Natural Language Date-Time Parsing in Chandler™

by
Darshana Chhajed

A PROJECT REPORT

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

in

Computer Science

Presented May 24, 2007
Commencement June 2007

ABSTRACT

The increasing need to share and synchronize personal information, such as schedules, tasks and events, amongst users has lead to the development of inter-personal information management software like Chandler™. Chandler is being developed in Python at the Open Source Applications Foundation, San Francisco. Before I started working on the project 'Natural Language Date-Time Parsing', Chandler recognized only one date format, mm-dd-yy, and one time format, hh:mm AM/PM. My goal was to allow Chandler users to enter any natural language date/time formats, such as 'May 10, 2007', '3pm' and 'lunch tomorrow', instead of being bound by any specific format. The project was divided into three parts. In the first project, I implemented natural language date/time parsing in the start date/time and end date/time fields for the Calendar Events in Chandler. The second project was to identify the start date/time and end date/time attributes of an Item when it is added to the Calendar. The third project involved the text widget in Chandler's Toolbar that was used only as a search-box. I converted this text widget into a Command Line Interface that can not only be used to search Items but also to create new Items quickly. The new Items created using CLI, were parsed for natural language date/time information to set their attributes properly.

ACKNOWLEDGEMENTS

I express my sincere thanks to my academic advisor and major professor, Dr. Timothy Budd for his tremendous support and the constant encouragement without which this project would not have been possible. I am particularly thankful to him for helping me get an internship at Open Source Application Foundation (OSAF).

I express my gratitude towards Dr. Bella Bose and Dr. Rajeev Pandey for sparing their valuable time to serve on my committee.

I am grateful to Philippe Bossut, Jeffrey Harris, Bryan Stearns and John Anderson at OSAF for helping and guiding me throughout the internship. I greatly appreciate Mike Taylor at OSAF for his help in the module ParseDateTime that was critical to this project.

My special thanks to my husband Manish Bothara and my family for their invaluable and unflinching support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

Personal Information Management (PIM) software [1] is used to manage information effectively. PIM software allows organizing personal information such as notes, to-do tasks, emails, schedules, reminders, contacts and more. They offer customizable planners and calendars for scheduling and task management. The most common examples of PIM software include Microsoft Outlook [2] and iCal [3].

In the last few years, the Internet has provided greater access to data and resources. Due to this, the Open Source culture [4] has proliferated and the sharing of data efficiently and collaboration between peers has become a prime necessity. Information stored using PIM software is isolated in nature and does not provide any support for collaboration and synchronization between users. Hence we need inter-personal information management software [5] to allow multiple users to share and collaborate all types of information. It facilitates information synchronization and updating between different repositories. This collaboration environment can be used to facilitate discussions, organize and coordinate projects, create and review documents, and manage the flow of information and tasks.

Chandler™ [6] is an Inter-Personal Information Management software application. It is an open source desktop application that runs on Macintosh, Windows and Linux. It is being developed by Open Source Applications Foundation (OSAF), which is based in San Francisco [7].

When I was searching for a project for my Master of Science degree, Dr. Timothy Budd, my advisor, directed me to the Open Source Lab (OSL) [8], at Oregon State University. OSL

serves as a medium for collaboration between the university and the industry. OSAF has an active association with OSL where it maintains a list of projects from which interested students can work on. While selecting a project of my choice, I was interviewed by OSAF. This led to an internship at OSAF in Summer 2006. During this internship, I became familiar with Chandler and worked on Natural Language Date/Time Parsing in Chandler.

Chandler integrates calendar, email, contact management, task management, notes, and instant messaging functions [9, 10]. Chandler has a rich ability not only to associate and interconnect items, but also to gather and collect related items in a single place creating a context sensitive "view" of many types of data. In Chandler, data is stored on repositories on the user's local machine and/or on shared resources such as servers to enable sharing of all types of information between users. Chandler's peer-to-peer calendaring system enables a group of users to efficiently schedule meetings, browse calendars of others, and see overlays of multiple calendars simultaneously. Chandler also provides unified search over all of the user's information stored in all Chandler repositories across the network. It allows searches to be saved for re-use. Chandler is a globalized software. It can be localized or customized to suit various locales like French and Spanish.

The programming language used for Chandler is Python [11]. Python is a dynamic multi-paradigm programming language [12]. It offers strong support for integration with other languages and tools and has extensive standard libraries. But Python programming language does not have a native graphical user interface or a presentation library. Instead graphical output is provided by third party libraries. Chandler uses the wxPython library [13, 14], which is a Python wrapper for the wxWidget cross platform GUI library written in C++.

Python is distributed under an open source license [15] that means it is free for anyone to use and the source code is available for anyone to look at and modify. Anyone can contribute fixes or enhancements to the project.

1.1 Chandler Terminology

Before describing Chandler's architecture [16], some concepts and terminology must be introduced: **Items, Kinds, Attributes, Repository, Collection and Stamping**.

The **Repository** is a database that contains information needed by Chandler as well as the contents of Chandler. The Repository is a persistent store, which supports full text search, sorting and indexing of sets of **Items** where an Item is the fundamental unit in Chandler. An Item is a Python object that persists to Chandler 's Repository. Each Item object belongs to a Python class.

The class that defines an Item is known as its **Kind**. The Kinds of Item available are Calendar Events, Notes, Tasks, Messages, etc. User Interface objects like menus and views are also stored as Items, as are services like the background task that fetches email. A Kind class describes the **Attributes** of its Items, and persists this information to the Repository.

Attributes are the properties of the Item and have two parts: a key and its value. For example, an Attribute of the kind Calendar Event has key 'startTime' to indicate the time the event starts and the value as "7:00 AM". Items can have zero or more Attributes. For example, the Attributes of a Calendar Event are start date/time, end date/time, location, title, body, frequency of occurrence, etc. 'Triage Status' is an Attribute of every Item. It helps the user to

manage and organize Items more effectively depending on the attention or focus the Items require. Triage status can be ‘Now’, ‘Later’ and ‘Done’.

A **Collection** is of group of Items that mix and match different Kinds of Items. Users create user-defined Collections for the categorization of Items. Items can belong to one or more Collections.

Stamping is process used to modify the Attributes or the Kind of the Items in Chandler. An Item can be stamped as multiple Kinds at a time. For example, an Item can be stamped as a Task and a Calendar Event at the same time. It will have the Attributes of both Task and Event Kind.

1.2 Chandler Architecture

The concepts explained in Section 1.1 are a consequence of Chandler's **data-driven architecture**. Data-driven architecture [17, 18] is a design paradigm in which the application design begins with some knowledge of the metadata about the underlying data structures. The application then uses the metadata to create user interfaces, drive the application's behavior and provide for other functions. Item objects that are stored in the Repository describe the application Chandler.

Another aspect of Chandler’s architecture is that when an Item of a particular Kind is displayed in the user interface, the application will automatically pull up the appropriate code to view or manipulate the Item, based on the multiple Kinds the Item is stamped as. For example, if an Item is stamped only as a Calendar Event, it is displayed in the Calendar

View. If the Item is stamped as Calendar Event and Task both, the summary of the Item is displayed. The user interface of Chandler is discussed in Section 1.3 ahead. Hence Chandler is said to have a data-driven architecture.

Chandler also uses a variant of Model-View-Controller (MVC) architectural pattern [19, 20, 21], In the MVC architecture, **Model** is the domain-specific representation of the information on which the application operates. The **View** renders the Model into a form suitable for interaction by the users, typically a Graphical User Interface (GUI) or a Command Line Interface (CLI). The **Controller** processes and responds to events, typically user actions. It may invoke changes on the Model. Controller also allows manipulating the View. Figure 1 depicts the relationship between the Model, View and Controller. The solid lines indicate a direct association, and the dashed line indicates an indirect association.

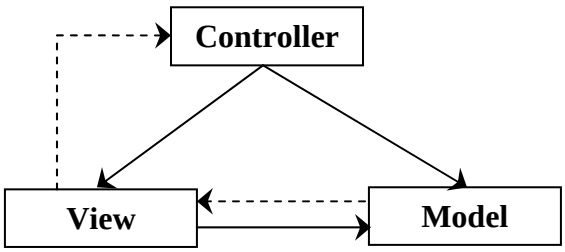


Figure 1: Relationship between the Model, View and Controller in the MVC architecture.

Like many applications built on GUI frameworks, Chandler tends to mix the View and Controller code in the Presentation layer. The Domain model and the Application Layer play the role of the Model and Controller in the MVC Architecture respectively. These main layers in Chandler architecture [16] are described as follows:

(i) Application Layer

The application layer is responsible for presenting the interface to the user. It handles the startup of Chandler. It contains the main "application" class that is primarily responsible for initializing Chandler components, as well as creating the MainView object that describes the layout of the application. The MainView object gets populated with UI Item objects like detail view, menus and sidebar.

(ii) Presentation Layer

Chandler's Presentation layer is handled by the Chandler Presentation and Interaction Architecture (CPIA). CPIA [22] blends the role of View and Controller in the MVC architecture. CPIA is a Chandler specific framework, so it has knowledge of the Repository, Items, etc. CPIA provides building blocks for Chandler's user interface, including some generic building blocks (e.g. Menus, Status Bar) as well as more Chandler specific building blocks (e.g. Sidebar, Calendar View, Detail View).

Another key concept in CPIA is that of an **Attribute Editor** [23]. Attribute editors are Python classes that handle the editing and rendering of the Attributes of an Item. Attribute editors are used by the Detail View or the Summary Table View to render and edit Attributes of a selected Item. For example, if a Calendar Event is selected in the Detail View, a date-time attribute editor handles the editing and rendering of the `startTime`, and a text attribute editor handles the editing and rendering of the `summary`. Attribute editors make use of wxPython widgets to help out with the rendering and editing.

(iii) Domain Model

The domain model defines all the domain specific Python classes that represent application content such as Calendar Events, Mail Messages, Tasks and Contacts. Each of these classes is a subclass of **ContentItem**. The ContentItem class is a base class for all Items that the user would typically think of as their personal data. The domain model has no knowledge of the layers above it, allowing it to be used effectively by different Views and Controllers. This layer also includes the code for Collections and notifications to the layers above.

Thus, the architecture of Chandler is derived from the MVC architecture pattern. Section 1.3 below explains the View or GUI of Chandler in details.

1.3 Chandler View

Chandler GUI has a three-pane or a three-column View [16], termed the Sidebar, Summary View and Detail View. The **Sidebar**, which is the leftmost pane, contains a list of all Collections, and one of them is selected. The list includes all the pre-defined Collections like ‘Dashboard’ and ‘Trash’ and also the user-defined Collections like ‘Home’ and ‘Work’. The **Summary View**, which is the center pane, displays the Items in the selected Collection, and one Item is selected. A few details like the title of the Item, its Kind and its Triage status are also displayed in the Summary View. The **Detail View**, which is the rightmost pane, displays details of the Item selected in the Summary View. All the Attributes of the selected Item can be viewed and edited in this pane. The three-pane View allows users to view the Collection list, the contents of the selected Collection, and the Details of the selected Item all at once.

The **Application Bar**, which spans the top of the GUI, allows the user to select an application area (All, Calendar, Event, Task). This selection is used to filter what Kinds of Items are displayed and what type of Summary View is used. If "Calendar" is selected in the Application Bar, for example, collections of Calendar Events can be seen in a Calendar Summary View. If "All" is selected in the Application Bar, Collections are seen in a more general Table Summary View. The application also has other common elements: Menus, a Toolbar, and a Status Bar. Figure 2 shows the schematic arrangement of the elements in a typical Chandler screen.

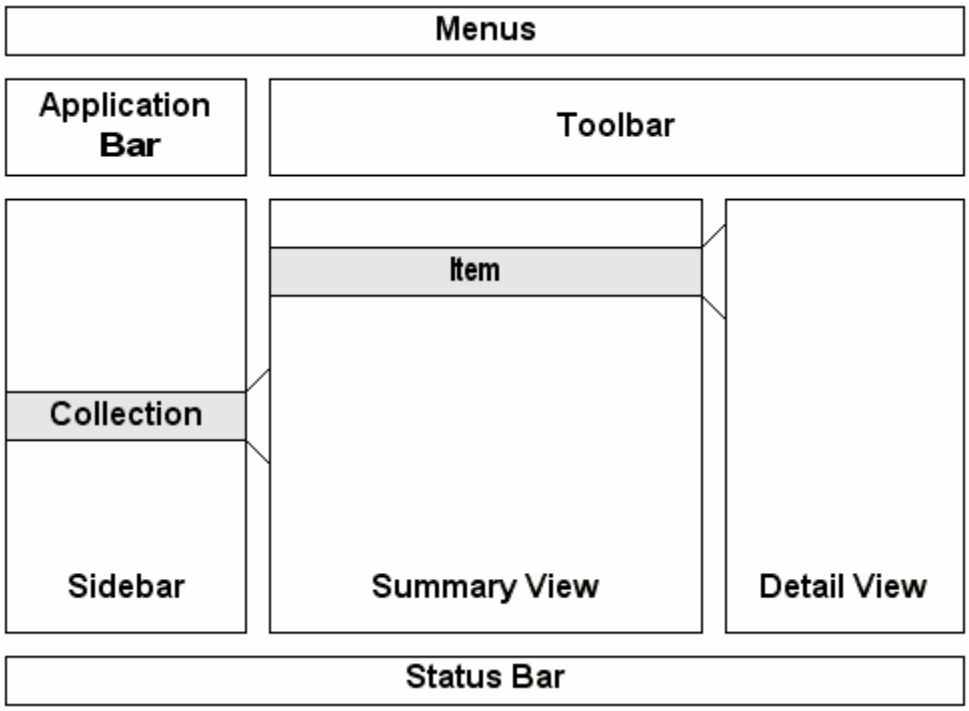


Figure 2: Schematic arrangement of various elements in a Chandler screen.

A screen shot of the actual Chandler screen showing the various elements and Views mentioned above is shown in Figure 3.

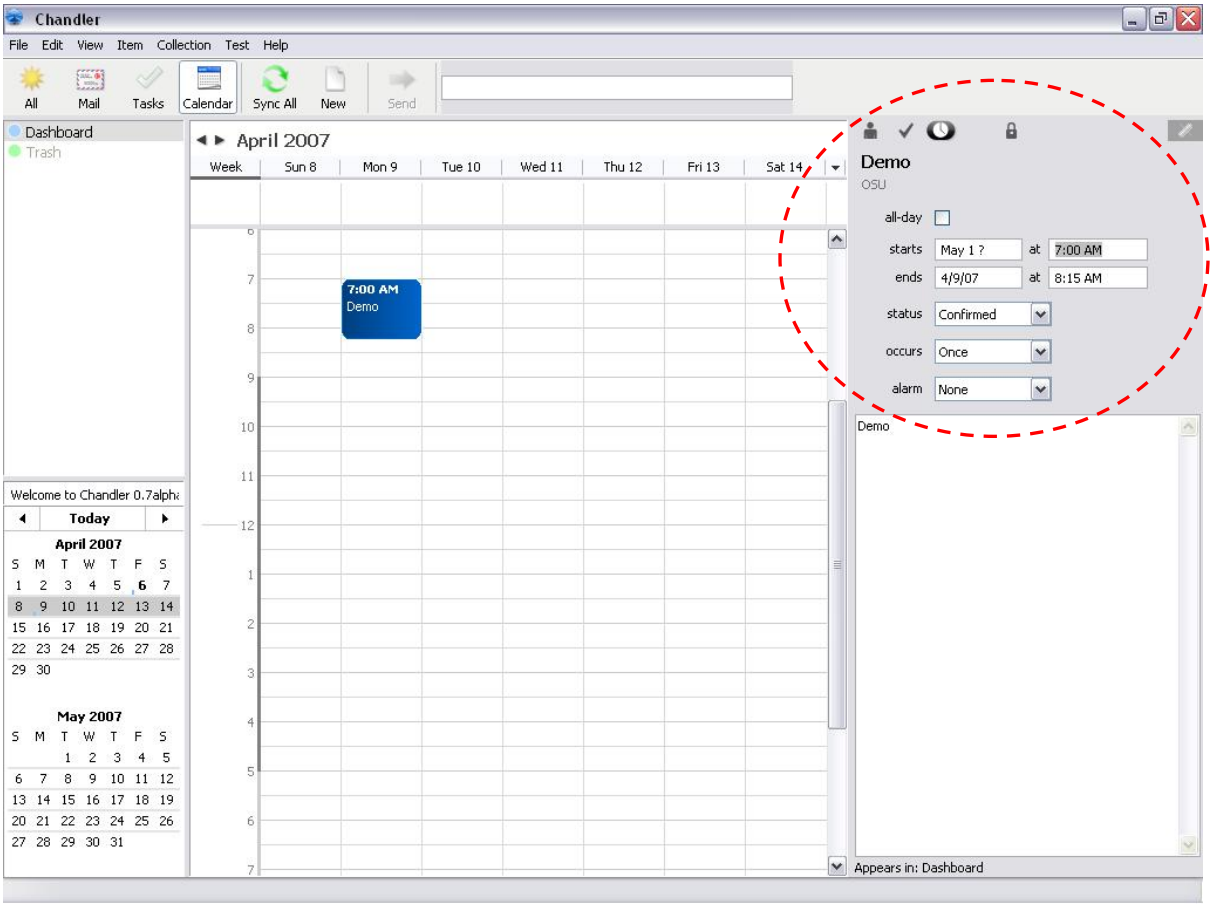


Figure 3: Typical screen shot of Chandler showing the various elements and views.

As seen in Figure 3, the ‘Calendar’ Application is selected in the Application Bar. The empty text widget in the Toolbar is the Command Line Interface (CLI) for Chandler. The ‘Dashboard’ collection is selected in the Sidebar. The ‘Calendar’ is seen in the Summary View. The area marked with the dotted circle marks the Detail View of the Event selected in the Calendar. The Attributes of the Event can be seen in this pane. ‘Demo’ and ‘OSU’ denote the Attributes Title and Location for the Event. The Event can be set for the entire day by checking the box ‘all-day’. The start and end dates/times are specified in the four text widgets named ‘starts’ and ‘ends’. For example, the start time for this Event is 7:00 AM and the end time is 8:15 AM. The status of the event can be set to Confirmed, Tentative or FYI.

The frequency of the occurrence of the event can be set to Once, Daily, Weekly, Biweekly, Monthly or Yearly. An alarm or a reminder can be set. More details can be added to the selected Item in the large text area widget for the Attribute ‘body’ of the Item. At the bottom right of the screen in the Detail View, all the Collections the Item belongs to are specified. For example, ‘Appears in: Dashboard’ indicates that this Event belongs only to the Dashboard Collection. The small icons just above the title ‘Demo’ in the Detail View are used for stamping the Item to other Kinds like Message and Task.

1.4 Objectives

The project is divided into three smaller projects depending on the area of focus in the Chandler GUI. The objective of the first project is to allow the users to enter natural language date and time strings in the start date/time and end date/time widgets in the Detail View of Chandler [24]. The second project is to parse the contents in the body of the Items for date/time information when they are stamped as Calendar Events using the Stamping icons in the Detail View or in the Dashboard Summary View [25]. If any date/time information is found, the start date/time and end date/time Attributes of the Calendar Event are set accordingly. The goal of the third project is to allow the text entry widget in the toolbar to be used as a CLI for creating new Items in Chandler quickly without the need to change the current view or context and also to search Items in the Chandler repository [26]. Natural Language date/time parsing is done to appropriately set the Attributes of the newly created Items. These three projects are discussed in details in the following sections.

2 Natural Language Parsing of Date/Time Strings in Detail View

Natural Languages [27] are ordinary languages used by humans for general-purpose communication. These languages are inherently ambiguous in nature. For example, the sentences “We gave the monkeys the bananas because they were hungry.” and “We gave the monkeys the bananas because they were over-ripe.” have the same surface grammatical structure [28]. However, in one of the sentences, the word ‘they’ refers to the monkeys, in the other it refers to the bananas: the sentence cannot be understood properly without knowledge of the properties and behavior of monkeys and bananas. Also, a string of words can be interpreted in myriad ways. In date/time parsing, the date “04/01/2007” could be interpreted as ‘January 4, 2007’ or ‘April 1, 2007’ depending of which format is considered: dd/mm/yyyy or mm/dd/yyyy.

Natural Language Parsing [29] is a process to interpret statements given in a Natural Language by resolving its inherent ambiguity and converting it to a more formal representation that is easier for computer programs to manipulate. In Chandler, users frequently need to enter dates and times in the Detail View shown in Figure 3 above. These dates and times can be used for several purposes such as setting Reminders and scheduling Calendar Events. The objective of this project is to allow the users to enter natural language date and time strings in the Detail View instead of being bound by any specific format.

2.1 Existing Working of Chandler

As discussed in Section 1.3, the date/time widgets in the Detail View of Chandler are used to indicate the start and end date/times for the Calendar Events and also for setting Reminders. Before I started working on this project, these widgets could parse the dates and times only in

one specific format. The recognized date format was mm/dd/yy and the time format was hh:mm am/pm. If the string was invalid or was in any other format, a question mark, '?', would appear at the end of the string indicating that the string is not recognizable. For example, as depicted in Figure 4(a), the string "May 1" entered in the start date widget is not a valid date format. Hence a '?' appears at the end. Similarly, in Figure 4(b), "5 pm" is not successfully parsed to 5:00 PM in the start time widget.

(a) (b)

Figure 4: The date/time widgets of the Detail View indicated by the red circle in Figure 3. 'May1' and '5pm' are not recognized by Chandler since these date and time formats are not supported.

2.2 Drawbacks of Existing Date/Time Widgets

The date/time widgets support only one format making them very rigid and error-prone. They do not understand commonly used natural language date strings such as weekday names, month names and time strings like noon and midnight. Also, the widgets provided very little or no support for auto-completion of recognizable strings. All these features such as auto-

completion and natural language parsing are essential to make the product more user-friendly and flexible to use. It is necessary to allow the users to choose amongst any widely used date/time formats. Also, these widgets did not support localization of dates and times in the locale or language Chandler is opened in.

2.3 Desired Features for Date/Time widgets

In this section, we look at the desired features that are required to enhance the working and usability of the start date/time, end date/time and the reminder time widgets in the Detail View of Chandler. The Design Team of Chandler at OSAF compiled this list of features and did research on it to check the feasibility of the features [30].

(i) Date Widgets

The date widgets should allow proper parsing of the following strings:

- More date formats like “May 1, 2007”, “May 1”, “May 1st, 2007”, mm/dd/yyyy, mm/dd
- Natural language date strings like today, tomorrow and yesterday.
- Weekday names from Monday through Sunday and their short formats like Mon through Sun
- Date separators like ‘-‘ and ‘.’
- Longer date strings like “ 3 weeks 4 days”
- Date strings with offsets like “day after tomorrow”, “next Monday”, “ 2 weeks ago”, “5 days from now”

(ii) Time widgets

The time widgets should allow proper parsing of the following strings:

- More time formats like “5 pm”, ”5pm”, “5p”, “1700”, ”5:00:00 PM” should all parse to “5:00 PM”
- Natural language time strings like noon, midnight, now, morning, evening, lunch.
- Longer time strings like “5 hrs and 45 min”, “4pm + 8hr”
- Time strings with offsets like “ 30 minutes from now”, “before 5hrs”, “10min after 12pm”

(iii) Shared Features:

Both the date and time widgets should have the following properties:

- Auto-completion support should be provided. For example, if the user types in “May 1” in the date widgets without specifying the year, it should be auto-completed to “05/01/2007”. Also, entering “5p” in the time widgets should be auto-completed to “5:00 PM”
- When the user types a string in the widgets, a drop down list containing all the possible date/time values starting with that string should be displayed. The first choice in the list is automatically selected. The user can change the selection by using the up and down arrows. On hitting TAB, the widget takes the value of the selection. For example, if the user enters “To” in the date widgets, then there should be two choices namely “Today” and “Tomorrow” in the dropdown list. The corresponding dates should also be displayed. Similarly, if the user enters “no” in the time widgets, the dropdown list will have “Now” and “Noon” as the choices along with corresponding times.

- If the date/time string entered by the user is not recognizable, a '?' should appear at the end of the string in the widget.

The desired features for the date and time widgets mentioned in (i) and (ii) above are generic in nature and are implemented using a standalone Python module explained in Section 2.5.1. Whereas, the shared features mentioned in (iii) are specific to the Detail View of Chandler and are discussed in Section 2.5.2.

2.4 Previous Work

Natural Language Date/Time parsing converts the date and times in different textual formats to a more uniform and unambiguous format which can be easily understood by the software applications. There are several toolkits and open source software such as MontyLingua [31], Natural Language ToolKit (NLTK) [32], FreeLing [33] and Wordnet [34] for Natural Language Processing. Software like python-dateutil [35], Chronic [36] in Ruby and NSGregorianDateFormatter [37] in Java specialize in date parsing.

Before I joined the project, Mike Taylor at OSAF had started working on a stand-alone Python module called **ParseDateTime** [38], which parsed a few natural language date time strings and converted them to a standard format. Starting with this simple module, we worked together to develop and enhance ParseDateTime to parse all the different date and time formats mentioned in Sections 2.3. The approach and implementation of ParseDateTime are discussed in detail in Section 2.5.1.

2.5 Implementation

The implementation of Natural Language Parsing in the Detail view of Chandler is done in two parts as mentioned in Section 2.3 above. The date and time features are implemented using the ParseDateTime module explained in Section 2.5.1. Changes made to Chandler to implement the shared features for the date and time widgets and to use ParseDateTime for parsing dates/times are discussed in Section 2.5.2.

2.5.1 ParseDateTime

ParseDateTime is an open source standalone Python module that parses the different natural language date/time expressions and converts them to a standard format. It is a small but powerful library which can be used to parse all the various date and time features mentioned in Section 2.5.1 (i) and (ii).

2.5.1.1 Approach and Implementation of ParseDateTime

A regular expression (regex for short) is a special text string for describing a search pattern [39]. We have used regex patterns to match the text entered by the user with the widely used date/time formats. If the text matches any pattern, the relevant date/time information is extracted from the text and is converted to a standard time object in Python.

The `re` module [40] in Python has a set of powerful regular expression facilities. The patterns provided by this module are similar to those available in Perl. We formed regex for all the widely used date/time formats, weekday names and other natural language date/time strings. An example of a regex for the time format hh:mm followed by the meridian information is shown in Figure 5. The regex is compiled and the text entered by the user is searched for the

regex pattern using the search() method. If the text matches the regex, the hour, minute and meridian information is extracted from the text using the group() method. Using this regex pattern, the strings “5:00 PM”, “5p”, “05:00PM”, “5:00p.m.”, will all be parsed successfully.

```
import re
RE_TIME = r'(?P<hours>\d\d?)(:(?P<minutes>\d\d))?\s?(?P<meridian>am|pm \
            |a.m.|p.m.|a|p)'
CRE_TIME = re.compile(RE_TIME,  re.IGNORECASE)
m = CRE_TIME.search(text)
if m is not None:
      min  = m.group('minutes')
      hrs   = m.group('hours')
      meridian = m.group('meridian')
```

Figure 5: Code showing pattern-matching using regex for the time format hh:mm followed by the meridian information

The output of ParseDateTime is a tuple of the form (time, typeFlag). The value of time is a standard time tuple in Python [41] having 9 attributes (year, month, day, hour, minute, second, weekday, yearday, is_dst). The attributes ‘year’, ‘month’ and ‘day’ specify the date. The attributes ‘hour’, ‘minute’ and ‘second’ denote the time. Irrespective of whether the user explicitly mentions the seconds in the time widget, seconds are stored because the standard time format in Python includes seconds. The attribute ‘weekday’ specifies the day of week; 0 being Monday, 1 being Tuesday and so on. The attribute ‘yearday’ specifies the day of the year; January 1 being 1 and so on. The attribute is_dst is a flag for Daylight Saving Time (DST). DST is an adjustment of the timezone by usually one hour during part of the year. The ‘is_dst’ flag is set to 1 if DST applies to the given time and it is set to –1 if it is unknown. If the time denoted is Coordinated Universal Time (UTC) or Greenwich Mean Time (GMT), the is_dst flag is set to 0.

These 9 attributes in the time tuple uniquely specify the date/time obtained after parsing the text entered by the user. The ranges of values for the attributes of the time tuple are listed in Table I.

Index	Attributes	Values
0	year	(for example, 1993)
1	month	range [1,12]
2	day	range [1,31]
3	hour	range [0,23]
4	minute	range [0,59]
5	second	range [0,59]
6	weekday	range [0,6], Monday is 0
7	yearday	range [1,366]
8	is_dst	0, 1 or –1

Table I: Attributes of time tuple in Python

The typeFlag in the output tuple (time, typeFlag) is an integer that denotes the type of information parsed from the text entered by the user. The typeFlag has value 1,2 or 3 if the text contains information of date, time or both. If the text does not have any date/time information that can be parsed by ParseDateTime, then the typeFlag value is 0 and the time tuple will have the current source date and time. This typeFlag helps to extract only the relevant information from the time tuple i.e. if the typeFlag has value 2, we can just use the ‘hour’, ‘minute’ and ‘second’ part of the tuple and ignore the rest of information.

The examples in Figure 6 show the working of ParseDateTime. The method parse() is called to parse the text. If the input is “now”, the current time is output and the typeFlag is 2. The date part can be ignored. When the input is “march 1”, the date is set to future March 1 which is in year 2008 since the date is already passed in the current year. The typeFlag is 1 since the text entered is date information. When the input is “noon tomorrow”, the date in the time tuple is set to the next day and the time is set to 12 pm. The typeFlag is set to 3 since both date and time information exists in the text.

```
$ python setup.py install
$ python
Python 2.4.2 (#67, Sep 28 2005, 12:41:11) [MSC v.1310 32 bit (Intel)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import parsedatetime.parsedatetime as pdt
>>> cal = pdt.Calendar()
>>> cal.parse("now")
((2007, 4, 18, 22, 43, 8, 2, 108, 1), 2)
>>> cal.parse("march 1")
((2008, 3, 1, 22, 44, 15, 2, 108, 1), 1)
>>> cal.parse("noon tomorrow")
((2007, 4, 19, 12, 0, 0, 3, 109, -1), 3)
```

Figure 6: Examples of parsing text using ParseDateTime

2.5.1.2 Testing

ParseDateTime was tested using the “black box” testing technique. This type of functional testing verifies the specifications of a program without consideration of the internal working or structure of the application. Testing was done by comparing the actual results and the expected results for the given set of inputs, without actually checking how the module arrived at those outputs.

Test cases were written for every unit of the application. A unit is the smallest testable part of the application. This testing procedure is called Unit testing and is used to validate that individual units of source code are working properly. Many unit tests have been written to test the working of the ParseDateTime library. Thorough testing was done with various examples of natural language date/time strings encompassing each and every date and time feature mentioned in Section 2.3. Mike and I developed test suites for simple dates/times, complex dates/times, dates/times with offsets, dates/times with different units, dates/times in long phrases and date/time ranges. A comprehensive list of all the dates and times tested is given in Appendix A.

While implementing ParseDateTime, we started with a very few natural language date/time formats and then incrementally added support for more formats. By doing this, we made sure that every line of code corresponds to the parsing of some date/time format given in the specifications. Thus, we were confident that all lines of code were being executed at least once. Also, each time we added a new format, we executed the tests cases from the previous formats to ensure that no new bugs were introduced. This is termed as regression testing. Hence implementation of other projects described in Chapter 3 and 4 was preferred over white box testing of ParseDateTime, although it would have been a foolproof method to assure complete code coverage.

A few examples of the unit tests for ParseDateTime are shown in Figure 7. The variable ‘target’ denotes the expected date/time for the given input. The output obtained after parsing the input using ParseDateTime module is of the form (tuple, typeFlag). The tuple is the standard time tuple and the typeFlag represents the type of parsed information as mentioned

in Section 2.5.1.2. The typeFlag 1 denotes date, 2 denotes time and 3 denotes both date and time. The output tuple is compared to the tuple (target, typeFlag). If both these are equal, the test is passed, otherwise fail. If any test case in the test suite fails, the testing is terminated immediately.

```
cal = parsedatetime.Calendar()
yr, mth, dy, hr, mn, sec, wd, yd, isdst = time.localtime()

target = datetime.datetime(yr, mth, dy, 23, 0, 0).timetuple()
self.assertTrue(_compareResults(cal.parse('11:00:00 PM'),  \
               (target, 2)))

target = datetime.datetime(2006, 8, 25, hr, mn, sec).timetuple()
self.assertTrue(_compareResults(cal.parse('Aug 25, 2006'),  \
               (target, 1)))

target = datetime.datetime(yr, mth, dy, 16, 0, 0).timetuple()
self.assertTrue(_compareResults(cal.parse('flight from SFO at 4pm'),  \
               (target, 2)))
```

Figure 7: Examples of test cases for ParseDateTime

2.5.2 Adapting Chandler

The standalone module ParseDateTime is used to parse the text entered by the user in the Date/Time widgets in the Detail View of Chandler. Changes are made to the working of these widgets in Chandler to allow the generation the dropdown list of choice and auto-completion [42].

2.5.2.1 Approach and Implementation

A comprehensive list of all the natural language date strings including weekday and month names is made. A similar list is made for natural language time strings. These lists are explained in detail in section 2.5.2.2.

When the user enters a character in the start or end Date widgets in the Detail view of Chandler, the list of date strings is scanned for all the strings that start with that character. If the matched string is a month name, the date is assumed to be the first day of the month. If the month has already passed in the current year, the next year is considered since the date is always preferably a date in the future. For example, if the user enters 'J' in the date widgets, the choices in the dropdown list are "July: 07/01/2008" and "June: 06/01/2008". The dates have value '1' and the year '2008'.

All the matches are now parsed iteratively by the ParseDateTime module, and the corresponding dates are obtained in the proper format. These matched strings constitute the choices in the dropdown list. This list keeps updating on every character the user enters. The first choice is selected by default. Other choices can be selected by using the up and down arrow keys. On hitting Return or Tab, the widget gets the date of the selected choice by auto-completion and the focus moves to the next item. The working of the start and end Time widgets is similar.

For example, in Figure 8(a), as soon as the user types 'T' in the start date widget, all the possible natural language date strings and their corresponding date appear in the dropdown list. Figure 8(b) shows the updated list as the user types the next character. The first choice in the dropdown list "Today" is selected by default. The user chooses the second choice "Tomorrow" by using the down arrow in Figure 8(c). On hitting Tab, the date value of the selected choice i.e. the date corresponding to "Tomorrow" is auto-completed in the text widget as shown in Figure 8(d) and the focus moves to the next widget, which is the start time widget.

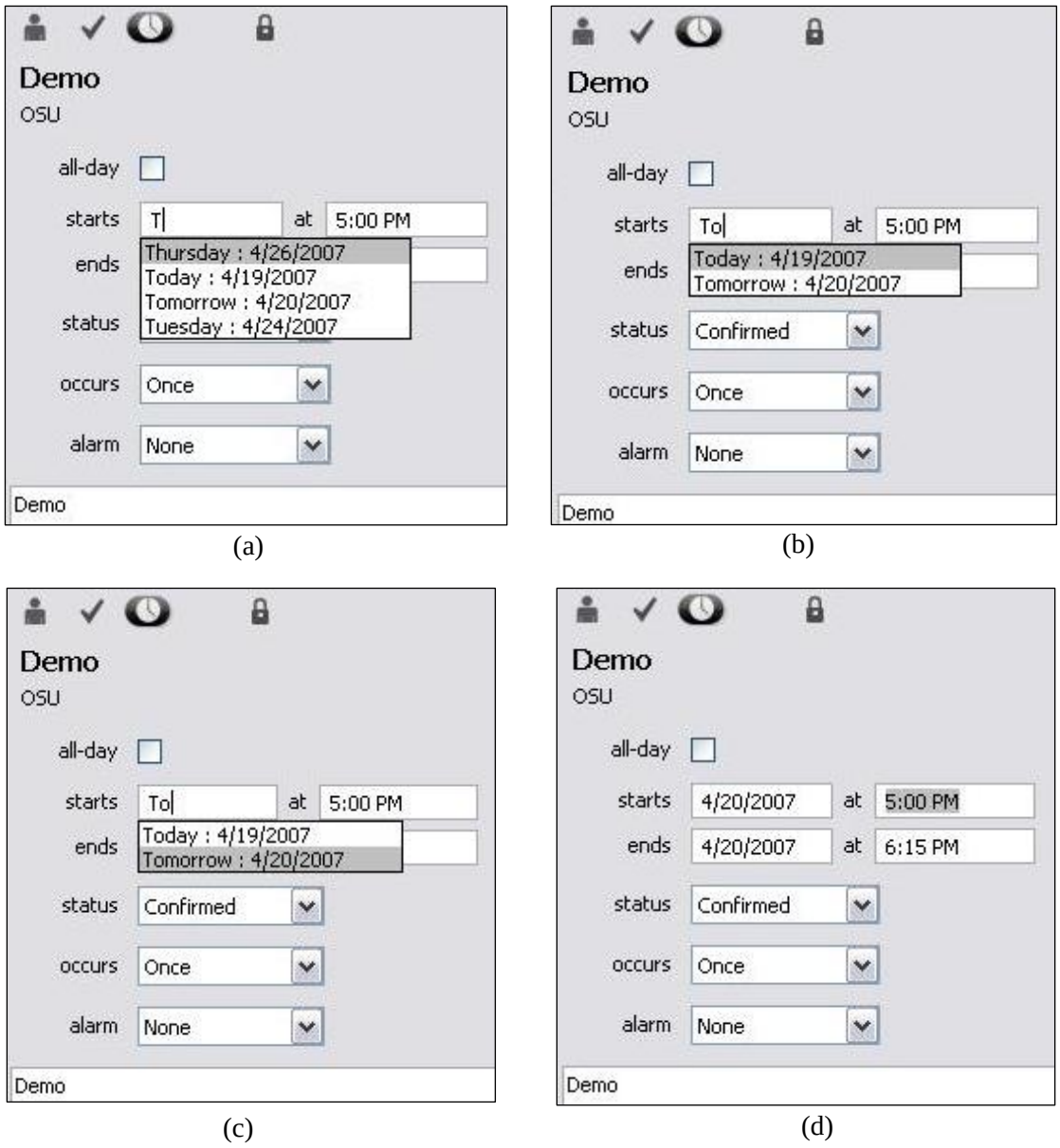


Figure 8: (a) Dropdown list appears as soon as user starts typing in the start date text widget. (b) The choices get updated when more text is entered. First choice is selected by default. (c) Select a different choice. (d) Date of the selected choice is auto-completed when the user hits TAB/RETURN.

2.5.2.2 Internationalization and Localization

Internationalization (i18n) and Localization (l10n) are the means of adapting software for non-native environments and languages [43]. The numeronym ‘i18n’ is an abbreviation for internationalization where 18 stands for the number of the letters omitted

(‘nternationalizatio’) between ‘i’ and ‘n’. Similarly, the numeronym ‘l10n’ is an abbreviation for localization. The process of making the framework such that it can be generalized to allow multi-language support is i18n. The process of customizing the software for a specific market or locale is l10n. Thus, i18n and l10n make the software portable to other locales.

All textual data in Chandler is Unicode [44]. Chandler is using the GNU gettext() approach in which a translation mechanism is used, that maps Unicode strings in the code to strings given in a translation dictionary. Localizing is the action of creating such a dictionary. In order to localize Chandler, all the strings that appear in the User Interface of Chandler are isolated and stored in separate resource files. These files get compiled into binary data that Chandler explicitly loads at runtime. In the code, these strings are marked by ‘_()’ syntax. For example, string ‘tomorrow’ is written as _(u'Tomorrow'). A tool scans the entire code for such strings and outputs their translated version. The open source standard translation tool **gettext()** is then used to lookup the proper localized translation of the English key.

The lists of natural language date and time strings are actually dictionaries such that the English words are the keys and their localized translations are the values. For example, the dictionary of natural language time strings is given in Figure 9 below.

```
#natural language strings for time
textMatches = {'Lunch':_(u'Lunch'),'Evening':_(u'Evening'),
               'Noon':_(u'Noon'), 'Midnight':_(u'Midnight'),
               'Breakfast':_(u'Breakfast'),'Now':_(u'Now'),
               'Morning':_(u'Morning'),'Dinner':_(u'Dinner'),
               'Tonight':_(u'Tonight'),'Night':_(u'Night'),
                u'EOD':_(u'End of day')}
```

Figure 9: Dictionary of natural language time strings

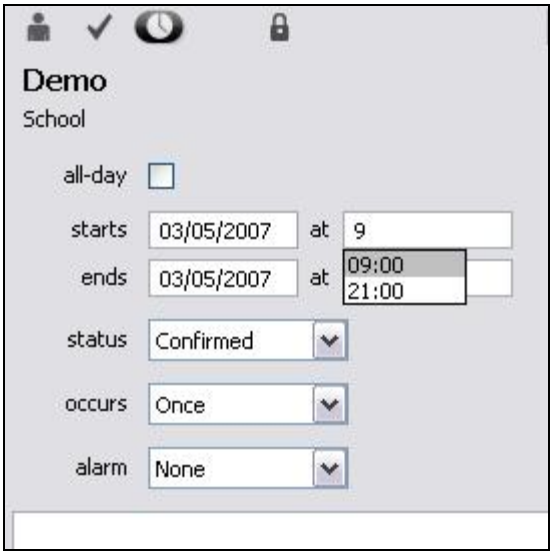
When the user enters some text in the native language, the values are looked up in the dictionary. If a match is found, the key, which is the English translation of the text, is then given to ParseDateTime for parsing. The entire dictionary is searched for more possible matches. These natural language date/time strings present in the dictionary are not used in forming any regular expressions. The text is directly compared with the values in the appropriate dictionary using string comparison in Python. If the text is entered in the date or time widget, the dictionaries of natural language dates or times are searched respectively for matches.

The International Components for Unicode (ICU) is a mature, widely used set of C/C++ and Java libraries for unicode support and software i18n. PyICU [45] is a swig Python wrapper for ICU's Application Programming Interface (API) developed and maintained by OSAF. It is used to format dates, times, numbers and currency according to the format specified for the current locale.

An example of the French localization [46] can be seen in Figure 10 below. The French date format is dd/mm/yyyy in contrast to the US English date format mm/dd/yy. When the user types 'a' in the start date widget in the Detail View of an Event in Chandler, a drop down list with two options "août : 01/08/2007" and "avril : 01/04/2008" appears as depicted in Figure 10(a). In French, "août" stands for the month name August. Hence the date is parsed to 'August 1'. Chandler always shows a future date during auto-completion. In this case, the year is 2007 since August 1 is yet to come in this year. So the date corresponding to 'août ' is 01/08/2007. In French, "avril" denotes month name April. Since the date April 1 is already passed in the current year, April 1 of the next year i.e. 01/04/2008 is the parsed date for

“avril”. Figure 10(b) shows the parsing of time in the start time widget in the Detail View. In French locale, meridian information is not used in time. So time is represented in a 24- hour format. When a user types ‘9’ in the widget, the options in the dropdown list are “09:00” and “21:00” denoting “9:00 AM” and “9:00 PM” in the US locale respectively. Thus the dates and times are customized for the French Locale when Chandler is launched in French. Thus following the i18n and l10n standards helps in localizing a software for different markets.

(a)



(b)

Figure 10: Date and time widgets in the Detail View localized in the French locale

3 Parsing of Items for Date/Time Information

Items in Chandler can be of various Kinds such as emails, notes, tasks and calendar events as mentioned in Section 1.1. Stamping is used to modify the Kind of the Items [16]. For example, a simple Note can be stamped as Task or Calendar Event or Email Message or a combination of these Kinds. If a Note is stamped as a Calendar Event, start date and time, and duration are added as additional attributes. Adding a Message stamp would add To, From, Cc, Bcc and Subject attributes. Stamping can be done by using the stamping buttons in the Detail View marked by the dotted red circle in Figure 11 below. The three icons are for stamping the Item as Message, Task and Calendar Event respectively.

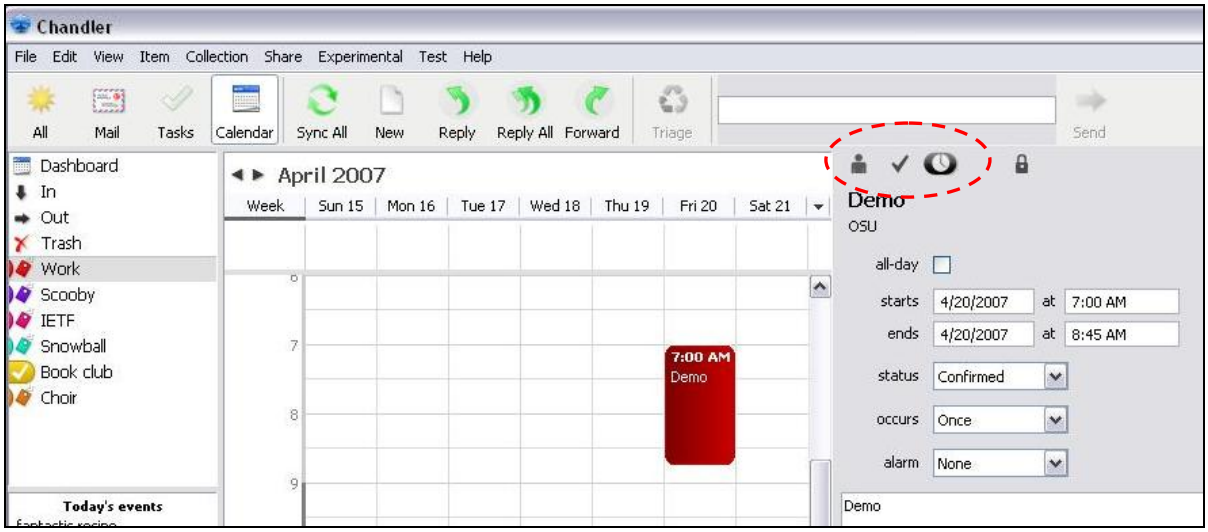


Figure 11: Stamping Buttons in the Detail View of Chandler

If an Item already marked as a Task is stamped again using the Task icon, the item is unstamped i.e. it is no longer a Task. The same is true for Mail Messages and Calendar Events. After unstamping, if the Item has no Kind, it becomes a generic Note Item. Stamping of an Item as a Calendar Event is frequently done to add various tasks and emails on the Calendar.

The goal of this project is to parse the contents in the body of the Items for date/time information when they are stamped as Calendar Events [25]. If any date/time information is found, the start and the end date/time of the Calendar Event are set accordingly.

3.1 Existing Working of Chandler

Prior to my working on this project, when an Item was stamped as a Calendar Event, the start and the end date of the Item would be set to the current date i.e. the date on which it is stamped [47]. No start/end time is specified for the Item, thus making it an any-time event. Any-time events do not have a specific start or end time, but are of short duration that could occur "any-time" during that day. The body of the item was not parsed at all for any date/time information.

For example, Figure 12(a) below shows a Task whose body has some time information. On stamping it as a Calendar Event, more attributes like start and end date/time are added to the Task as seen in Figure 12(b). The start and end date is set to 4/24/07 which is the day the task is stamped as an Event. The Item is made an any-time event by not specifying any start and end time as indicated by the blue dotted circle. The event is not set at 9 AM which is the time specified in the body of the Item. The any-time events can be seen just below the toolbar and above the time slots for the particular day as marked by the red dotted circle in Figure 12(b).

Figure 12(a): Detail view shows a task whose body has some time information

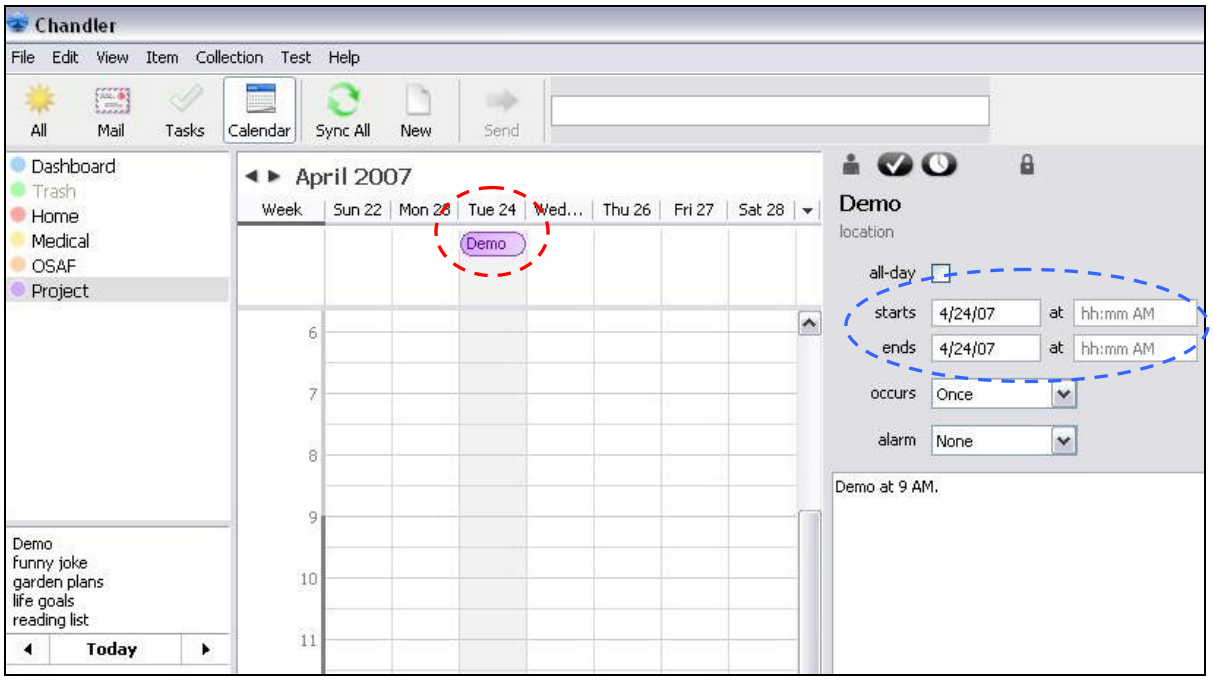


Figure 12(b): When the task in Figure 12(a) is stamped as a Calendar Event, the start date of the event is set to the current date and it is made an any-time event.

3.2 Drawbacks of Existing Working

The drawback of existing approach was that the date/time information specified in the body of the Items was not used for placing the Item appropriately on the Calendar when stamped. This can be misleading to the user who might be intending to put the item somewhere else on the Calendar depending on the date and time information present in the body of the Item. For example, if the user creates a Task with body containing the text “Lunch with Bob on 4/28/07”. If he later wishes to add it to the Calendar on April 24, 2007, he would stamp the Task using the Calendar icon in its Detail view. The Task will be added to the Calendar as an any-time event on 4/24/07 instead of an event on 12pm on 4/28/07. This can be very confusing and can lead to loss of very valuable information. Hence it is necessary to parse the body of the Items for date/time when they are stamped as Calendar Events.

3.3 Desired Working

When an Item is stamped as a Calendar Event, the contents in the body of the Item should be parsed for date/time information [48]. The start and end date/time of the Event should be set according to the rules mentioned below:

- If only one date is found, set the start and end date of the event to the parsed date and no start and end time should be specified making it an any-time event on that day.
- If only one time is found, set the start and the end date of the event to the current date and the start and the end time to the parsed time.
- If one time-range is found, set the start and the end date of the event to the current date and the start and end time to the parsed start time and parsed end time. For example, if the text is “Meeting 2 pm – 3:30 pm”, the start time of the event should be 2pm and the end time should be 3:30pm.
- If no date/time is found, by default the start and the end date will be the current date and there will be no start and end time making it an any-time event on that date.
- If more than one date or times are found, ignore all date/times and set the event to its default date which is current date and default time which is any-time.

Appropriate message should be displayed in the Status bar of Chandler to inform the user of the action taken to set the Calendar Event’s date and time. This will guide him to make any changes if he wishes.

3.4 Implementation

ParseDateTime can be used to parse the body of the Item when it is stamped as a Calendar Event. But ParseDateTime returns only time-tuple. Therefore it cannot be used to parse date

or time ranges such as “March 24 – April 2” and “2 – 5 pm” since it will just return the date/time that comes latter in the text. So we won’t have separate start and end dates/times. Hence we added this functionality to ParseDateTime, which helped it to parse date/time ranges and return two time tuples instead of one. Changes made to ParseDateTime are discussed in Section 3.4.1. We also made changes to Chandler to implement parsing of item bodies for date/time information and to accommodate changes done to ParseDateTime. This is explained in Section 3.4.2.

3.4.1 Changes to ParseDateTime

ParseDateTime should be able to parse the widely used formats of date and time ranges and return the starting time and ending time of the range. In order to do this, regular expressions were written for these formats for date and time ranges. Following are the different formats for date and time ranges that have been implemented:

(i) Starting date and ending date are both in mm/dd/yy format. For example: "06/07/06 - 08/09/06".

(ii) Starting date and ending date both have month names in words. For example: “march 31 - june 1, 2006"

(iii) Starting date has month name in words and the ending date implicitly assumes the month name of the starting date. For example: "march 1st -13th"

(iv) Starting time and ending time both have meridian information. For example: "4:00:55 pm - 5:20:44 pm"

(v) Starting time and ending time both do not have meridian information. For example: "4:00 - 5:30"

(vi) Ending time has meridian information and starting time implicitly assumes the meridian of the ending time. For example: "4-5pm"

We added a method evalRanges() in ParseDateTime which evaluates the text entered by the user and determines if it has a date or time range by matching it with regular expressions for the various date/time range formats. If the text matches any regular expression, it means that a date or time range is found. The code in Figure 13 below shows the working of evalRanges() for the time range format (iv) mentioned above.

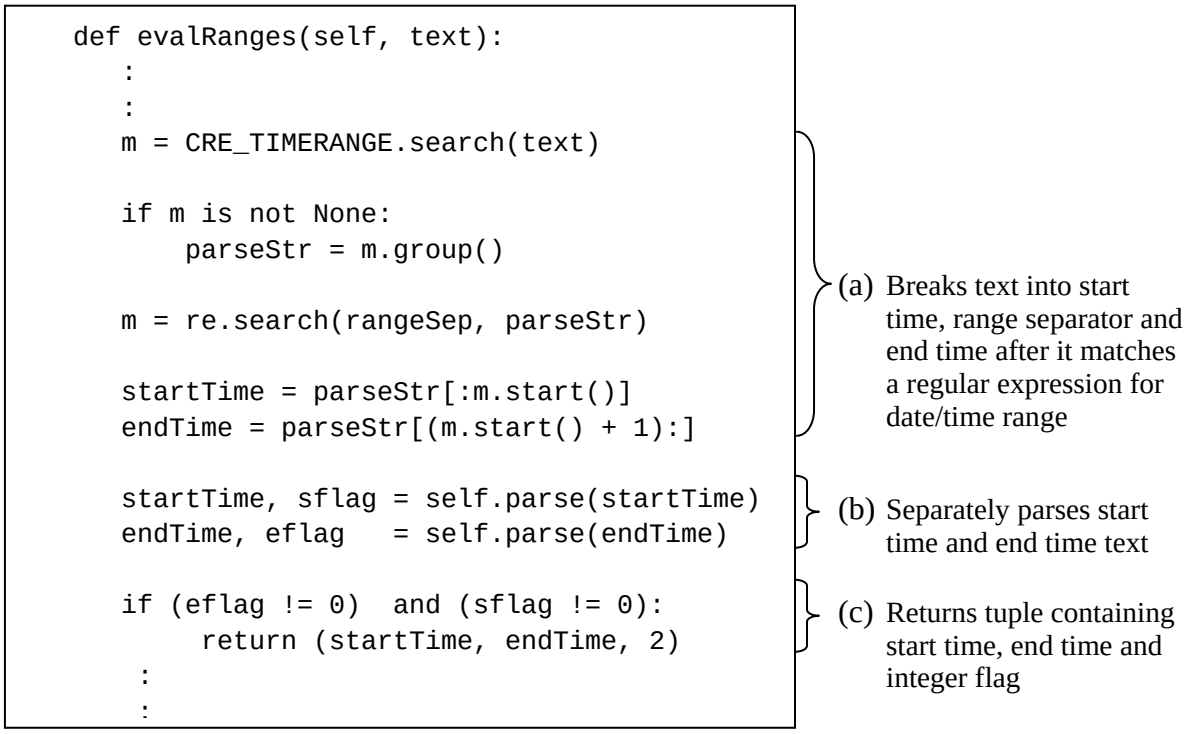


Figure 13: Code snippet for parsing a time range using a method evalRanges() in the ParseDateTime module

The regex CRE_TIMERANGE, in Figure 13, is obtained after compiling the regular expression RE_TIMERANGE which is given by:

RE_TIMERANGE = RE_TIME + r'\s?-\s?' + RE_TIME

where RE_TIME is the regular expression for simple time format hh:mm am/pm as shown in Figure 5 in Section 2.5.1.2.

If the text entered by the user is “2 pm – 5 pm”, then it will match the regular expression RE_TIMERANGE. The text will be then split into startTime, range separator and endTime which will be “2 pm”, “-” and “5 pm” respectively. This is done by Part (a) of the code snippet in Figure 13.

The startTime and endTime are then parsed separately using the method parse() as shown by Part (b) in Figure 13. The method parse (), used to parse single date/time information, returns a time tuple and a typeFlag which is an integer having value 1 or 2, if the parsed information is a date or time respectively. The method evalRanges() then returns a tuple of the form (time tuple, time tuple, typeFlag) denoting start date/time, end date/time and the type of parsed information respectively as shown in Part (c) of Figure 13.

Working of evalRanges() for complex date and time formats is slightly different. If the text matches the regex for any date/time format, it is spit into start date/time, range separator and end date/time. Before parsing them using the method parse(), we need to make sure that both start and end date/time are in proper date/time formats. In order to do this, we need to share some information between them as explained by the cases below. If either start date or end date has year information and the other does not, the year is extracted from the text and appended as a suffix to the date without year information. For example, if the text entered by the user is “march 31 - june 1, 2006", the start date will be “march 31” and the end date will be “june 1, 2006”. The year information “2006” is extracted from the end date and appended

as a suffix to the start date making it “march 31, 2006”. For special date range format (iii) above, the month name in the start date is extracted and is appended as a prefix to the end date. Similarly, for the special time range format (vi) above, the meridian information is extracted from the end time and appended as a suffix to the start time. Now the start date/time and end date/time are exactly in the same format. Hence, it can be seen that several cases of the date/time range formats, given by (i) through (vi) above, are possible. Various examples of these cases are shown in Table II below along with the appropriate modifications made to start date/time and end date/time within the method evalRanges().

Original Text	**Start date/time**	**End date/time**	**Modified Start date/time**	**Modified End date/time**
March 31 – May 1	March 31	May 1	Same	Same
March 31 – May 1, 2007	March 31	May 1, 2007	March 31, 2007	Same
March 1st – 31st	March 1st	31st	Same	March 31st
March 1st – 31st, 2007	March 1st	31st, 2007	March 1st, 2007	March 31st, 2007
03/01 – 05/01/07	03/01	05/01/07	03/01/07	Same
2 pm – 4 pm	2 pm	4 pm	Same	Same
2 – 4 pm	2	4 pm	2 pm	Same

Table II: Examples of strings with date/time ranges and the modifications done to start date/time and end date/time before parsing. The entry “Same” indicates that no modifications are required.

Once the start and end date/time are in the same format, they can be parsed using the method parse() in ParseDateTime. The method evalRanges() then returns a tuple of the form (time tuple, time tuple, typeFlag) denoting start date/time, end date/time and the type of parsed

information, respectively, as explained earlier. The typeFlag is an integer having value 1 or 2 depending on whether the parsed text is a date or a time range, respectively.

ParseDateTime currently recognizes only the above-mentioned formats for date/time ranges. Any other format would not be recognized and would result in an error. For example, if the text is " Meeting from 2 pm to 5 pm", the time range '2pm – 5pm' in not recognized since currently there is not support for ranges with 'from' and 'to' in them. Also, natural language time strings like "lunch" are not considered as a time range. For example, the string "Lunch with Bob" would result in a single time 12:00 PM and not a range 12:00 PM to 1 PM. If the starting date/time and ending date/time are in different sentences, the date/time range will not be recognized. This is because semantic parsing of the text is not done. Syntactic parsing matches the text with the given regex patterns only without understanding the meaning of the text. For example, if the text is "The workshop will begin on April 5. It will end on April 9.", the information about the date range 'April 5 – April 9' will not be extracted from the text.

3.4.2 Changes to Chandler

The way in which stamping an Item as a Calendar Event works needed to be modified to allow the parsing the body of the Item for date/time information. If no date/time or multiple dates/times are found in the body, the Event is set to the default date/time. If only one date/time is found, the Event is set to that date/time.

In order to modify the working of Chandler, we added a method parseText() which is called when an item is stamped as a Calendar Event. The input to the method parseText() is the text in the body of the Item being stamped. This text is split into sentences using the method

text.split('.') in Python. Currently, sentences are not split at '?' or '!'. So, if sentence ends with either a '?' or '!', it is not considered as the end of the sentence and the text is split at the next occurring '.' or end of text. For example, the text, "Are you free tomorrow? Can we meet at 9 am.", is considered as one sentence only when the splitting is being done. Each sentence is then parsed using the modified standalone module ParseDateTime discussed in Section 3.4.1 above.

The output of ParseDateTime is a tuple of the form (startTime, endTime and typeFlag). To keep a track of the number of occurrences of various dates/times in the text, an integer variable countFlag is used. The variable countFlag is incremented whenever ParseDateTime returns a valid output. If the countFlag has value 2, it indicates that multiple date/times are present in the text. Hence, the remaining text is not parsed since we have to set the Calendar Event to the default date/time in this case. If no date/time is present in the body of the Item, the countFlag is never incremented and has a default value of 0. If only one date/time is found in the entire text, the countFlag will have value 1 since it is incremented only once. Now the method parseText() returns the tuple (start date/time, end date/time, countFlag, typeFlag). Depending on the countFlag and the typeFlag, the Calendar Event is then set to the appropriate date and time as described in Section 3.3 above. Now to notify the user of the action taken by Chandler we need to update the message in the status bar. Depending on the value of the countFlag (0,1, and 2), the messages "No date/time found", "Event set to the date/time found" and "Multiple date/times found" are displayed respectively in the status bar. This helps the user to identify the date and time at which the Event was set when he stamped the Item and put it on the Calendar. The steps taken after the item is stamped as a Calendar

Event are shown in Figure 14. The way in which the startTime and the endTime of the Event are set can also be seen.

```
if stampClass == Calendar.EventStamp:
    # If the item is being stamped as CalendarEvent, parse the body of
    # the item for date/time information

startTime, endTime, countFlag, typeFlag = Calendar.parseText(item.body)

statusMsg = { 0:_(u"No date/time found"),
              1:_(u"Event set to the date/time found"),
              2:_(u"Multiple date/times found")}

# Set the appropriate status message in the status bar
wx.GetApp().CallItemMethodAsync("MainView", 'setStatusMessage',  \
                                                 statusMsg[countFlag])

# Set the event's start and end date/time depending on typeFlag.
# typeFlag value 0,1,2 and 3 indicates no date/time, only date, only
time and both date and time respectively.

if (typeFlag == 1) or (typeFlag == 0):
    # No time is present
    EventStamp(item).anyTime = True
else:
    EventStamp(item).anyTime = False

EventStamp(item).startTime = startTime
EventStamp(item).endTime = endTime
```

Figure 14: Code Snippet showing the working of stamping an Item as a Calendar Event in Chandler

4 Using Text Entry Widget in the Toolbar for Quick Entry of Items

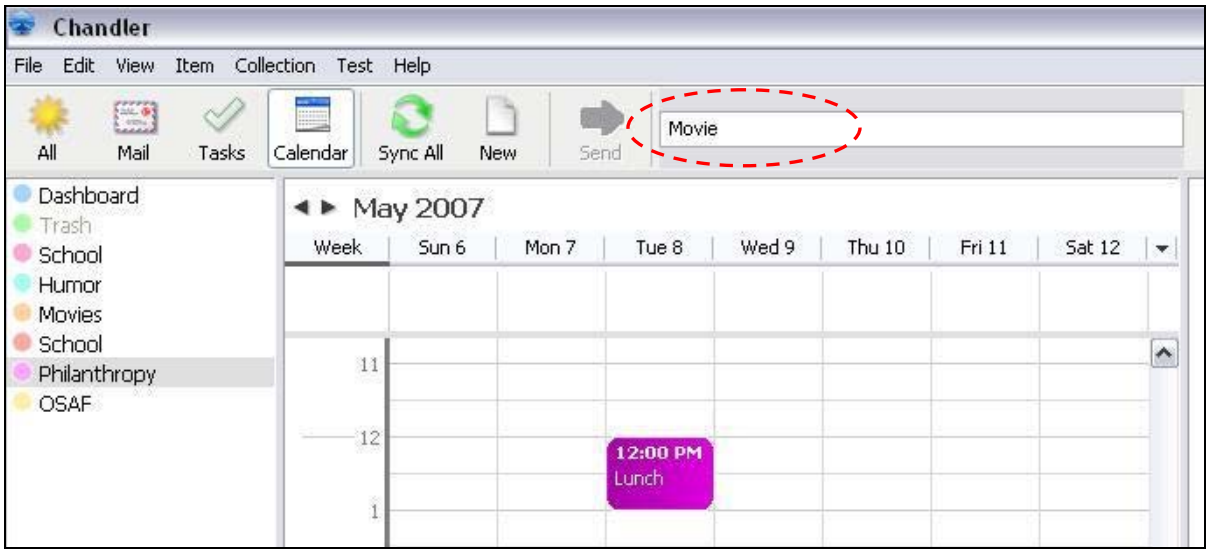
The text entry widget in the toolbar is a handy tool to perform operations without changing the current context. Consider the case in which the user is in the Calendar view of Chandler, and he needs to just add a task quickly without entering many details. Currently, to perform this task, the user normally switches to the Task View, creates the task and then switches to the Calendar View again to resume his work. This workflow takes a lot of effort, and decreases the ease and usability of the software. Also, changing contexts frequently can diminish the productivity of the software.

Making the text widget in the Toolbar work as a 'Quick Item Entry' box or Command Line Interface (CLI) helps in solving this problem [49]. Irrespective of the View the user is in, he/she can use the text entry widget in the Toolbar as a command line to create an Item quickly by entering a command and a few details of the Item. This new Item should be saved in the appropriate Collection. This is done without changing the context or View the user is in i.e. if the user is in the Calendar View, he can create a task using the CLI without changing his View. The user can later edit this Item's details to add more information if necessary. The user should be able to create other Items like Email Messages, Calendar Events, Notes, Tasks, Invites and Reminders using the CLI. This text entry widget should also be used a search box. The goal of this project is to allow the text entry widget in the toolbar to be used for quick entry of Items [26].

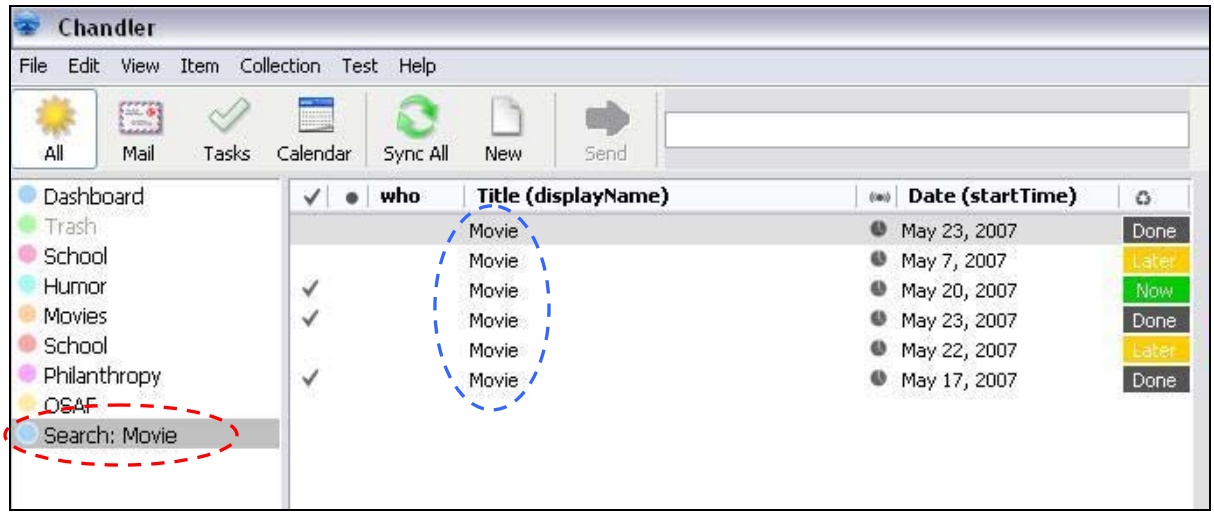
4.1 Existing Working of Chandler

Before I started working on this project, the text entry widget in the Toolbar of Chandler was used only as a search box. For example, if the user wanted to search an Item in the Chandler repository with the word 'Movie' in any Attribute of the Item, he would simply enter

"Movie" in the text entry widget in the Toolbar as marked by the red dotted circle in Figure 15(a). All the items were searched and the results were put in a new Collection called 'Search: Movie' which can be seen in the Sidebar View of Chandler as marked by the red dotted circle in Figure 15(b). All the Items in this Collection i.e. the Items with the word 'Movie' can be seen in the Summary View as marked by the blue dotted circle in Figure 15(b).



(a)



(b)

Figure 15: Use of the text entry widget in the Toolbar for searching Items in the Chandler repository

The drawback of this working was that the text widget had very limited functionality. This simple text widget being used as a search box could be transformed into a very powerful CLI by adding several commands such as commands for quick entry of Items in Chandler.

4.2 Background

In order to comprehend the process of quick entry of Items, the concept of the Dashboard [50] in Chandler must be understood. The Dashboard provides at-a-glance display of all the data pulled from the repository of Chandler. It spans all Kinds of information. All Items in each user-defined Collection are also presented in the Dashboard view. Dashboard helps in triaging the Items depending on their status. This helps to organize and prioritize Items. If a user is unsure about which Collection to put an Item in, he can put it in the Dashboard and deal with it later.

The Dashboard View is shown in Figure 16. The red dotted circle depicts that ‘All’ Items option is selected in the Toolbar and the Dashboard collection is selected in the Sidebar. All the items in the Chandler repository are listed in the Summary View in the middle. Notice that the Items are partitioned into 3 sections depending on their Triage status: ‘Now’, ‘Later’ and ‘Done’. The Triage status can be modified in the Dashboard. The list of Items can be sorted and rearranged based on the different Kinds and Attributes displayed in the Dashboard. The Detail View on the right displays the details of the Item selected in the Summary View.

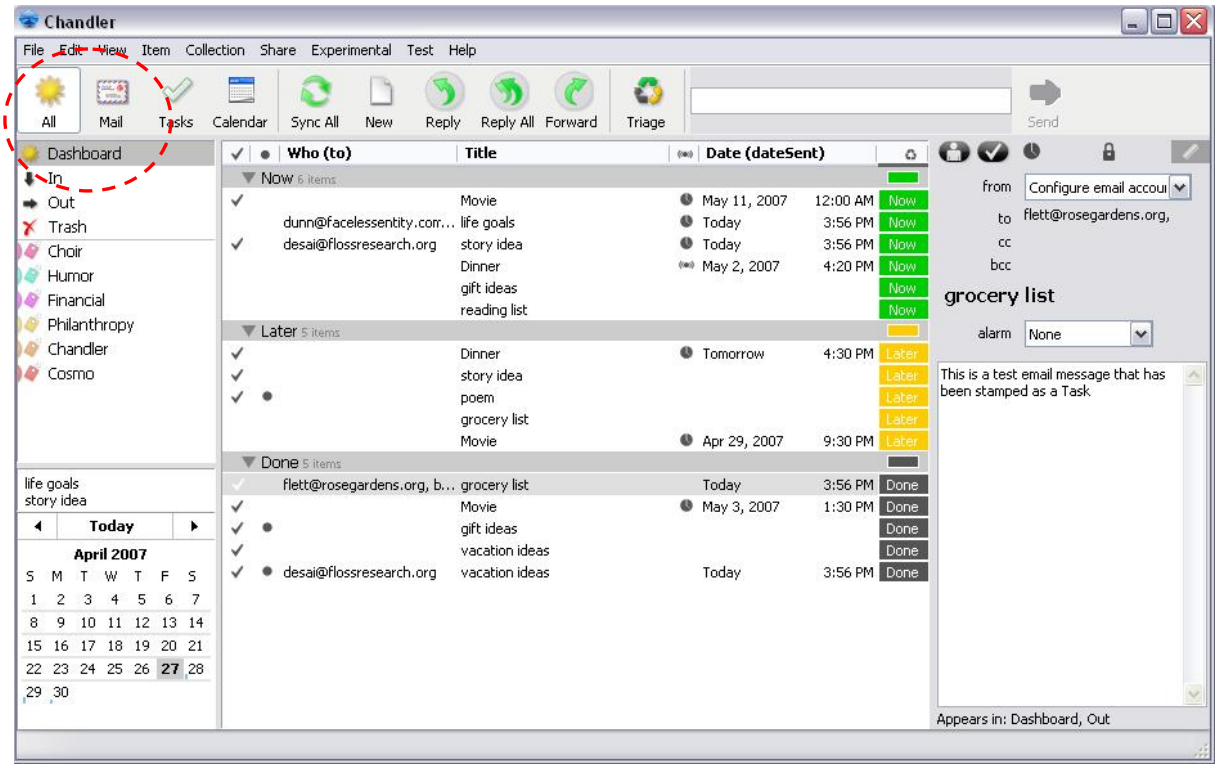


Figure 16: Dashboard View of Chandler

4.3 Desired Working

The text widget in the toolbar should allow accept commands to create all Kinds of Items such as Tasks, Notes, Messages, Invites, Calendar Events and Requests. The search functionality of the text widget should be retained. The commands [26] to perform these tasks are as follows:

- /Note *{body}*: Creates a Note
- /Msg *{body}*or /Message *{body}*: Creates an email Message
- /Task *{body}*: Creates a Task
- /Event *{body}*: Creates a Calendar Event
- /Invite *{body}*: Creates an Iitem which is stamped as a Message and a Calender Event both

- /Request *{body}*: Created an Item which is stamped as a Message and a Task both
- /Search *{body}*: Searches for the *{body}*
- /(Invalid cmd): "/(Invalid cmd) ? " should appear in the text widget in the Toolbar
- (Some text): The default Item of the current View is created and added to the currently selected user-defined Collection.
- **Multiple** commands also can be entered. For eg: "/Msg /Task *{body}*" will create an Item which is stamped as a Message and a Task both.

The *{body}* in the command becomes the 'Subject' Attribute or heading of the newly created Item.

The newly created Items are always put into the Dashboard collection; unless the Kind of the Item matches the option or View selected in the Toolbar [51]. In that case, the new item is placed in the currently selected user-defined Collection. For example, if the user is in the Calendar View of the 'Home' collection and he/she creates a Calendar Event using the CLI, then the new Event created should be placed in the Calendar of the 'Home' Collection only and not in the Dashboard. If the user creates a 'Task' while being in the Calendar View, the new Task should be placed in the Dashboard Collection.

If the user enters text in the command line which has a date/time range in the *{body}*, the new Item is automatically stamped as a Calendar Event, whether the user explicitly says "/Event" or not. For example, the command "/Msg Meeting from 2-4pm " will create an Item which is stamped as a Message and a Calendar Event both. The startTime and endTime of the Item will be set to 2:00 PM and 4:00 PM respectively.

If the user creates an Item (other than a Calendar Event) with a single date/time in the *{body}*, a Reminder is set at that date/time. If only the date is mentioned and no time is specified, the Reminder is set at the custom time of 5:00 pm. If the user enters "/Event" with a single date/time, a Calendar Event is created and the startTime and the endTime of the Event are both set to the specified date/time. Such Events are also called as ‘at-time’ Events in which the duration of the Event is zero.

Valid Email addresses should also be parsed from the *{body}* of the command ‘/Msg’ or ‘/Message’. These addresses are put in the ‘To’ attribute of the Message Item. The text appearing after the email addresses becomes the ‘Subject’ Attribute. For example, if the user types in the command ‘/Msg bob@orst.edu, jack@orst.edu: group study’ will create a Message with ‘group study’ as the ‘Subject’ and ‘bob@orst.edu, jack@orst.edu’ as the ‘To’ Attributes. If no email addresses are parsed, the entire *{body}* becomes the ‘Subject’ of the Message Item.

The existing mechanism for Search’ should not change except that the command ‘/Search’ should be entered explicitly followed by the text to be searched. The entire repository should be searched and a new Collection should be created which includes all the Items, which result from the search query. This helps in saving the searches for further use.

4.4 Implementation

The working of the text entry widget was changed in order to implement the CLI for quick entry of Items. In order to provide support for the various commands mentioned in Section 4.3 above, a method parseCommand() was added to Chandler. The command or query

entered by the user is given as an input to this method. The command and the body of the command are separated. The commands are matched with the list of allowed commands. If the command is to create a new Item, the body of the Command is parsed using ParseDateTime to check for any simple dates/times or any date/time ranges. If a simple date/time is found and the Item to be created is not a Calendar Event, a Reminder is set automatically on the parsed date/time. If a date/time range is found, the Item is stamped as Calendar Event irrespective of the Kind of Item created and the start date/time and end date/time Attributes are set to the parsed dates/times. Depending on the command the user enters in the CLI, various tasks are performed as explained with examples below.

If the user enters the command '/Search *{body}*', a search is performed for the given text in the *{body}* using the method searchItems(query). For example, if the user wants to search an Item with the text 'Movie', he/she has to now enter the command "/Search Movie" in the text entry widget in the Toolbar. All the items in the Chandler repository are searched and the results were put in a new Collection called 'Search: Movie' which can be seen in the Sidebar View of Chandler. All the Items in this Collection i.e. the Items with the word 'Movie' can be seen in the Summary View. The results of the Search are displayed exactly like in Figure 15(b) above, since the working of search has not been modified.

If the user enters an invalid command, a '?' appears in the end. For example, if the query is '/foo abcd', then '/foo abcd ?' is displayed in the CLI. This is highlighted by the red dotted circle in Figure 17. A status message "Command entered is not valid" appears in the Status Bar at the bottom, to prompt the user to rectify the command and repeat the search. This is marked by the blue dotted circle.

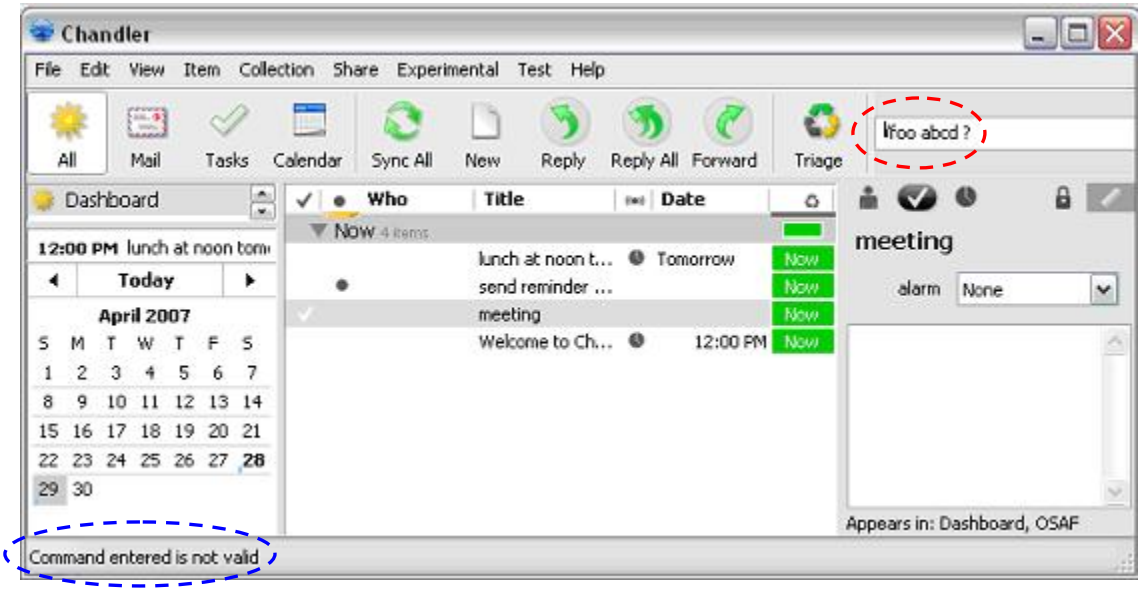
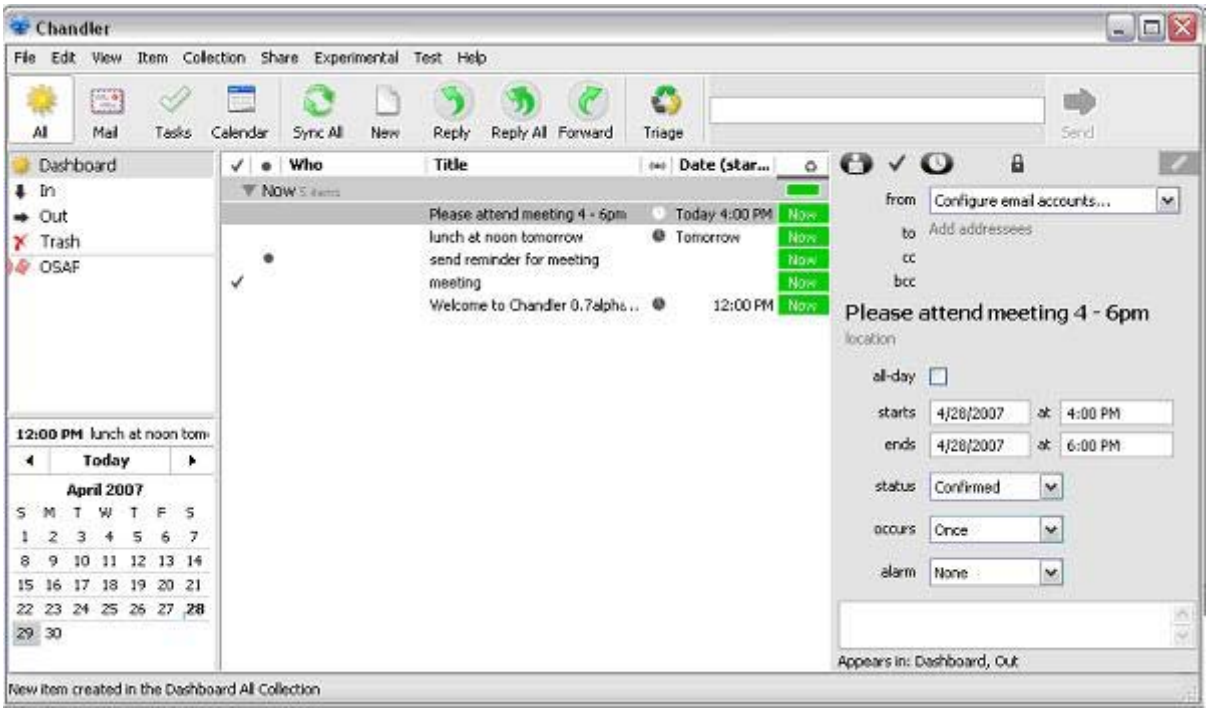


Figure 17: Working of an invalid command entry in the text widget (CLI) in the Toolbar.

If the user creates an Item that is different from the default Item of the current View Chandler is in, the Item is placed in the Dashboard Collection and not in any user-defined Collection. For example, let the user be currently viewing Tasks and he enters the command "/Message Please attend meeting 4 – 6pm" to create a Message Item using the CLI as shown in Figure 18(a). Then a Message Item is created without changing the context or the view, the user is in. Since the user was viewing Tasks and the Item created is not a Task but a Message, the new Item is placed in the Dashboard. The status message "New item created in the Dashboard Collection" informs the user of the action taken. Since the body of the Item has a time range 4 – 6pm, the Item is also stamped as a Calendar Event though the user did not explicitly mention it. The startTime and endTime attributes of this Item are set to 4:00 PM and 6:00 PM respectively on the current date. Since the user did not specify any valid email addresses in the command, the To address attribute of the Item is not set and hence is the default value null. All these details of the Item can be viewed in Figure 18(b).

(a)

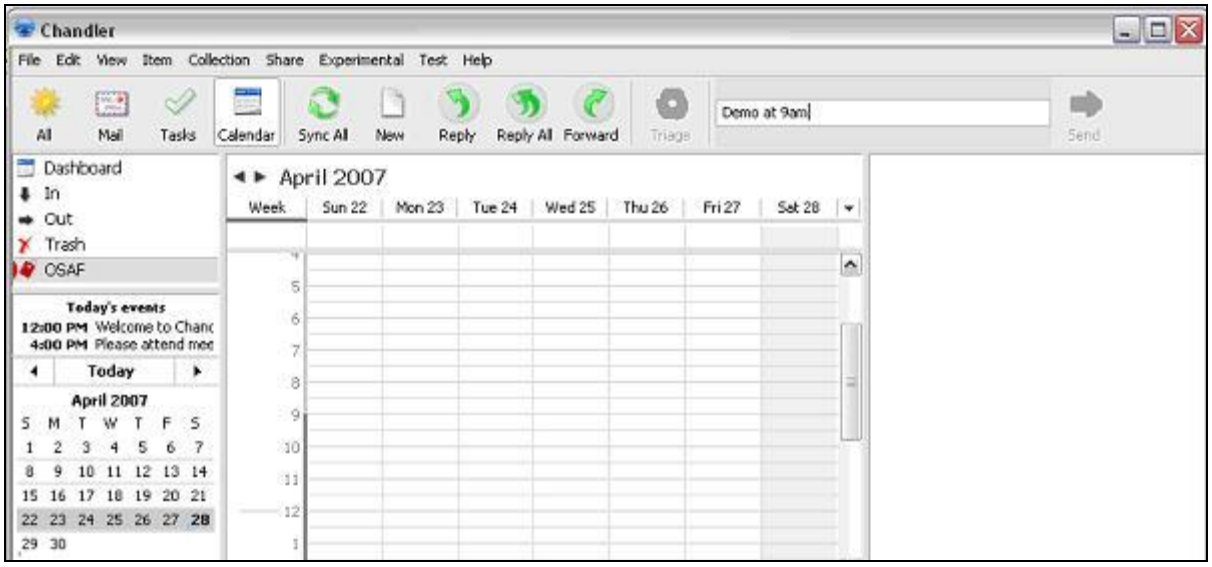


(b)

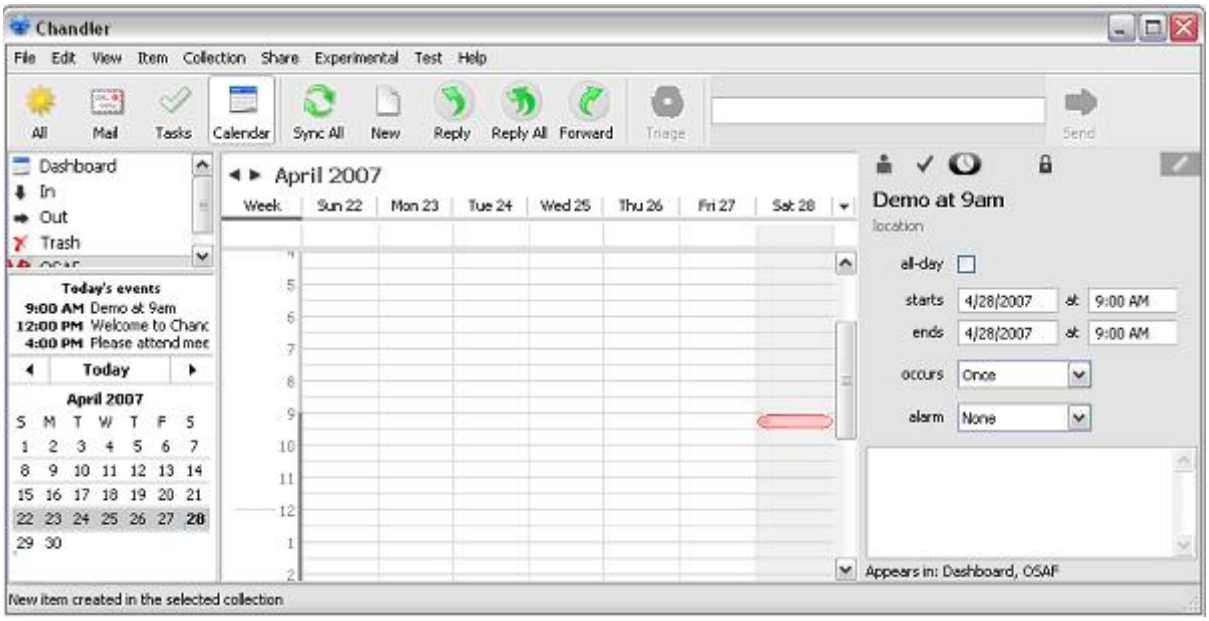
Figure 18: (a) Creating a Message by entering a command using the CLI (b) Details of the new Message Item created in the Dashboard can be seen in the Detail View.

If the user enters text in the CLI without entering a command starting with '/', then an Item of the default Kind will be created and placed in the Collection the user is in. For example, let the user be in the Calendar View of a user-defined Collection 'OSAF' and he enters text "Demo at 9am" in the CLI as shown in Figure 19(a) below. On executing the command, a Calendar Event will be created and placed in the OSAF collection. The status message "New

item created in the selected Collection" appears in the Status bar. The body of the Item will be parsed using ParseDateTime and since a time 9am is found the startTime and endTime attributes will be set to 9:00 AM and the date will be the current date. The new Event can be seen in the Calendar View in Figure 19(b) along with the details of the Event in the Detail View.



(a)



(b)

Figure 19: Item of the default Kind is created and placed in the selected Collection

Thus, the CLI is used to create all Kinds of Items and also to search Items in Chandler without the need to change the current View. The Items are placed in appropriate Collections and also parsed for date/time information using ParseDateTime to set Reminders and stamp as Events.

5 Testing and Bugs in Chandler

After each of the three projects was implemented, the entire test suite of Chandler was run. This test suits included units tests, integration tests and regression tests. These tests implemented both “white box” and “black box” testing techniques [52]. The Design Team and the developers at OSAF tested the projects for boundary-value cases and functional requirements manually. All the bugs that were found during testing were logged using Bugzilla [53], which is a software for bug-tracking and project management. Each bug goes through a lifecycle as described below.

When a user reports a bug, Bugzilla assigns a unique identification number to each ‘New Bug’ filed. The part or code of the project affected by the bug is identified. The developer in charge of that code is assigned the bug and is sent an email notification of the bug. A screenshot of such a bug notification is depicted in Figure 20.

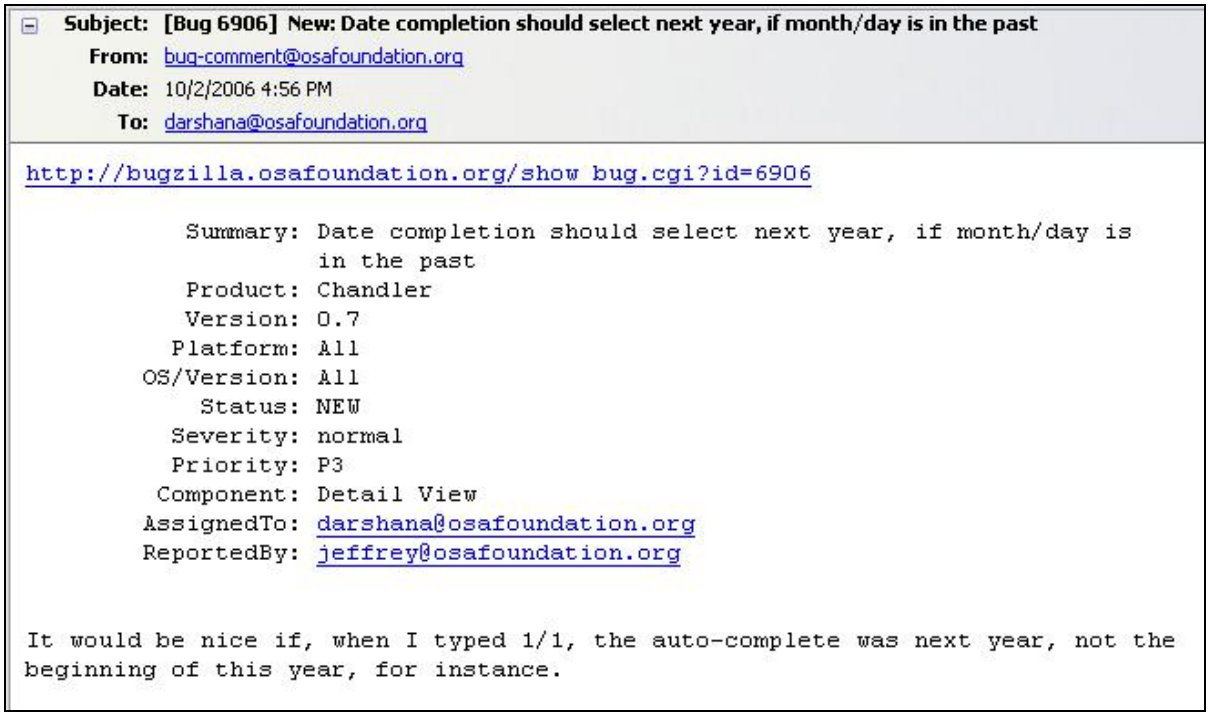


Figure 20: Screenshot of an email notification send to a developer when a new bug is filed in Bugzilla

As seen in the Figure 20, each bug is assigned several attributes, so that it is easy to classify, track and manage it. The Summary gives a short description of the bug. The Product and its Version are mentioned. The Platform and O/S version specify the configuration on which the bug was encountered. Severity specifies whether the extent of damage the bug does. Severity can be Critical, Normal, Trivial and Enhancement. Status denotes the current stage in the lifecycle of the bug [54], which is shown in Figure 21.

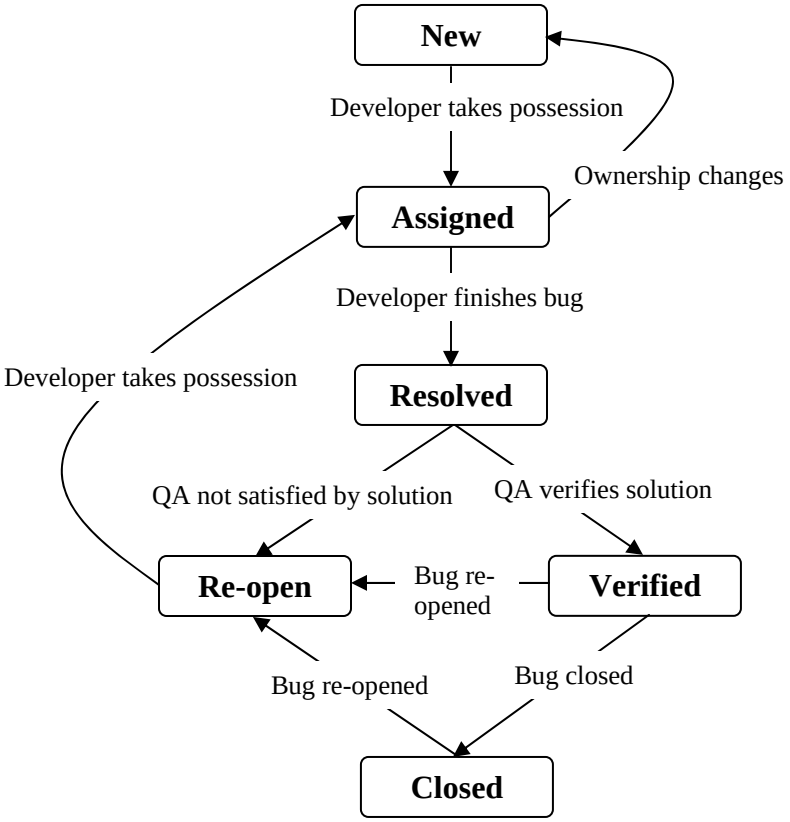


Figure 21: Lifecycle of a typical bug at OSAF

Since I did not have direct access rights to apply my patch for the bug in Chandler's main code repository, my mentor would commit my patch after the QA had verified the bug. Several bugs related to my projects were assigned to me. A few bugs that are either Normal or Enhancement bugs are discussed further.

5.1 Bug 6381: Make the drop down display the value it's selection will result in

The start date/time and end date/time widget in the Detail view provide support for auto-completion of recognized dates and times. When the user types anything in these widgets a drop down list of all the possible date/time options is displayed. Initially, only the strings were displayed and not the corresponding date/time. For example, if the user typed ‘W’ in the date widget, the option ‘Wednesday’ would show up in the drop-down list but not the date it corresponds to. But to help the user choose from the dates, it was requested as an enhancement feature to also display the value of the matched text in the drop-down list [55]. So, instead of just ‘Wednesday’, ‘Wednesday: 05/09/07” should be displayed.

In order to implement this, when the list of possible matches is found, each choice in the list is parsed using ParseDateTime iteratively and the corresponding dates/times are found. The dates/times are then converted to the proper format in the current locale and appended to the matched text. A snippet of code implementing this is shown in Figure 22.

```
# textMatches is a list of natural language date strings
# target is the text entered by the user in the text widget

for matchKey in textMatches:
    if (textMatches[matchKey]).startswith(target):
        cal = parsedatetime.Calendar()
        (dateVar, invalidFlag) = cal.parse(matchKey)
        #invalidFlag is 0 if the string cannot be parsed successfully
        if invalidFlag != 0:
            dateStr = shortDateFormat.format(*dateVar[:3])

            # append matchKey to its value
            matchKey += " : %s" % dateStr
            yield matchKey
```

Figure 22: Code snippet showing the patch submitted for Bug 6381

After implementing the patch for this bug, the drop-down list for dates with text matches and their values can be seen in Figure 8(a). An example of a drop-down list for time widgets when a user types ‘n’ can be seen in Figure 23.

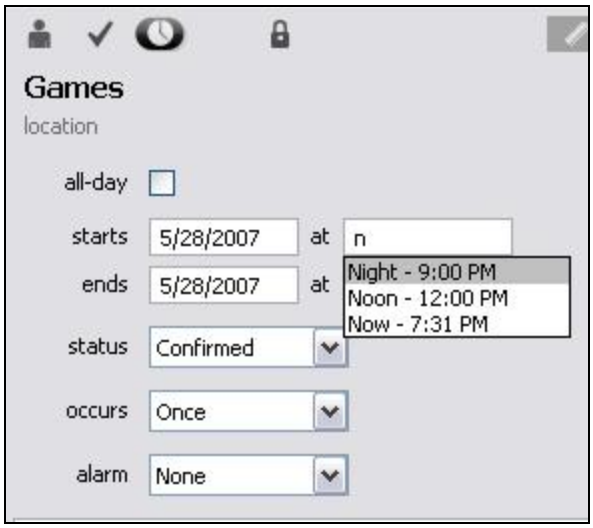


Figure 23: Drop-down list for time widgets displaying the possible text matches and their resulting values

5.2 Bug 6883: Auto-completion does not take place when Tab is hit

When the user types anything in the start date/time and end date/time widgets in the Detail view, a drop down list of all the possible date/time options is displayed. The user can select any of these choices by clicking on them. The user can also use the up and down arrow keys to select the appropriate choice. On hitting Enter, the text in the widget will auto-complete to the date/time in the selected choice. The focus remains on the same widget after auto-completion. On hitting Tab, the text in the widget should auto-complete to the date/time in the selected choice [56]. The focus, however, should move to the next widget after auto-completion.

The working of the start date/time and end date/time widgets, when Tab was hit, was changed. On hitting Tab, it was checked whether any choice in the drop-down list was selected. If such a string was found, it was auto-completed otherwise the focus was moved to the next widget without changing anything. The implementation of the patch for this bug is shown in Figure 24.

```
if keyCode == wx.WXK_TAB:
    # Finish autocompleting, if we have a selection
    if selectionIndex != wx.NOT_FOUND:
        self.completionCallback(self.GetStringSelection())
```

Figure 24: Implementation of the patch for bug 6883

5.3 Bug 6906: Date completion should select next year, if month/day is in the past

When user enters a date in the start/end date widgets in the Detail View of Chandler while editing a Calendar Event or creating a new Calendar Event, the date is mostly in the future. If the user entered a date without the year information, initially ParseDateTime would use the current year to complete the date. This would sometimes cause Events to be created in the past. In order to avoid this, it was suggested that the date be auto-completed to the next year if the month and day have already passed in the current year [57].

For example, when the input is “march 1”, the date is set to future March 1st which is in year 2008 since the date is already passed in the current year. If the input is “12/27”, the date is set to future December 27th which is in this year 2007 since the date has not yet passed in the current year.

The working of ParseDateTime was changed to auto-complete the given incomplete date to a date in the future. If the year information for a date is not given, the month and day are compared to the current month and day. If the given date is past, the current year value is incremented otherwise, the year takes the same value as the current year. This working is implemented as shown in Figure 25.

```
# if the year is not specified and the date has already passed,
# increment the year

if (yr is NULL) and ((currentMth > mth) or (currentMth == mth  \
                                            and currentDy > day)):
    yr = currentYr + 1
else:
    yr  = currentYr
```

Figure 25: Changes made to parsing of incomplete dates in the ParseDateTime module for Bug 6906

The comparison of the working of Chandler before and after the patch for the bug was committed, for the example 'March 1' given above, can be seen in Figure 26(a) and (b).

(a) (b)

Figure 26: (a) Before fixing bug: '3/1' auto-completes to '3/1/07', which is already past. (b) After fixing bug: '3/1' auto-completes to '3/1/08', which is in the future'

5.4 Bug 6223: Auto-completion should not match exact matches

When the user types text in the start date/time and end date/time widgets in the Detail View, a list of possible auto-completion matches appears in a drop-down list. When the user types a string, if any match in the drop-down list exactly matches the string completely, then that match should disappear from the list and only its value should be displayed in the drop-down list [58]. For example, if the user types “lunc” in the start/end time widget, the drop-down list should display “lunch – 12:00 PM”. But as soon as the user types the “lunch” completely, the string matches the auto-completion string perfectly and hence the drop-down list should display only “12:00 PM”. The working should be similar for the start/end date widgets also.

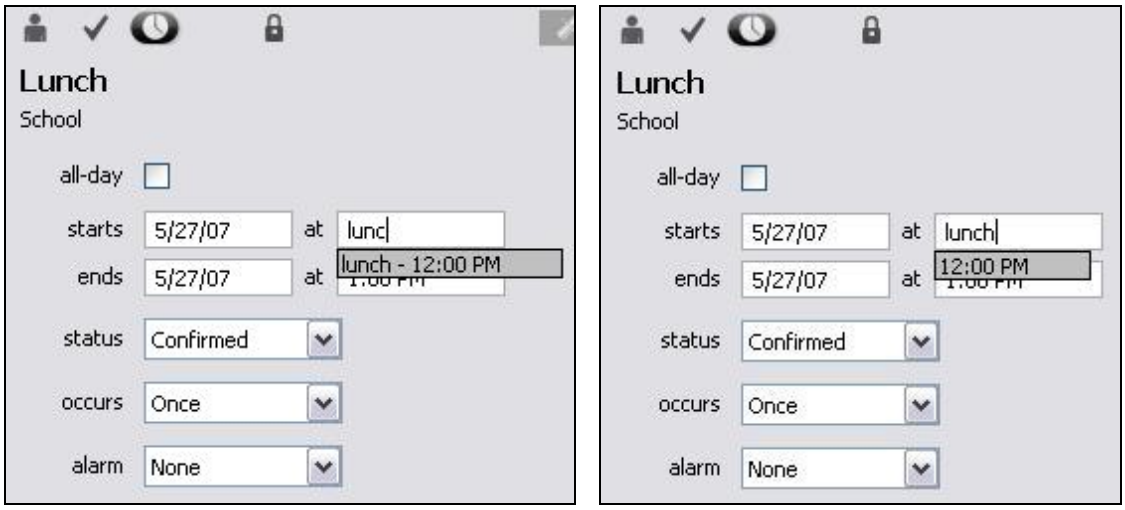
When the dictionaries of natural language dates and time, discussed in Section 2.5.2.2, are searched for possible auto-completion matches, if the text matches an entry in the dictionary exactly, then that match is not yielded. Hence the match is not displayed in the drop-down list; only the parsed date/time value is displayed. The code snippet shown in Figure 27 shows this implementation.

```
# target is the string entered by the user
# match is the text-match found

# display the match only if the target is not exactly equal to the match
if unicode(match).lower() != target:
    yield match
```

Figure 27: A code snippet showing a part of the patch for Bug 6223

The screenshots in Figure 28(a) and (b) depict the corrected working of auto-completion of matches in Chandler when the user types in ‘lunch’ in the start time widget.



(a) (b)

Figure 28:(a) Drop-down list displays the match and its resulting value. (b) Drop-down list displays only the resulting value since the text entered by the user exactly matches the auto-completion string.

5.5 Bug 6567: When entering a weekday, the computed day seems to be off by one

When a weekday is entered in the start/end date widgets in the Detail View, the computed date of the weekday was one day more than the actual date [59]. For example, if the actual date for Sunday was 05/06/2007, then the computed date would be 05/07/2007.

The error was resulting because in the standard time tuple discussed in Section 2.5.1.1, the weekday 'Monday' is considered as 0, whereas in ParseDateTime we were considering Sunday as 0 and Monday as 1. The list of weekdays in ParseDateTime was:

```
Weekdays = [_(u'sunday'), _(u'monday'), _(u'tuesday'), _(u'wednesday'),
_(u'thursday'), _(u'friday)', _(u'saturday')]
```

This resulted in the weekday being wrongly computed. So to rectify this, we reordered the list of weekdays to start with Monday instead of Sunday so the index of Monday would be 0.

The corrected list is as follows:

```
Weekdays = [_(u'monday'), _(u'tuesday'), _(u'wednesday'), _(u'thursday'),
_(u'friday)', _(u'saturday'), _(u'sunday')]
```

Making this change solved the bug and the computed date of the weekday then matched the actual date.

6 Summary

Personal Information Management (PIM) software is used to organize personal information like schedules, contacts, tasks, reminders and emails effectively. Chandler™ is an Inter Personal Management Software which allows users to collaborate, share and synchronize this personal information. Chandler is still in its alpha stages and is being developed at Open Source Applications Foundation (OSAF) situated at San Francisco, CA.

My advisor Dr. Budd and OSAF both collaborate with the Open Source Lab at Oregon State University through which I got an opportunity to do an internship at OSAF. During my internship, I got a chance to work on Natural Language Date/Time Parsing in Chandler.

When I joined OSAF, the date and time widgets in the Detail View of Chandler recognized only one date and time format, which made Chandler very rigid and error-prone. The user could not enter any of the widely used natural language date/time words and formats. This limited the usability of the start date/time and end date/time widgets. My first project was to modify the date and time widgets and enable them to understand and parse commonly accepted natural language text, as well as provide auto-completion support to improve the efficiency and usability of Chandler. Mike Taylor at OSAF and I created a module called ParseDateTime, that parsed different date and time formats and converted them to a standard time tuple identified by Chandler. Localization support was also provided to allow the users in different locales to get the same benefits of auto-completion and flexibility. Thus I completed my first project successfully.

When Items were stamped as a Calendar Event, the date/time information in the body of the Item was lost because such information was not recognized and parsed by Chandler. Hence, the start date/time and end date/times attributes of the Event were always set to the current date and a default time. The goal of my second project was to parse the body of the Item when it is stamped as a Calendar Event and extract the date/time information found in the body and set the start date/time and end date/time for the Item appropriately. The ParseDateTime module created during the first project was used to accomplish this task. The start date/time and end date/time of the Event is set depending on the number of natural language dates and times found in the body of the Item.

For the last part of my internship, I worked on the text widget in the Toolbar of Chandler. The text widget had very limited functionality, just allowing the user to search Items in Chandler repository. The objective of the third project was to convert this text widget to a Command Line Interface (CLI) that can be used to create Items quickly without the need of changing context simply by entering the appropriate commands in the CLI. The command was parsed and the time-related attributes of the new Item like start date/time, end date/time and Reminder Time were set according to the date and time obtained after parsing the text using ParseDateTime. The items created using the CLI would be placed in appropriate Collections depending on the active View in Chandler.

After these projects were implemented, other developers, the Design team and the Quality Assurance team at OSAF did testing. Bugs thus found, were assigned to me. By the end of my internship I was able to fix five of these bugs. Some of these bugs were errors while other were enhancements which other developers had requested.

Natural Language Date/Time Parsing was successfully implemented to improve the usability of Chandler. It increased the functionality and the efficiency of the Chandler, making it a more powerful software.

7 Future Work

Improvements and enhancements are needed in both ParseDateTime module and in Chandler. This section is divided in two sub-sections to distinguish the future work that can be done in these two projects.

7.1 ParseDateTime

ParseDateTime currently does syntactic parsing of text. Hence the number of date and time formats recognized by it is still limited. Implementing semantic parsing of text would help ParseDateTime to recognize the date and time information that is not bound by any specific format. For example, if the user splits a date/time range in two separate sentences, the current version of ParseDateTime would consider them as two different dates/times instead of understanding the relation between them and considering it as one date/time format. Consider the text “The meeting starts at 3 pm. It will end at 5 pm”. Here both the times relate to the same event Meeting. Hence, it should be considered as a time range with start time ‘3 pm’ and end time ‘5 pm’. Also, ranges with words like ‘from’ and ‘to’ should be recognized after implementing semantic parsing.

7.2 Chandler

The CLI in the Toolbar of Chandler can be made more powerful by adding auto-completion support for the existing commands. For example, if the user types ‘/T’ it should be auto-completed to ‘/Task’ since that is the only command starting with ‘/T’. I attempted to implement this, but was unable to do so because of the rigidity of the widgets in the Toolbar. These widgets inherit properties from the Toolbar, so the restrictions on the Toolbar are superimposed on these widgets. The text entry widget in the Toolbar did not allow a drop-

down list to be displayed because of which auto-completion of commands could not be implemented.

To demonstrate the desired working, I developed a prototype of the CLI just for experimental purpose in the Detail View since I had already implemented auto-completion for the text widgets in the Detail View. This temporary CLI was placed right below the Title of the Item in the Detail View. When a user enters '/', a list of all the possible command is displayed as seen in Figure 29.

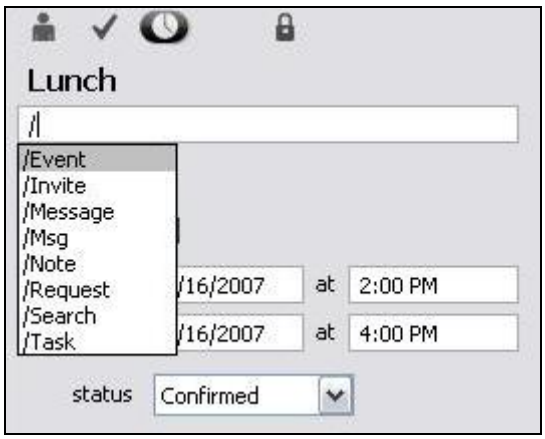


Figure 29: Drop-down list of all command available in the CLI

Also, the user can create an Item, which is stamped as multiple Kinds. If the user enters a command, say "/Invite", then a list of possible commands is displayed when the user enters a '/' again i.e. "/Invite /". This can be seen in Figure 30. Note that only two choices '/Request' and '/Task' come up. This is because an Invite is already a Calendar Event and a Message, and every Item is, by default, a Note. So, the drop down list does not include commands '/Event', '/Message' and '/Note'. Also, '/Search' is not in the list since, '/Invite' suggests that the command is used for quick entry of Items and not to search Items in Chandler. Thus, the

auto-completion list should be continuously be updated depending on what the user enters and the possible commands available at that point.

Figure 30: Item being created being stamped as multiple Kinds using the CLI

The entire Toolbar needed to be revamped to allow this type of auto-completion functionality to be provided in the actual CLI in the Toolbar. Since, it was out of the scope of the project, I did not implement it. This feature would greatly ease the usage of the CLI by showing the possible commands at each stage and hence, should be implemented in the future.

More functionality can be added to the CLI by providing support for more commands that would help create new user-defined Collections and user-defined Items. The commands can be modified to give the user the choice to specify various Attributes of the Items while creating them using CLI. Commands can also be provided to delete, update and synchronize Items.

References

[1] http://en.wikipedia.org/wiki/Personal_information_manager

[2] http://office.microsoft.com/en-us/outlook/default.aspx

[3] http://www.apple.com/macosx/features/ical/

[4] http://en.wikipedia.org/wiki/Open_source

[5] David Frohlich, "Requirements for Interpersonal Information Management", 1995, Hewlett Packard Laboratories Bristol.

[6] http://chandler.osafoundation.org/

[7] http://osafoundation.org/

[8] http://osuosl.org/

[9] http://www.osafoundation.org/Chandler_Compelling_Vision.htm

[10] http://chandler.osafoundation.org/1.0_vision.php

[11]http://en.wikipedia.org/wiki/Python_(programming_language)

[12] Timothy Budd, "An Introduction to Object Oriented Programming, Third Edition", 2002, Addison-Wesley

[13] http://www.wxpython.org/what.php

[14] http://en.wikipedia.org/wiki/WxPython

[15] http://www.python.org/

[16] http://chandler.osafoundation.org/docs/0.6/overview.html

[17] http://www.otd.ou.edu/tech/05TUL030C

[18] http://www.osafoundation.org/0.5/ChandlerGuidedTour.pdf

[19] Ian Sommerville, *Software Engineering Eighth Edition*, 2007, Addison-Wesley

[20] http://en.wikipedia.org/wiki/Model-view-controller

[21] http://ootips.org/mvc-pattern.html

[22] http://wiki.osafoundation.org/Projects/CpiaZeroPointFourStatus

[23] http://wiki.osafoundation.org/bin/view/Projects/AttributeEditorAPI

[24] http://wiki.osafoundation.org/Journal/NLP

[25] http://wiki.osafoundation.org/Journal/ParsingEmails

[26] http://wiki.osafoundation.org/Journal/QuickItem

[27] http://en.wikipedia.org/wiki/Natural_language

[28] http://en.wikipedia.org/wiki/Natural_language_processing

[29] Eugene Charniak, "Statistical Techniques for Natural Language Parsing", 1997, AI Magazine

[30] http://wiki.osafoundation.org/bin/view/Journal/DesignDateTimeWidgetNotes

[31] http://web.media.mit.edu/~hugo/montylingua/

[32] http://nltk.sourceforge.net/

[33] http://garraf.epsevg.upc.es/freeling/

[34] http://wordnet.princeton.edu/

[35] http://labix.org/python-dateutil

[36] http://www.rubyinside.com/chronic-natural-date-parsing-for-ruby-229.html

[37]http://developer.apple.com/documentation/Cocoa/Reference/Foundation/Java/Classes/NSCalendarDate.html

[38] http://code.google.com/p/parsedatetime/

[39] http://www.amk.ca/python/howto/regex/

[40] http://effbot.org/librarybook/re.htm

[41] http://docs.python.org/lib/module-time.html

[42] http://wiki.osafoundation.org/Journal/EnhancingAttributeEditors

[43] http://en.wikipedia.org/wiki/Internationalization_and_localization

[44] http://people.osafoundation.org/bkirsch/i18n/ChandlerInternationalizationProposal.html

[45] http://pyicu.osafoundation.org/

[46] http://wiki.osafoundation.org/Projects/LocalizationProject

[47] http://wiki.osafoundation.org/bin/view/Journal/StampingStoryboards

[48] http://wiki.osafoundation.org/bin/view/Journal/ParsingEmailsProposal

[49] http://wiki.osafoundation.org/bin/view/Journal/QuickItemEntry

[50] http://svn.osafoundation.org/docs/trunk/docs/specs/rel0_7/Dashboard-0.7.html

[51] http://wiki.osafoundation.org/bin/view/Projects/JottingNewItemsWorkflow

[52] http://wiki.osafoundation.org/Projects/ChandlerAutomatedTestSystem

[53] https://bugzilla.osafoundation.org

[54] http://en.wikipedia.org/wiki/Bugzilla

[55] https://bugzilla.osafoundation.org/show_bug.cgi?id=6381

[56] https://bugzilla.osafoundation.org/show_bug.cgi?id=6883

[57] https://bugzilla.osafoundation.org/show_bug.cgi?id=6906

[58] https://bugzilla.osafoundation.org/show_bug.cgi?id=6223

[59] https://bugzilla.osafoundation.org/show_bug.cgi?id=6567

Appendix A

List of Date/Times for which Units Tests have been written for ParseDateTime

This list of examples has been compiled from the unit tests TestSimpleDateTimes, TestSimpleOffsets, TestMultiple and TestUnits for the standalone Python module ParseDateTime. More date and times can be found at:

http://wiki.osafoundation.org/bin/view/Journal/DateTimeParsing.

```
5 minutes from now
5 min from now
5m from now
in 5 minutes
in 5 min
5 min from now
5 minutes
5 min
5m
5 minutes before now
5 min before now
5m before now
5 hours from noon
5 hours after noon
5 hours after 12pm
5 hours after 12 pm
5 hours after 12:00pm
5 hours after 12:00 pm
5 hours before noon
5 hours before 12pm
5 hours before 12 pm
5 hours before 12:00pm
5 hours before 12:00 pm
in 1 week
1 week from now
in 7 days
7 days from now
next week
1 week before now
7 days before now
last week
tomorrow
next day
yesterday
today
1 minute
1 minutes
1 min
```

```
1min
1 m
1m
1 hour
1 hours
1 hr
1 day
1 days
1days
1 dy
1 d
-1 day
-1 days
-1days
-1 dy
-1 d
- 1 day
- 1 days
- 1days
- 1 dy
- 1 d
1 week
1week
1 weeks
1 wk
1 w
1w
1 month
1 months
1month
1 year
1 years
1 yr
1 y
1y
11:00:00 PM
11:00 PM
11 PM
11PM
2300
23:00
11p
11pm
11:00:00 AM
11:00 AM
11 AM
11AM
1100
11:00
11a
11am
730
0730
1730
173000
08/25/2006
08.25.2006
```

```
8/25/6
8/25
8.25
08/25
morning
breakfast
lunch
evening
dinner
night
tonight
midnight
12:00:00 AM
12:00 AM
12 AM
12AM
12am
12a
0000
00:00
noon
12:00:00 PM
12:00 PM
12 PM
12PM
12pm
12p
1200
12:00
3 years 2 weeks 5 days
3years 2weeks 5days
3 y 2 w 5 d
3y 2w 5d
3y 5h 50m
3 years, 2 weeks, 5 days
3 years, 2 weeks and 5 days
3y, 2w, 5d
4pm + 3 days
4pm +3 days
4pm - 3 days
4pm -3 days
```