INTELLIGENT CALENDAR SCHEDULER ON EMAILS

Sonali. K

Department of LTRC, IIIT, Hyderabad, Gaccibowli

ABSTRACT

In past years of research efforts many have tried to implement an automated system based on time management. But besides calendaring, communication and collaboration event is negotiated. Based on experience in developing and deploying an automated event scheduling system, this paper sets the requirements for an intelligent calendar scheduler that retrieves emails from Gmail, and schedules all of them by taking into account of the natural language and dissolving certain ambiguities. We outline all these issues and schedule and proper calendar for the user and also change the according to the user's convenience or if timings clashes an alternating time is recommended.

KEYWORDS

Scheduler, python programming, Graphical user interface (GUI).

1. INTRODUCTION

Various electronic calendar applications are these days ubiquitous. Such as these common commercial applications, like Google Calendar, and Microsoft Outlook. Among these applications, some of them provide assistance with the related aspect of task management. Indeed, these applications, doesn't take into account the natural language conversions (e.g., "we shall meet tomorrow") the word "tomorrow" had to be converted to a date referring to the date at which the client is referring for, these ambiguities cannot be dissolved by these systems and so are many (e.g., "The meeting is scheduled at 10") the notion "10" here has to be identified as time and not as a number. Our approach was to make a better one by using NLP techniques. Firstly extracting emails from users Gmail account and parse through the sentences and extract the person, location, time, date. Then schedule the activities removing ambiguities of not clashing the timings and more.

Research in event scheduling has mainly concentrated for coordinating and arranging meetings between groups of people and left out such conversions from natural language to machine understandable scheduler system which results in poor overall utilization and unnecessary user effort in overall time management. Our proposal is based on our experience from working on systems like schedulers. We outline a rich set of ambiguities dissolved. We also sketch a hierarchical preference model that could be used to evaluate alternative schedules. Finally, some engineering issues are discussed that concern the development of an intelligent calendar assistant based on the proposed model. The goal is to emphasize the multiple aspects remove the ambiguities formed from natural language and inherent complexities of the problem of organizing a user's time and, thus, provide the ideas for implementations and subsequent formulations. In this paper the logic is implemented in python programming language and GUI is designed in java.

2. PROBLEM FORMULATION

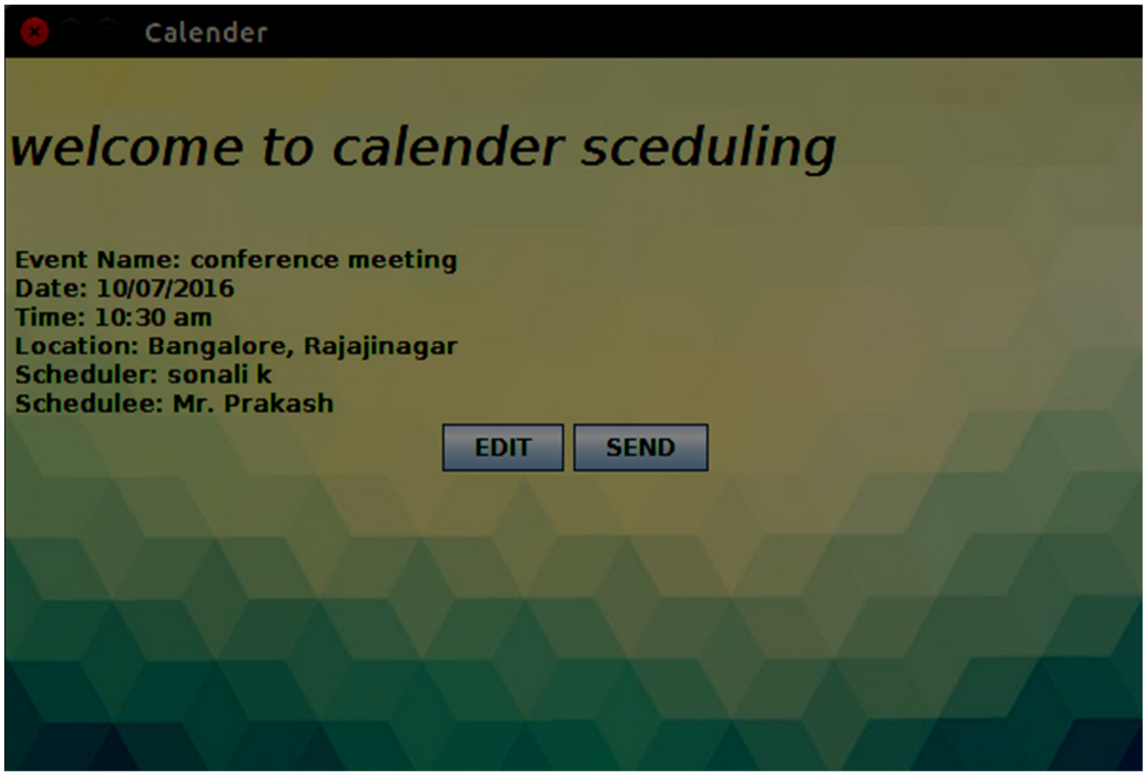
Before delving into the details of an event's description, we would like to discuss our view of hard and soft constraints. Organizing a user's calendar is a very ambitious task to be accepted by the user, so we expect from the system to be as flexible as possible by presenting to the user severalalternative schedules and let him decide the best. Flexibility is greatly favoured by soft constraints instead of hard ones. So, it is expected that the user will use as far as possible soft constraints (i.e., preferences) to describe events and tasks. However, there are cases where several value assignments to the decision variables are strongly prohibited. Here are some examples:

- If the user wants to attend a lecture, in this event has a very specific time and location. There is little value in scheduling this event in another time or location.
- The user is attempting to organize a meeting with another participant (a two-person meeting). In that case, the constraint that both participants are necessary for the meeting is hard; there is no point in having a meeting without one of the participants!
- A location with specific features is required for a meeting.

Physical constraints such as that the duration of an event cannot be a negative value or that the end time is equal to the sum of the start time and the duration. In all the above examples there are some decision variables, such as the start time or the location, that have very specific acceptable values or ranges of values. There is no point in assigning to these variables out-of-the- domain values; in such cases it is preferable not to schedule the event at all! However, even in such cases, whether an event can be omitted or not is something that the user decides. If an event is necessary (another type of hard constraint), the system should include it if possible. So, provided that a problem has been formulated with hard and soft constraints, we always prefer a schedule where all hard constraints are satisfied.

3. METHODOLOGY

Scheduler aims at helping the user to schedule her own individual events. Each event is characterized by its duration, date, and location (among several other attributes). The system tries to accommodate the tasks within the user's calendar, taking into account a variety of constraints and ambiguities. It aims at helping the user to schedule events, while taking into account user Preferences also. The system does not support overlapping of events. This is a consequence of the fact that the system can solve a different scheduling problem where all events need a single resource that is the user. The system is processed into 4 different phases:



welcome to calender sceduling

Event Name: conference meeting
Date: 10/07/2016
Time: 10:30 am
Location: Bangalore, Rajajinagar
Scheduler: sonali k
Schedulee: Mr. Prakash

Figure 1: Front page of scheduler.

3.1. Main function:

Extract emails from imaplib module in python starting from the recent mails to any number of mails to be scheduled.

```
def mails():
 _parser = HTMLParser.HTMLParser()
 def listLastInbox(top = 3):
mailbox = imaplib.IMAP4_SSL('imap.gmail.com')
                    mailbox.login('USERNAME', 'PASSWORD')
                    selected = mailbox.select('[Gmail]/All Mail')
                    assert selected[0] == 'OK'
                    messageCount = int(selected[1][0])
                    for i in range(messageCount, messageCount - top, -1):
                              reponse = mailbox.fetch(str(i), '(RFC822)')[1]
 typ, data = mailbox.fetch(i, '(UID BODY[TEXT])')
if __name__ == '__main__':
                    for message in listLastInbox():
```

3.2. Phase One:

Check whether its a scheduling mail or not by identifying synonyms of meeting in it.

```
//create a list of synonyms of meeting
lists1=["meet", "meeting", "join", "connect", "catch up", "get-together", "engagement",
"convention", "conclave", "event", "come"]
for words in data[0][1]:
if words in lists1:
            break
```

3.3. Phase Two:

Synonyms of abort are searched for not scheduling the activity if the user doesn't want to. The algorithm is similar to phase one.

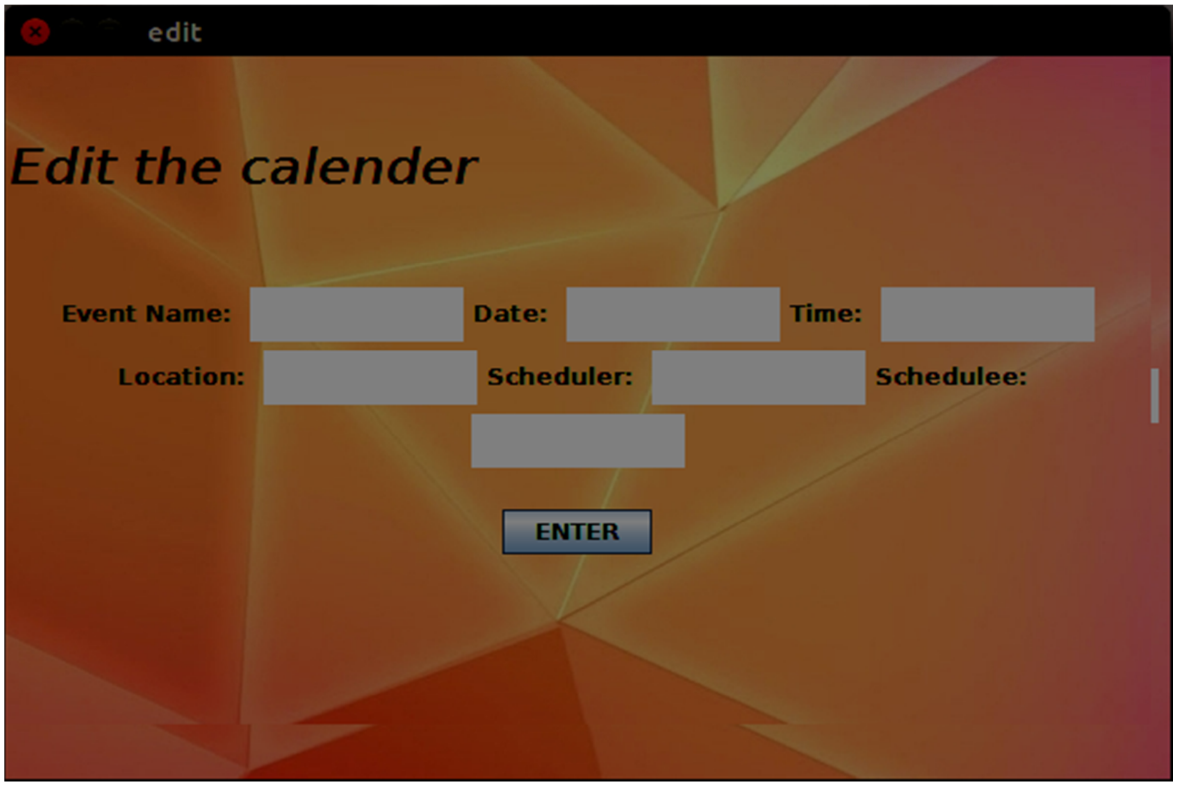


Figure 2: Edit page of scheduler

3.4. Phase Three:

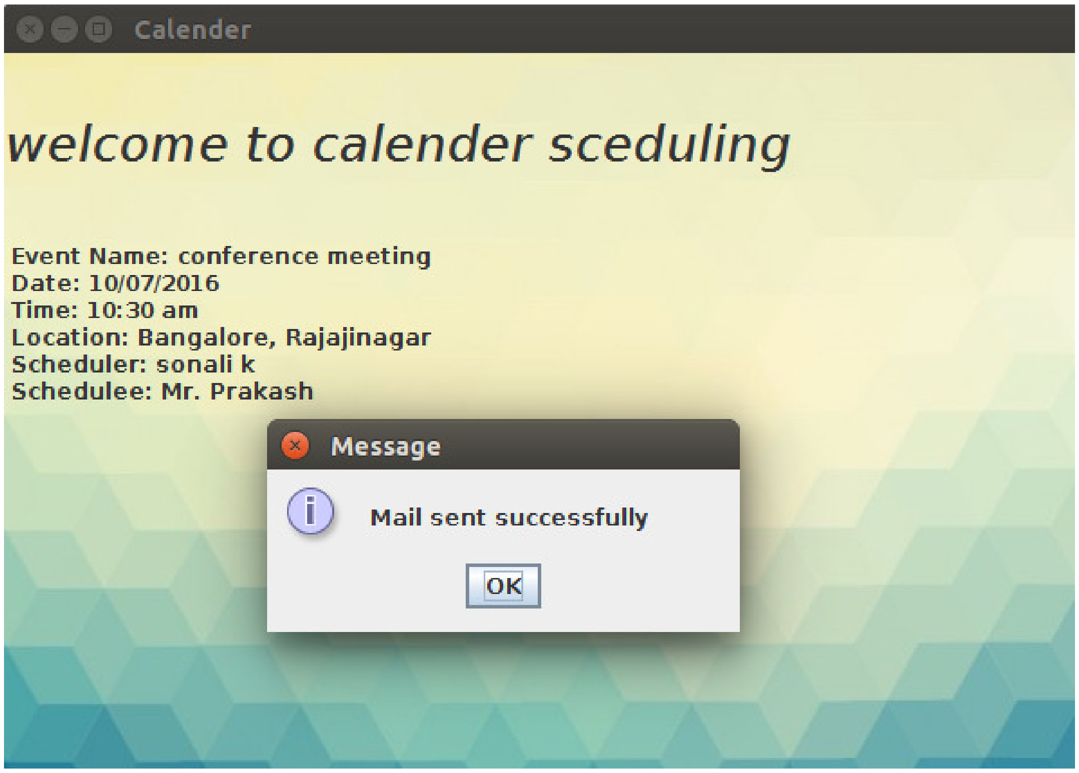
Checks whether synonyms of confirmation is there to proceed for final schedule. If it's a scheduling mail and confirmed for scheduling the body of the mail is written into a file. Then read_blocks function is called where Stanford's software of Named Entity Recognition is used. Where it parses through each words and names whether it's a name, location (organisation), time, date, and stores them in its respective files.

```
def read_blocks(subjects):
      for line in file:
         entities=st.tag(line.split())
            for i in entities:
                     if (j=='noon'):
```

```
                //calculate the time that's 12pm
                if (i[count]=='PERSON'||'ORGANIZATION'||'LOCATION'||'DATE'||'TIME'):
                //place the words in appropriate lists by appending them
```

Then a date function is called for correctly identifying its date if the user has written in words. It identifies words like Monday, next then a date function is called for correctly identifying its date if the user has written in words. The system can also cancel the scheduling after certain mails. The system refers to previous discussions/email threads and consolidate across the same email thread to schedule.

```
def dates():
for data in file:
               if (data=="next"):
            //calculate the date according to day mentioned in sentence based on the word "next"
               if (data is a day)
//calculate the by (time.strftime(data+"-%m-%Y"))
```



welcome to calender sceduling

Event Name: conference meeting
Date: 10/07/2016
Time: 10:30 am
Location: Bangalore, Rajajinagar
Scheduler: sonali k
Schedulee: Mr. Prakash

Mail sent successfully

Figure 3: Save and send confirmation.

3.5. Binary constraints:

No two events are scheduled at same time is taken care. Example two events like A and B events are considered then either event A or B is scheduled on different time if they are clashing based on user preferences.

3.6. Partial Scheduling:

A partial schedule is created if no schedule can be found that accommodates all tasks, the best incomplete schedule is adopted and presented to the user. Furthermore, the user is informed of all the changes and which tasks or parts of tasks were unable to schedule.

4. DECISION VARIABLES AND HARD CONSTRAINTS

In the following we use the term 'event' to refer to an event or a task interchangeably. Towards a unified calendaring assistant that exploits AI scheduling, we propose attributes to include:
Duration range: All the meetings are fixed by default for 30 minutes.Time identification: if a digit is given it identifies whether the person is asking to schedule it in am or pm only between the time period go 8am to 6 pm otherwise a prompt is given to schedule it within the constraint
Allotting time period: if already a time period is taken then the user is prompted to schedule it in another time. It can we work on the program differentiating between the following examples:
Task 1: "Let's meet up tomorrow?" "Sure, let us meet at 5.00 pm.", "Okay, see you then." or "Sure, I'll be there.": The program is able to reference the previous discussions/email threads and understand what 'then' refers to and consolidate across the same email thread to schedule. "Then" is critical to the discussion as it is the final confirmation on the meeting.

Task 2: "Let us meet tomorrow" vs. "I'll send you the report/file tomorrow": The program should be able to detect if it's a schedulable email or not. A key would be if the email thread has variations of the words "meet", "meeting" and also, synonyms like "join", "connect", "catch up", "get together", "engagement", "convention", and "conclave "," event".Some challenges while disambiguating:

Example 1: "Can we meet tomorrow (or Thursday) at 5.00 pm": The system should be able to pull out the date based on either the trigger "tomorrow" or "Thursday" and calendar for June 2nd (tomorrow/Thurs).

Example 2: "Let us meet next Monday/the day after tomorrow at 5.00 pm": The system should be able to pull out the date based on either the trigger "next Monday" or "the day after tomorrow" and calendar accordingly.

Example 3: "Let us meet on May 5th at noon": The system should be able to pull out the time (12.00 pm) based on the trigger "noon" and calendar accordingly.

Example 4: "Let us meet at Christie's": The system should be able to take "Christie's" as a place (even if it's a friend's place) and not as a person's name and use it for calendaring "the location". One clue could be in the preposition "at" which will not be used with a person's name.

5. CONCLUSIONS AND FUTURE WORK

This paper argues that organizing a user's time in an automated way is an opportunity and a challenging problem. Based on our experience in the field, we present a rich set of attributes and ambiguities that can be used to model this problem. We also proposed a set of attributes that could be used to evaluate alternative schedules. In the implementation of an intelligent calendar assistant, we identify several engineering problems:

- Problem solving: A fast, but incomplete, search algorithm could be used to produce the alternative schedules.
- Over constrained problems. The scheduler should always present a schedule to the user, even if it is not possible to satisfy all hard constraints and ambiguities. Suggestions to relax such constraints should be provided.
- Incremental scheduling. Contrary to the previous requirement, the system should also give the user the option to keep its current scheduling parts as fixed as possible.
- Adaptively. The system should provide the user with to define new constraints and preferences, over the decision variables. This can be achieved through a constraint programming language. Thus, our next step is to develop a consistent and rigorous model using better scheduling algorithms for the problem formulation described in this paper and to evaluate the alternative schedules using a subset of the suggested evaluation model.
- Also implementing all the limitations like scheduling, calendaring across time zones, incorporating user location maps, etc. In addition, it should also detect the issue of am or pm if only digit is written comes when we are working across different time zones, (for e.g., 2.00 pm IST vs. 2.00 pm PST).This constitutes our future work.

REFERENCES

[1] Ioannis Refanidis 1, 2 And Neil Yorke-Smith "On Scheduling Events And Tasks By An Intelligent Calendar Assistant" 1-10

[2] Google Calendar "Https://Apps.Google.Com"

[3] Microsoft Outlook Https://Www.Microsoft.Com/En-In/Outlook-Com/

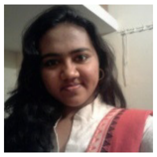
[4] Python Programming "Http://Learnpythonthehardway.Org/"

[5] Retrieve Emails "Http://Code.Runnable.Com"

[6] Send Emails "Http://Www.Tutorialspoint.Com"

[7] Python Programming "Https://Www.Codecademy.Com/"

Author



Sonali k,Student of National Institute of technology, Karnataka, 3rd year, IT branch. Passionate in research in the field of NLP.