiges Problem war, dass auf dem Raspberry Pi nicht der neueste Stand des GitLab-Repo gepulled werden konnte, da das GitLab nicht erreichbar war. Somit konnte öfter nicht der aktuelle Stand der Anwendung getestet werden und es ließ sich nur vermuten, ob die Entwicklung auf dem richtigen Weg war. Dieses Problem wirkte sich ebenfalls auf die Möglichkeit aus, Code generell auf einem Branch im Repo upzudaten und mit den anderen Teammitgliedern teilen zu können.

Ein weiteres Problem bestand darin, dass die NextCloud Seite häufig sehr langsam geladen oder sich aufgehängt hat. Dadaurch war es schwerfälliger zu überprüfen, ob Events wirklich angelegt, gelöscht oder geupdatet wurden.

Ebenfalls war es oft schwierig, effektiv mit dem Raspberry Pi in der Hochschule während der Vorlesungszeit zu arbeiten. Es trat das Problem auf, dass der Pi sich nicht über das LAN Kabel mit dem Internet verbinden ließ. Auch kam es häufig zu der Problematik, dass die Tastatur während des Ausführens von mycroft-cli-client nicht erkannt wurde und deshalb häufige Reboots erfolgen mussten. Dies war sehr zeitaufwändig.

Implementierte Skills

Methoden & Methodenbeschreibungen

Create_client

Erstellt mithilfe der übergebenen Caldav-Url, Username und Password einen DAVClient, welcher die Verbindung zum Kalender ermöglicht.

Fetch_calendars

Fragt alle Kalender des eingeloggten Users ab und gibt den ersten Kalender zurück.

fetch_events

Gibt alle Events in einem bestimmten Zeitraum in geordneter Reihenfolge zurück. Dafür werden die Events zusätzlich durch die Methoden `get_title_and_time_of_events()` und `create_parsed_date_objects()` für die Dialogausgabe vorbereitet.

get_title_and_time_of_events

Verarbeitet den ICal-String jedes Events und fügt jedes Event zu einer Liste hinzu. Jedes Event-Objekt enthält Start- und Endzeit sowie die „Summary“ bzw. den Titel des Events.

Create_parsed_date_objects

Erwartet eine Liste von Event-Objects und erstellt aus dieser eine parsed_events-Liste mit den für die Ausgabe korrekt formatierten Start- und Endzeiten. Ruft für dieses die Methoden parse_dates und `generate_output_date_string()` auf.

Parse_dates

Formatiert jedes Start- und Enddatum in ein konsistentes Format. Beispielsweise unterscheiden sich die Zeitangaben von Fullday-Events und Events, die eine bestimmte Uhrzeit besitzen. Die Formate werden aneinander angepasst, um sie gegenseitig auch vergleichen zu können.

Generate_output_date_string

Generiert den „response-string“ für die Datumsausgabe im Dialog wie zum Beispiel „7th of May“ oder „from 6pm to 7 pm“. Unterscheidet dabei zwischen fullday und normalen Events, da die Ausgabe für diese Events unterschiedlich sein soll. Ruft ebenfalls check_ordinal dafür auf.

Check_ordinal

Erwartet den “response-string” eines Events und überprüft, ob die Endungen für die jeweiligen Datumsangaben stimmen. Beispielsweise wäre die Endung in „1th“ falsch, weshalb die Methode dieses zu „1st“ korrigiert. Gibt den korrigierten response-string zurück.

Create_datetime_object

Erstellt für die übergebenen Parameter (Jahr, Monat, Tag, Stunde, Minute, Sekunde) ein Dateobjekt.

Create_parsed_events

Erwartet eine Summary (Titel des Events), Start- und Endzeit eines Events und erstellt ein ParsedEvent-Objekt aus diesen. Dafür wird innerhalb der Methode ebenfalls `parse_dates()` und `generate_output_date_string()` aufgerufen.

Rename_event

Benennt das übergebene Event um. Dafür wird über das Event-Objekt auf den Wert der Summary-Property zugegriffen und dieser auf den neuen Titel gesetzt. Mit `event.save()` werden die Änderungen auch im Kalender gespeichert.

Fetch_events_for_date

Diese Methode erwartet ein Dateobjekt, für welches die gespeicherten Events abgefragt und zurückgegeben werden müssen. Dafür wird auf dieses Datum genau ein Tag addiert. Anschließend werden die Methode `fetch_events()` mit dem ursprünglichen Datum und das eben berechnete Datum übergeben. So ist es möglich, alle Events von einem bestimmten Tag abzufragen.

Create_event

Diese Methode hat das Ziel, einen ICal String zu erstellen. In diesen werden die erwarteten Parameter “Title”, “Begin”, “End”, “Rule=None” eingepflegt. Das Format des zurückgegebenen Strings ist davon abhängig, ob im späteren im NextCloud Kalender ein ganztägiges Event angelegt oder dieses auf eine bestimmte Dauer festgelegt werden soll. Dafür wird der Boolean “Fullday” übergeben. In Abhängigkeit seines Wertes wird das passende String Format, wie im Folgenden dargestellt, verwendet. Da jeder Kalendereintrag eine eindeutige ID benötigt, wird ein Zeitstempel des aktuellen Zeitpunkts zum Erstellen des Termins genommen und mit kryptographischer Hashfunktion SHA1 kodiert. Die Variable „rule“ wird defaultmäßig auf den Wert „None“ gesetzt, da ein Termin nicht wiederholt werden soll. Bei einem Serientermin würde die Abfrage des „rule“ Parameters greifen und der ICal String um den Eintrag "FREQ={}\n" ergänzt werden. Im Rahmen des Projektes sind wir nicht weiter darauf eingegangen, ein Serienelement per Spracheingabe anzulegen. Im Caldav Code haben wir diese Möglichkeit jedoch grundsätzlich schon integriert.

ICal-String-Format eines mit Zeitpunkt angegeben Termins:

```
s = """BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//Sabre//Sabre VObject 4.3.0//EN
BEGIN:VEVENT
UID:{}
DTSTAMP:{}
DTSTART;TZID=Europe/Berlin:{}
DTEND;TZID=Europe/Berlin:{}
{}SUMMARY:{}
END:VEVENT
END:VCALENDAR
"""
```

ICal-String-Format eines ganztägigen Events :

```
    s = """BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//Sabre//Sabre VObject 4.3.0//EN
BEGIN:VEVENT
UID:{}
DTSTAMP:{}
DTSTART;VALUE=DATE:{}
DTEND;VALUE=DATE:{}
{}SUMMARY:{}
END:VEVENT
END:VCALENDAR
```

"""

Add_event

Diese Methode ruft die Methode `create_event()` auf und generiert einen gültigen ICal String. Dieser String wird benutzt, um die importierte Caldav Methode `save_event()` zu verwenden. Durch die Mitgabe des ICal Strings wird ein Termin im verbundenen NextCloud Kalender angelegt.

Remove_events

Diese Methode hat als Parameter eine Liste aus Events. Diese Liste wird iteriert und jedes Event aus dem Kalender gelöscht.

fetch_last_n_events

Gibt die letzten n Events, welche durch die Methode `fetch_events()` abgefragt wurden, zurück. Aus Performancegründen werden immer für ein bestimmtes Zeitintervall die Events aus dem Kalender geholt. Falls die Anzahl der Events kleiner als n ist, werden für das nächste Interval erneut die Events geholt. Dies wird so lange wiederholt, bis genug Events da sind oder bis das höchstmögliche Datum erreicht wurde.

fetch_next_n_events

Gibt die nächsten n Events, welche durch die Methode fetch_events abgefragt wurden, zurück. Auch hier werden aus Performancegründen immer nur für ein bestimmtes Zeitintervall die Events aus dem Kalender geholt. Falls die Anzahl der Events kleiner als n ist, werden für das nächste Interval erneut die Events geholt. Dies wird so lange wiederholt bist genug events da sind oder bis das höchstmögliche Datum erreicht wurde.

Skills

Nachdem die Entwicklung des CalDav Codes grundlegend abgeschlossen war, wurde mit der Entwicklung des eigenen Mycroft Skills begonnen. Hierfür haben wir uns an die im Pflichtenheft definierten Dialoge gehalten. Diese wurden als Vorlage genutzt und möglichst genau, mit ein paar Optimierungen durch eigene @intent_file_handler für jede Aufgabe umgesetzt. Im weiteren Bericht wird detailliert dargestellt, welche der im Pflichtenheft definierten Funktionen erreicht werden konnte.

Allgemein lässt sich sagen, dass für die Weiterverarbeitung der Spracheingaben in den verschiedenen @intent_file_handler Methoden Funktionen der Mycroft-Bibliothek benutzt wurden. Dies ist zum einen die Funktion `extract_datetime()`, mit deren Hilfe eine gesprochene Datumseingabe in ein gültiges Python Datetime Objekt geparsed wird. Somit ist es beispielsweise möglich, einen Termin für den nächsten Sonntag anzulegen und die `extract_datetime()` Methode parsed die Spracheingabe „Sunday“ in das Datum des nächsten Sonntags im Kalender. Ebenfalls wurde die Methode `extract_number()` genutzt, welche die Spracheingabe einer Zahl (z.B. „three“) in einen gültigen Integerwert umwandelt. Dies war hilfreich, um zum Beispiel mögliche Optionen mit Indexen als Spracheingabe auswählen zu können.

In Ergänzung wurden ebenfalls die Funktionen `speak_dialog()`, `get_response()`, `ask_yesno()` der Mycroft Instanz genutzt.

Dabei diente `speak_dialog()` dazu, die in einer Dialog-Datei definierten Sätze auszusprechen. Hierbei können auch Parameter übergeben werden, welche dann fließend in die Sprachausgabe integriert werden, so dass der gesprochene Text nicht statisch ist.

Ein Beispiel ist im Folgenden dargestellt.:

```
self.speak_dialog('calendar.si.appointment.date', {'date': parsed_events[0].date_response})
```

__init__.py

```
your appointments {date} are:
```

calendar.si.appointment.date

Die Funktion `get_response()` ermöglicht es, unkompliziert die Antwort des Nutzers in einer Variablen zu speichern.

Mögliche Anwendung:

```
event_title = self.get_response('calendar.si.ask.title')
```

__init__.py

```
Tell me the title of the new event.
```

calendar.si.ask.title

Die Methode `ask_yesno()` dient dazu, die Zustimmung des Nutzers zu einer bestimmten Frage zu erhalten. Der Aufruf der Methode evaluiert die gegebene Spracheingabe des Nutzers dahingehend, dass ein Boolean zurückgegeben wird, je nachdem ob es eine positive Rückmeldung war oder nicht. Allerdings funktioniert diese Evaluierung nur bei einer gesprochenen Eingabe und es ist beispielsweise nicht möglich über die Texteingabe mit „yes“ zuzustimmen. Deswegen entschieden wir uns dazu, immer auch noch eine Bestätigung über den Ausdruck „I confirm“ einzubauen, falls man die Anwendung nicht über die Spracheingabe bedienen, sondern getippte Antworten aufrufen möchte.

Beispielhafter Einsatz der Methode:

```
if confirmation_fullday_event == 'yes' or confirmation_fullday_event == 'I confirm':
    self.log.info("Fullday Event")
```

__init__.py 1

Verbinden mit dem Kalender

Startet man den Pi und möchte den Kalender-Skill verwenden, muss dieser sich zunächst mit dem NextCloud Account des Users verbinden. Dafür ist es notwendig, zunächst einen weiteren Ordner im Ordner ***mycroft-core*** mit dem Namen ***credentials*** auf dem Pi zu erstellen. Anschließend muss in diesem Ordner eine Textdatei mit dem Namen ***nextcloud.txt*** angelegt werden, die die Login-Daten des NextCloud-Accounts im folgenden Format enthält:

Max.mustermann@hdm-stuttgart.de

Musterpasswort

Danach kann durch die Eingabe „Connect my calendar“ eine Verbindung zum NextCloud-Kalender hergestellt und der vollständige Skill verwendet werden. Dieser Befehl muss immer zuerst ausgeführt werden, sobald der Pi gestartet wurde.

Der Handler connect_calendar, der die Verbindung zum NextCloud-Account herstellt, liest die unter der nextcloud.txt angelegten Login-Daten und erstellt mit diesen eine Instanz der CalDavCalendar Klasse, in der die Logik der einzelnen Features implementiert ist.

Abfrage des nächsten Events

Das erste Feature, welches implementiert wurde, ist die Abfrage des nächsten Termins, welcher durch das Statement „What is my next appointment?“ abgefragt werden kann (s. calendar.si.next.appointment.intent). Dazu wurde ein Handler in der __init__.py `get_next_appointment` erstellt, in welchem über die Caldav-Instanz die Methode `fetch_next_n_events` mit „1“ als Parameter aufgerufen wird. Diese Methode gibt demnach das nächstliegende Event zurück und die entsprechenden Informationen wie zum Beispiel Titel und Datum werden als Variablen im Dialog übergeben. Sind keine Events notiert, wird der Dialog „calendar.si.no.planned.events“ ausgegeben.

Abfrage der n nächsten Events

Eine weitere implementierte Zusatzfunktion ist die Abfrage von beliebig vielen Events. Dieses erfolgt durch die Abfrage „What are my next {n} appointments?“, wobei zu beachten ist, dass für {n} ausschließlich ausgeschriebene Zahlen wie „six“ oder „three“ verwendet werden müssen. Der dafür implementierte Handler `get_next_n_appointments()` parsed die übergebene Zahl und ruft über die Caldav-Instanz `fetch_next_n_events()` mit der Zahl als Parameter auf.

Falls nur drei Termine im Kalender existieren, jedoch nach den nächsten fünf Terminen gefragt wurde, werden nur diese drei Termine zurückgegeben. Ist kein Termin im Kalender gespeichert, wird dies auch im Dialog erwähnt.

Abfrage eines Events an einem bestimmten Tag

Der Calendar-Skill ermöglicht ebenfalls die Abfrage von Events an einem bestimmten Tag. Dabei sind Abfragen wie „What are my appointments on Monday?” oder “What are my appointments on May, 27th 2022?” möglich. Auch hier wird mithilfe der Methode `extract_datetime()` das Datum extrahiert. Anschließend wird die Methode `fetch_events_for_date()` mit dem Datum als

Parameter aufgerufen und die jeweiligen Events zurückgegeben. Auch hier wird der User benachrichtigt, falls für das Datum keine Termine existieren.

Anlegen eines neuen Termins

Als nächstes Feature wurde die Zusatzaufgabe angegangen, einen neuen Termin im Kalender über eine Spracheingabe anlegen zu können. Wir sahen uns diese Aufgabenstellung als zweites Feature an, da sie uns besonders umfangreich erschien und wir genug Entwicklungszeit einplanen wollten.

Die Funktion einen neuen Termin anzulegen kann über die im „calendar.si.create.event.intent“ definierten Spracheingaben aufgerufen werden (s. calendar.si.create.event.intent) . Eine dieser Varianten ist beispielsweise:

(create|add|plan|set up|make|schedule|arrange) (an|a new|a) (appointment|event|meeting|get-together|) (entitled|) {title}

Die Wörter in Klammern (()) stellen hierbei, wie zu Beginn beschrieben, eine Auswahlmöglichkeit dar,mit der eine gewisse Varianz der Spracheingaben geschaffen wird.

Für dieses Feature wurde ein Handler in der __init__.py `create_event_mycroft` erstellt, in welchem über die Caldav-Instanz die Methode `add_event` mit Event-Titel, Startzeitpunkt, Endzeitpunkt und dem Boolean Fullday, als Parameter aufgerufen wird. Diese Methode legt demnach ein neues Event im verbundenen NextCloud Kalender an. Die mitgegeben Parameter werden über die Spracheingabe gesetzt. Der Dialog läuft nach folgendem Schema ab:

- Hey Mycroft. Create a new event.
- ***Tell me the title of the new event***
- “New TITLE”
- ***For which date should I enter the appointment?***
- “DAY, MONTH, YEAR”, “SUNDAY”, “TOMORROW” etc
- ***Should it be an full day event?***
- “YES”, “NO”, “I CONFIRM”

Wenn es ein ganztägiges Event sein soll:

- ***I will add the fullday event {event_title} to your calendar {date_response}. Is that correct?***
- “YES”, “NO”, “I CONFIRM”

Wenn es ein zeitlich begrenzter Termin sein soll:

- ***At what time the event starts?***
- "9 AM” , “2 PM”, “11 o’clock”
- ***At what time the event ends?***
- "9 AM” , “2 PM”, “11 o’clock”
- ***I will add the event {event_title} to your calendar {date_response} {time}. Is that correct?***

Löschen eines Termins

Eine weitere Zusatzaufgabe bestand darin, dem Nutzer per Spracheingabe zu ermöglichen, Termine zu löschen. Dies setzten wir mit zwei verschiedenen Schemata um. Häufig möchte der Nutzer seinen nächsten oder letzten Termin löschen. Diese Löschung haben wir über zwei separate Intents umgesetzt. Den nächsten Termin zu löschen ist über die Spracheingabe „remove (my|the|) next (event|appointment)“ und den letzten Termin zu löschen ist über „remove (my|the|) last (event|appointment)“ möglich. Beide Dialoge laufen nach dem gleichen Muster ab:

- Hey Mycroft. Remove my next appointment.
- ***Should I remove {event_title} on {dateResponse}***
- „YES“, „NO“, „I CONFIRM“
- ***I deleted your appointment for {event_title} on {dateResponse}***

Für den Fall, dass der Nutzer ein Event löschen möchte, das nicht das letzte oder nächste ist, mussten wir uns einen Dialog überlegen, der es dem Nutzer so einfach wie möglich macht, einen Termin zu löschen. Um die Interaktion bei allen Skills so ähnlich wie möglich zu halten, entschieden wir uns für den gleichen Ansatz, den wir für das Umbenennen eines Termins genutzt haben. Dieser Skill wird durch die Spracheingabe „(remove|delete) (any|an) (appointments|events|appointment|event)“ ausgelöst. Häufig kennt der Nutzer den Tag des Termins, den er löschen möchte. Daher bauten wir den Dialog so auf, dass der Nutzer zuerst nach dem Datum des Termins gefragt wird. Daraufhin listet Mycroft alle Termine an diesem Datum mit Index auf. Daraufhin wird der Nutzer nach dem Index des Termins gefragt, den er löschen möchte. Um sicherzustellen, dass dieser Termin wirklich gelöscht werden soll, fragt Mycroft noch einmal nach, ob dieser bestimmte Termin wirklich gelöscht werden soll. Wenn der Nutzer zustimmt, wird der Termin aus dem NextCloud Kalender gelöscht. Der Dialog sieht wie folgt aus:

- Hey Mycroft. Remove event.
- ***On which date do you want to delete an appointment?***
- "DAY, MONTH, YEAR", “SUNDAY”, “TOMORROW”, etc.
- ***{index} {summary} {date_response}***
- ***Which appointment do you want to delete ? Please tell me the index of that appointment.***
- „INDEX“
- ***Should I remove {event_title} on {dateResponse}***
- „YES“, „NO“, „I CONFIRM“
- ***I deleted your appointment for {event_title} on {dateResponse}***

Umbenennen eines Termins

Als letztes Feature beschäftigten wir uns damit, wie es einem Nutzer ermöglicht werden kann, über eine Spracheingabe einen vorhandenen Termin umzubenennen. Bei der Überlegung, wie man den dazu benötigten Dialog designen könnte, fiel schnell auf, dass dies nicht so einfach umgesetzt werden kann, wie es auf den ersten Blick scheint. Um eine gute und einfache Nutzung zu schaffen, reicht es nicht aus, den Nutzer einfach einen Termin nennen zu lassen und diesen Termin umzubenennen. Das Problem bei diesem Format ist, dass der Nutzer bei der Spracheingabe zwar oft wissen wird, an welchem Tag der Termin, welchen er umbenennen möchte, stattfindet. Den genauen Wortlaut des ursprünglichen Titels wird er jedoch in wenigen Fällen wissen, so dass eine eindeutige Identifizierung des Termins nicht über die Abfrage des Titels möglich ist. Dieses Problem gingen wir in unserem Dialog-Design so an, dass der Nutzer zunächst nach dem jeweiligen Datum, an welchen er einen Termin updaten möchte, gefragt wird. Im Folgenden werden ihm die an diesem Tag anstehenden Termine mit einem Index versehen genannt. Der Nutzer kann anschließend über den passenden Index seinen gewünschten Termin zum Abändern auswählen und dessen Titel ändern.

Diese Lösung ist über die Spracheingaben „(rename|update|change) (event|appointment)“ (s. calendar.si.rename.event.intent) möglich. Für das Feature wurde ein Handler in der __init__.py `rename_event_date` erstellt, in welchem über die Caldav-Instanz die Methode `rename_event` mit dem neuen Event-Titel als Parameter aufgerufen wird. Im Folgenden wird der entsprechende Termin umbenannt.

Der Dialogablauf sieht dabei beispielsweise folgendermaßen aus:

- Hey Mycroft. Rename Event.
- ***On which day should I rename an event?***
- "DAY, MONTH, YEAR", “SUNDAY”, “TOMORROW”, etc.
- ***{index} {summary} {date_response}***
- ***Which appointment do you want to rename? Please tell me the index of that appointment.***
- "INDEX”
- ***Should I rename {event_title} on {dateResponse}?***
- "YES", “NO”, “I CONFIRM”
- ***What should be the new title?***
- “TITLE”
- ***I am going to rename {old_title} to {event_title}. Is that correct?***
- "YES", “NO”, “I CONFIRM”
- ***I successfully renamed the appointment. |I couldn't rename the appointment.| You don't have any events planned.***

Abgrenzung

Wie zuvor im Pflichtenheft anhand der aufgezeigten Szenarien definiert, gingen wir im Rahmen dieses Projektes nicht auf alle Möglichkeiten ein, welche der kombinierte Einsatz von Mycroft und Caldav bietet. Es wurde keine Lösung implementiert, um bei einem Termin die Ortsinfo abzufragen oder ergänzen zu können. Ebenfalls ist es aktuell nicht möglich, über Spracheingabe die gewünschte Zeitzone anzugeben, sondern diese orientiert sich an der für das Team relevanten Zeitzone

„Europe/Berlin“. Jedoch wurde der Timeoffset gegenüber der UTC, welcher dazu dient, die Zeit korrekt anzugeben, schon variabel implementiert. Zusätzlich ist anzumerken, dass wir uns nicht näher mit der Option beschäftigt haben, Serienelemente an Events anlegen oder auch abzufragen zu können. Diese Option wurde vom Projektumfang ausgeschlossen.

Reflexion und Abgleich Pflichtenheft

Insgesamt gelang es uns gut, die Aufgabe umzusetzen und uns in Mycroft einzuarbeiten. Die Vorarbeit durch das Eliza Projekt war dabei sehr hilfreich, da der Aufbau von Dialogen und Antwortmöglichkeiten ähnlichen Prinzipien folgt. Die größte Schwierigkeit bestand für uns darin, den Caldav-Code mit dem Mycroft Code zu vereinen sowie mit den Zeitzonen umzugehen.

Unseren Anforderungen aus dem Pflichtenheft konnten wir größtenteils gerecht werden. Hierbei kam es nur zu kleinen Änderungen an den Dialogabläufen, wie zum Beispiel dass beim Löschen oder Umbenennen eines Events, die Auswahl über einen Index und nicht über den Titel des Events festgelegt wird. Grund hierfür war, dass der Index über die Spracheingabe leichter weiter zu verarbeiten und weniger fehleranfällig war. Ein Index wird von Mycroft zuverlässiger von Sprache in Text umgewandelt als ein ausgesprochener Eventtitel. Auch für den Anwender erschien es uns leichter, sich den Index anstatt den gesamten Titel im genauen Wortlaut zu merken, um das entsprechende Event zum Löschen oder Umbenennen auszuwählen.

Bei dem Anlegen eines Events wurde hinzugefügt, dass über die Spracheingabe bestimmt werden kann, ob es sich um einen ganztätigen Termin handelt oder nicht. Die Begründung für die Änderung liegt darin, dass wir vor Beginn der Implementierung nicht wussten, wie genau ein Termin im Nextcloud Kalender gespeichert wird. Dementsprechend war uns die verschiedene Formatierung von ganztägigen und nicht ganztägigen Terminen nicht bekannt.

Insgesamt passten wir auch einige der Dialogabläufe dahingehend an, dass der Gesprächsfluss noch fließender klingt. Deswegen wurden die Sprachausgaben nicht 1:1 dem Pflichtenheft entsprechend implementiert. Der Inhalt blieb gleich, nur der Wortlaut hatte sich teilweise verändert.

Im Pflichtenheft wurde definiert, dass bei Nichtverstehen ein default-Dialog gestartet wird, welcher fragt, ob abgebrochen oder die Eingabe wiederholt werden soll. Dies haben wir nicht extra umgesetzt, da Mycroft bereits bei Nichtverstehen der Eingabe den letzten Dialog wiederholt, um die Eingabe beim zweiten Mal zu verstehen.

Die Meilensteine konnten wir meist zum gesetzten Zeitraum erreichen. Zu Beginn der Entwicklungsphase brauchten die ersten Meilensteine länger. Bei den Features kamen später Fehler auf, welche Zeit in Anspruch nahmen. Das Projekt konnten wir jedoch zeitgerecht abschließen, da gegen Ende die Feature-Entwicklung durch das gesammelte Wissen im Umgang mit Mycroft beschleunigt werden konnte.

Insgesamt, wurde noch die Bedingung ergänzt, dass der Benutzer vor dem Benutzen der Features unseres entwickelten Skills, sich zunächst mit seinem Kalender verbinden muss, wie im Abschnitt „Verbinden mit dem Kalender“ beschrieben. Dieser Vorgang wurde im Pflichtenheft noch nicht angeführt, da uns nicht bewusst war, dass es eine zwingende Voraussetzung sein würde, den „connect calendar“ Befehl als erstes auszuführen.