Out of Office (OoO) Language Specification

Prairie Resch

May 19, 2024

Introduction

Caretaking, especially over a long period of time, can be a needlessly complicated task. It's especially irritating for the person leaving their house/kids/plants/pets in someone else's hands. You, of course, might be totally familiar with what each plant needs, or each child's bedtime, or how often your dog needs to be walked, but a sitter has to have all of that explained to them in step-by-step, painstaking detail (when *were* they supposed to clean the litterbox?). For the sitter, too, it becomes a LOT to remember (was I supposed to fertilize the plants today, or water them?), especially as there are more things to take care of and more tasks to do for them.
Out of Office (OoO) provides an answer to all of this. Users can easily write a .txt file that outlines what kinds of things the sitter has to take care of (plants? kids?), the instances of the things themselves (Bob and Alice) with descriptions and actions attached to each (Bob, age 5, gets put down for a nap every day and Alice, age 8, has to be taken to soccer practice every other day). These descriptions and actions are all defined by the user in greater detail, so as to provide maximum flexibility for the user within the context of scheduling (hence why this project is a programming language and not just a program). The language, when run, outputs a .pdf file that communicates all this information in a concise, easily usable format, including a day-by-day/hour by hour checklist so that everything gets done in the way it needs to.

Design Principles

The two biggest design principles that guide OoO are flexibility and ease of use. The language needs to be easy to use for the user and produce an effective and clear manual for the sitter. The language also needs to be somewhat generalizeable for the user in order for it to be more useful than a simple hand-written note: it needs to be able to deal with cats' needs as well as it does with plants'.

Examples

A note about these examples: line breaks have been inserted where code runs off the page. This will not work in the actual code itself–everything must stay on one line.

1.
```
define date range 04/17/2023 - 05/18/2023
caretaker (Susie Bell)
location Home
action (walk Alma) 5 hours
action (walk Calvin) 3 hours
action (feed both dogs) 1 day
action wash 1 week
new dog (Calvin) breed: Corgi; age: 3; issues: Barks a lot and
  will go crazy if you sneeze;
new dog (Alma) breed: Corgi; age: 8;
```

This program creates a schedule between the dates April 17, 2024 to May 18, 2024. The schedule introduces a dog-sitter (Susie Bell) to the dogs, Alma and Calvin, and instructs the sitter to perform walk Alma every 5 hours, walk Calvin every 3 hours, feed both dogs once a day, and wash once a week. Note that wash does not specify what is being washed–it is possible that this instruction applies to another object, say a car.

2.
```
define date range 05/27/2024 15:00 - 06/02/2024
caretaker (Mom and Dad)
caretaker (Calder)
new plant (Small Pothos) pot: ceramic; size: (4 inches);
new plant (Medium Pothos) pot: ceramic; size: (8 inches);
action (water small pothos) 2 days
action (propagate jade tree cuttings) 1 year
new plant (Barrel Cactus) pot: striped ceramic; warning: (very spikey);
new plant (Prickly Pear Cactus) pot: plastic; warning: (you REALLY
  DO NOT want to touch this one trust me);
location (Mission Windowsill) sun: full; (near heater): yes; orientation:
  South;
location (Top of Closet) sun: full shade; (near heater): no;
```

This program creates a schedule that introduces a plant sitter to a pothos and tells a plant sitter to repot (implying the pothos) twice a year.

3.
```
caretaker (Mrs. McCave)
location house
new kid (Bodkin Van Horn)
new kid (Hoos-Foos)
new kid (Snimm)
new kid (Hot-Shot)
new kid (Sunny Jim)
new kid (Shadrack)
new kid (Blinkey)
new kid (Stuffy)
new kid (Stinky)
new kid (Putt-Putt)
new kid (Moon Face)
new kid (Marvin O'Gravel Balloon Face)
new kid (Ziggy)
new kid (Soggy Muff)
new kid (Buffalo Bill)
new kid (Biffalo Buff)
new kid (Sneepy)
new kid (Weepy Weed)
new kid (Paris Garters)
new kid (Harris Tweed)
new kid (Sir Michael Carmichael Zutt)
new kid (Oliver Boliver Butt)
new kid (Zanzibar Buck-Buck McFate)
action (Come into the house, Dave) 48 minutes
```

This program is aimed at Mrs. McCave (from Dr. Seuss's story *Too Many Daves*). It generates a schedule (probably assuming a start date of midnight on the date of creation and an end date of one week later, since none have been given) introducing a baby sitter to the kids Dave, Bobkin VanHorn, Snim, and the rest. It also instructs the babysitter to do the action ”Come into the house, Dave” every 48 minutes, whatever that means. In this case, though, the lack of a defined date range means that the program generates about 800 lines of LaTeX code behind the scenes and gives the poor babysitter a god-awful checklist. Hopefully Mrs. McCave isn't too exhausted from her 23 sons and notices her mistake.

Language Concepts

This language isn't too complicated. Users should know that each line is its own object (each line is an expr, essentially). Users should, however, know where to use parens (to contain strings longer than one word). Flawed input will hopefully generate usage prompts, so that will minimize stress on the user. The user also needs to know keywords ("new [Category]", "location", "caretaker", "define date range", "action") and the formatting for Fields ("([string]): ([string]); ... ([string]): ([string]);"). Overall, the code is pretty simple and users will quickly pick up on the intuition.

Formal Syntax

```
<expr>          ::= caretaker <name>
                 |  location <location>
                 |  action <action>
                 |  new <instance>
                 |  define date range <date_range>
<name>          ::= string
<location>      ::= string <fields>
<action>        ::= string * int * <timescale>
<instance>      ::= <category> * string * <fields>
<date_range>    ::= <date> * <date>
<fields>        ::= string: string; *
<date>          ::= int/int/int int:int*

<catergory> ::= Kid | Plant | Dog | Cat | House | Fish | Adult
<timescale> ::= Minute | Hour | Day | Week | Month | Year
```

Notes on formal syntax: all phrases can omit the parentheses if the string inside is a single word or number without any special characters. Additionally, the hour:minute format in the ¡date¿ construct is optional; the object will run as long as it has the month/day/year format correct.

Semantics

My program is interpreted roughly as a sequence of objects/commands that a user inputs (via a .txt file) and then outputs a .pdf file formatted nicely with all of the information in a usable, readable format. The program interprets the sequence of objects/commands and writes to a .tex file, already set up with a rough template, which then runs under the hood and produces the PDF. Combining forms in OoO appear in the form of compound objects: an Action is made up of a string, int, and a Timeframe, for example. Primitives in this language are strings and ints.

- **Category:** one of the following types of things someone could look after:
  - Kid
  - Plant
  - Dog
  - Cat
  - House
  - Fish
  - Adult
- **Timescale:** one of the following scales on which a given action could occur
  - Minute (action happens every i minutes)
  - Hour (action happens every i hours)
  - Day

  - Week
  - Month
  - Year

- **Date:** an input string of the loose format [int]/[int]/[int] [int]:[int] (hours:minutes portion optional) that gets converted to a DateTime object
- **Fields:** a string, string tuple list with elements of the form "[label]:[description];". For Fields, as with other objects, the string input can either be a single word (consisting of digits or alphabetical characters) without parentheses or a longer string enclosed in parentheses and including special characters '.', ',', '-', ''', and ' '.
- **Date_Range:** a tuple of dates expressing the start and end dates of the schedule. If multiple date ranges are defined, the program takes the most recent one.
- **Instance:** an object consisting of a Category, string, Fields tuple representing an instance of that Category with the string name and Field descriptors. As with Fields, the string input can either be a single word (consisting of digits or alphabetical characters) without parentheses or a longer string enclosed in parentheses and including special characters '.', ',', '-', ''', and ' '.
- **Action:** an object consisting of a string, int, Timescale tuple representing the action name, the frequency of the action, and at what timescale the action occurs. As with Fields, the string input can either be a single word (consisting of digits or alphabetical characters) without parentheses or a longer string enclosed in parentheses and including special characters '.', ',', '-', ''', and ' '.
- **Location:** an object consisting of a string, Fields tuple representing a location with string name and Field descriptors. As with Fields, the string input can either be a single word (consisting of digits or alphabetical characters) without parentheses or a longer string enclosed in parentheses and including special characters '.', ',', '-', ''', and ' '.
- **Name:** a string representing a name. As with Fields, the string input can either be a single word (consisting of digits or alphabetical characters) without parentheses or a longer string enclosed in parentheses and including special characters '.', ',', '-', ''', and ' '.
- **Expr:** a containing expression consisting of one line of code. Each expr can be one of the following:
  - Caretaker: the name of a one or more caretakers in charge of the care schedule (multiple definitions of Caretaker will add caretakers to the roster)
  - Location: a location, as defined above
  - Action: an action, as defined above
  - new Instance: a new instance, as defined above
  - define date range Date_Range: a new Date_Range, as defined above

Remaining Work

At this point, I'm still trying to figure out how to heck to get my code to run the LaTeX code behind-the-scenes and just generate a PDF for the user from their input code. That's the biggest hurdle. If you're reading this, I haven't figured it out.

Other remaining work for this project includes: increased robustness to user input and error-specific error messages; ability to create new user-defined categories beyond just Kid, Plant, Dog, etc.; connection between actions and categories (a defined walk action would be applied to only dogs, because why would you try to walk a plant?) and therefore assignment between individual objects (all dogs need to be walked every 4 hours, so therefore Calvin must be walked every 4 hours); and an ability to designate times of day for certain actions and maybe even days of the week (Betsy has to go to daycare at 08:00 every weekday).